TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Desarrollo de un videojuego FPS

Autor: **Rubén Sanabria Díaz**

Grado: **Ingeniería Informática**

Director: Carlos Javier Ogayar Anguita
Departamento del director: Informática

Fecha: junio de 2024

Licencia CC

CREA

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Ogayar Anguita, Carlos Javier, tutor del Trabajo Fin de Grado titulado: '**Desarrollo de un videojuego FPS**', que presenta Don Sanabria Díaz, Rubén, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, 06 de 2024

El alumno: El tutor:

Sanabria Díaz, Rubén Ogayar Anguita, Carlos Javier

(Página intencionalmente en blanco)

Agradecimientos

Este proyecto ha sido realizado con esfuerzo sudor y lágrimas, pero jamás habría podido ser realizado sin el apoyo de mis familiares, amigos y conocidos, que me han ayudado a salir adelante en los momentos más difíciles y a no rendirme nunca.

Agradecer a mis padres, que nunca desconfiaron de mí, sabiendo que sería capaz de esto y más. Que sé que me quieren con locura y que han hecho todo lo posible y más para que pueda llegar hasta aquí. No me podría imaginar un mundo sin ellos, sin su apoyo ni sus consejos. Gracias papis, me gustaría devolveros todo lo que habéis hecho por mí, pero creo que jamás podré devolvéroslo, puesto que es imposible mover el mundo con el meñique…

Agradecer a mi pareja, la persona más cercana que tengo en mi vida y la que más me ha ayudado, no solo en este proyecto, sino en todo a cerca de mi vida. Siempre demostrando que está ahí tanto en las buenas como en las malas y ayudándome a superar todo. Sin ella, todo hubiera sido muy diferente… te quiero.

Agradecer a todos mis amigos y compañeros de clase, que aportaron su granito de arena siendo testers de todos los incrementos y cosas que quería que probasen y dándome sus feedbacks para hacer un mejor proyecto.

Agradecer también a mi tutor, que ha estado ahí siempre que lo he necesitado, revisando el trabajo además de recomendarme las cosas que se podrían añadir o mejorar.

Gracias a todos, por todo.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Tecnologías de la Información
Mención (solo TFG)	Sistemas Gráficos
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Sanabria Díaz, Rubén
Fecha de asignación	20/03/2024
Descripción corta	Un FPS (First Person Shooter) es un género de videojuegos en el cual el protagonista usa armas de fuego y tanto la visión del mundo como la interacción con el mismo se ve desde una perspectiva de primera persona. En este trabajo se implementará un videojuego de este tipo haciendo uso de la herramienta Unity, la cual es un motor de videojuegos, software que permite el diseño, creación y funcionamiento de un videojuego o proyecto tanto en tres dimensiones como en dos dimensiones. El objetivo de este trabajo será tener un juego al que se pueda recurrir para disfrutar de tu tiempo libre.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

1 Especificación del trabajo 25

1.1 Introducción 25
1.2 Objetivos del trabajo 26
1.3 Antecedentes y estado del arte 26
1.4 Requisitos iniciales 28
1.4.1 Audiencia 28
1.4.2 Herramientas utilizadas 30
1.4.2.1 Motor de videojuegos 30
1.4.2.2 Herramienta de modelado y animación 31
1.4.2.3 Herramienta de animación 33
1.4.2.4 Herramienta de control de versiones 34
1.4.2.5 Herramienta de diseño 35
1.4.2.6 Herramienta de modelado y diseño software 36
1.4.2.7 Herramienta de gestión de proyectos 37
1.4.3 Plataformas disponibles 38
1.5 Alcance 38
1.6 Hipótesis y restricciones 39
1.7 Estudio de alternativas y viabilidad 39
1.7.1 Género del videojuego 40
1.7.1.1 FPS vs TPS 41
1.7.1.2 FPS vs RTS 41
1.7.1.3 FPS vs RPG 42
1.7.1.4 FPS vs Battle Royale 42
1.7.1.5 FPS vs Simuladores 42
1.7.2 Software 43
1.7.2.1 Unity vs Unreal 43
1.7.2.2 Unity vs Godot 44
1.7.3 Hardware 45
1.8 Descripción de la solución propuesta 45
1.9 Tecnologías utilizadas 46
1.10 Metodología de desarrollo de software 47
1.10.1 Scrum VS Desarrollo Lean 49
1.10.2 Scrum VS Desarrollo en cascada 50
1.10.3 Scrum VS Desarrollo en espiral 51
1.11 Análisis de riesgos 51
1.11.1 Riesgos técnicos 52
1.11.2 Riesgos de gestión 53
1.12 Estimación del tamaño y esfuerzo 54
1.13 Planificación temporal 58
1.14 Presupuesto 60
1.14.1 Activos fijos 60
1.14.2 Recursos humanos 61
1.14.3 Gastos indirectos 62
1.14.4 Licencias 62
1.14.5 Conclusión 62

2 Diseño inicial 63

2.1 Especificaciones del sistema 64
2.1.1 Requisitos funcionales 65
2.1.1.1 Sistema de menús e interfaz 65
2.1.1.2 Gestión del jugador 66
2.1.1.3 Sistema de combate 67
2.1.1.4 NPCs 68
2.1.1.5 Sonidos y música 69
2.1.2 Requisitos no funcionales 69
2.1.2.1 Rendimiento 70
2.1.2.2 Usabilidad 70
2.1.2.3 Estabilidad 70
2.1.2.4 Compatibilidad 71
2.1.2.5 Mantenimiento 71
2.1.2.6 Retroalimentación 71
2.2 Análisis y diseño del sistema 72
2.2.1 Diagrama de casos de uso 72
2.2.1.1 Autores 72
2.2.1.2 Ilustración del diagrama de casos de uso 74
2.2.2 Casos de uso 75
2.2.2.1 Movimiento del personaje 77
2.2.2.2 Opciones de los menús 80
2.2.2.3 Interacción 83
2.2.2.4 Combate 86
2.2.2.5 Funciones del sistema 93
2.2.3 Diseño arquitectónico 94
2.2.4 Diseño de la interfaz 95
2.2.4.1 Menú principal 96
2.2.4.2 Menú de opciones 99
2.2.4.3 Pantalla de muerte 102
2.2.4.4 Interfaz del jugador 105
2.2.4.1 Diagrama de flujo de navegación 107

3 Desarrollo 108

3.1 Iteraciones 109
3.1.1 Primera iteración 110
3.1.1.1 Introducción 110
3.1.1.2 Tareas 110
3.1.1.3 Desarrollo del sprint 112
3.1.1.4 Diagrama Burndown 113
3.1.1.5 Diagrama de velocidad 115
3.1.1.6 Comentarios finales 116
3.1.2 Segunda iteración 119
3.1.2.1 Introducción 119
3.1.2.2 Tareas 119
3.1.2.3 Desarrollo del sprint 121
3.1.2.4 Diagrama Burndown 122
3.1.2.5 Diagrama de velocidad 123
3.1.2.6 Comentarios finales 125
3.1.3 Tercera iteración 134
3.1.3.1 Introducción 134

3.1.3.2 Tareas ... 134
3.1.3.3 Desarrollo del sprint ... 136
3.1.3.4 Diagrama Burndown ... 137
3.1.3.5 Diagrama de velocidad ... 139
3.1.3.6 Comentarios finales ... 140
3.1.4 Cuarta iteración ... 144
3.1.4.1 Introducción ... 144
3.1.4.2 Tareas ... 144
3.1.4.3 Desarrollo del sprint ... 146
3.1.4.4 Diagrama Burndown ... 147
3.1.4.5 Diagrama de velocidad ... 148
3.1.4.6 Comentarios finales ... 150
3.1.5 Quinta iteración ... 156
3.1.5.1 Introducción ... 156
3.1.5.2 Tareas ... 156
3.1.5.3 Desarrollo del sprint ... 158
3.1.5.4 Diagrama Burndown ... 159
3.1.5.5 Diagrama de velocidad ... 161
3.1.5.6 Comentarios finales ... 162
3.2 Funcionalidades desarrolladas ... 168
3.2.1 “GameManager“ ... 168
3.2.1.1 Pruebas ... 175
3.2.2 “GameLogic“ ... 176
3.2.2.1 Pruebas ... 180
3.2.3 “SoundManager“ ... 180
3.2.3.1 Pruebas ... 187
3.2.4 Entorno ... 189
3.2.4.1 Nave del jugador ... 189
3.2.4.2 La cúpula ... 195
3.2.4.3 Nave espacial ... 200
3.2.4.4 Exterior ... 245
3.2.5 Personaje ... 254
3.2.5.1 Animación ... 255
3.2.5.2 Scripts asociados ... 258
3.2.6 Cámaras ... 265
3.2.6.1 Cámara del jugador ... 265
3.2.6.2 Cámara del menú ... 273
3.2.7 Arma ... 278
3.2.8 Enemigos ... 292
3.2.8.1 Banana-man ... 292
3.2.8.2 Dron de combate ... 296
3.2.8.3 Dron terrestre ... 304
3.2.8.4 Dron aéreo ... 311
3.2.8.5 StarLord ... 313
3.2.8.6 Pilar de invocación de enemigos ... 324
3.2.9 Interacción ... 330
3.2.9.1 Interacción con NPCs aliados ... 330
3.2.9.2 Interacción con elementos del entorno e ítems ... 332
3.2.10 NPCs aliados ... 335
3.2.10.1 Kyle ... 336

3.2.10.2 Freddy 347
3.2.11 Menús y acciones 350
3.2.12 Cambio de dificultad 354
3.2.13 Ítems 355
3.2.13.1 Poción de curación 356
3.2.13.2 Poción de velocidad 357
3.2.13.3 Poción de ácido 359
3.2.13.4 Caja de munición 369
3.2.13.5 Líquidos 372
3.2.13.6 Cambio entre ítems y el arma 380
3.2.14 Aparición aleatoria de ítems 384
3.2.15 Iluminación 389
3.2.16 Créditos 393
3.3 Herramientas usadas 398
3.3.1 ProBuilder 398
3.3.2 Shader Graph 401
3.3.3 Profiler 401

4 Conclusiones y trabajos futuros 403

5 Apéndices 405

5.1 Guía original del Trabajo Fin de Título 405
5.2 Instalación y configuración del sistema 410
5.3 Manuales de usuario 411

6 Definiciones y abreviaturas 412

7 Bibliografía 414

Índice de ilustraciones

Ilustración 1.1; Ilustración de la carátula Doom 26
Ilustración 1.2; Ilustración de una partida de Doom 27
Ilustración 1.3; Diferentes iconos de PEGI 29
Ilustración 1.4; Logo de Unity 30
Ilustración 1.5; Logo de Blender 32
Ilustración 1.6; Animación hecha con la herramienta Blender 33
Ilustración 1.7; Logo de Mixamo 33
Ilustración 1.8; Logo de Mixamo 34
Ilustración 1.9; Logo de UXPin 35
Ilustración 1.10; Logo de VisualParadigm 36
Ilustración 1.11; Logo de GanttProject 37
Ilustración 1.12; Diagrama de Gantt 59
Ilustración 2.1; Diagrama de casos de uso 75
Ilustración 2.2; Diseño arquitectónico del proyecto 94
Ilustración 2.3; Sketch del menú principal 96
Ilustración 2.4; Wireframe del menú principal, con rejilla 97
Ilustración 2.5; Wireframe del menú principal, sin rejilla 97
Ilustración 2.6; Mockup del menú principal, con rejilla 98
Ilustración 2.7; Mockup del menú principal, sin rejilla 98
Ilustración 2.8; Sketch del menú de opciones 99
Ilustración 2.9; Wireframe del menú de opciones, con rejilla 100
Ilustración 2.10; Wireframe del menú de opciones, sin rejilla 100
Ilustración 2.11; Mockup del menú de opciones, con rejilla 101
Ilustración 2.12; Mockup del menú de opciones, sin rejilla 101
Ilustración 2.13; Sketch de la pantalla de muerte 102
Ilustración 2.14; Wireframe de la pantalla de muerte, con rejilla 103
Ilustración 2.15; Wireframe de la pantalla de muerte, sin rejilla 103
Ilustración 2.16; Mockup de la pantalla de muerte, con rejilla 104
Ilustración 2.17; Mockup de la pantalla de muerte, sin rejilla 104
Ilustración 2.18; Sketch de la interfaz del jugador 105
Ilustración 2.19; Wireframe de la interfaz del jugador, con rejilla 105
Ilustración 2.20; Wireframe de la interfaz del jugador, sin rejilla 106
Ilustración 2.21; Mockup de la interfaz del jugador, con rejilla 106
Ilustración 2.22; Mockup de la interfaz del jugador, sin rejilla 107
Ilustración 2.23; Diagrama de flujo de navegación entre interfaces 108
Ilustración 3.1; Diagrama Burndown del sprint1 114
Ilustración 3.2; Diagrama de velocidad del sprint1 116
Ilustración 3.3; Skybox de la escena 117
Ilustración 3.3; Plantas de la nave espacial enemiga 118
Ilustración 3.4; Personaje principal y cámara 118
Ilustración 3.5; Diagrama Burndown del sprint2 123
Ilustración 3.1; Diagrama de velocidad del sprint2 124
Ilustración 3.7; Modelo del arma del jugador 125
Ilustración 3.8; Interfaz del usuario, vista general 126

Ilustración 3.9; Interfaz del usuario, panel de dialogo y de interacción 126
Ilustración 3.10; Interfaz del usuario, panel de tareas.....127
Ilustración 3.11; Interfaz del usuario, panel de munición 128
Ilustración 3.12; Interfaz del usuario, panel de pociones 129
Ilustración 3.13; Interfaz del usuario, puntero y vida del jugador 130
Ilustración 3.14; Interfaz del usuario, bala y arma.....131
Ilustración 3.15; Menú principal 132
Ilustración 3.16; Menú de opciones 132
Ilustración 3.18; Pantalla de muerte 133
Ilustración 3.19; Caja de munición, sin abrir 133
Ilustración 3.20; Caja de munición, abierta 134
Ilustración 3.20; Diagrama Burndown del sprint3.....138
Ilustración 3.22; Diagrama de velocidad del sprint3.....140
Ilustración 3.23; Dron aéreo 141
Ilustración 3.24; Dron terrestre 141
Ilustración 3.25; Dron de combate.....142
Ilustración 3.26; Pillar de invocación de enemigos 142
Ilustración 3.27; StarLord 143
Ilustración 3.28; Botón de interacción.....143
Ilustración 3.28; Diagrama Burndown del sprint4.....148
Ilustración 3.29; Diagrama de velocidad del sprint4.....149
Ilustración 3.30; Kyle 150
Ilustración 3.31; Freddy 151
Ilustración 3.32; Darth Vader 152
Ilustración 3.20; Poción de curación 153
Ilustración 3.34; Poción de ácido 153
Ilustración 3.35; Poción de ácido, trayectoria de lanzamiento.....154
Ilustración 3.36; Poción de ácido, lanzamiento 154
Ilustración 3.37; Poción de ácido, piscina de ácido.....155
Ilustración 3.38; Poción de velocidad 155
Ilustración 3.39; Batería.....156
Ilustración 3.40; Diagrama Burndown del sprint5.....160
Ilustración 3.41; Diagrama de velocidad del sprint5.....162
Ilustración 3.42; Interacción con poción de curación 163
Ilustración 3.43; Interacción con poción de ácido 163
Ilustración 3.44; Interacción con poción de velocidad.....164
Ilustración 3.45; Interacción con caja de munición.....164
Ilustración 3.46; Interacción con batería 165
Ilustración 3.47; Animación de Darth Vader.....165
Ilustración 3.48; Dialogo con Kyle 166
Ilustración 3.48; Dialogo con Freddy 166
Ilustración 3.48; Tierra y Luna 167
Ilustración 3.48; Sol.....167
Ilustración 3.52; Mensaje de recarga de munición.....170
Ilustración 3.53; Luz de la lintera del arma 171
Ilustración 3.54; Mensaje de nueva tarea 172

Ilustración 3.55; Pantalla de muerte tras morir 177
Ilustración 3.56; Menú de opciones tras pausar partida.....178
Ilustración 3.57; Menú de inicio 179
Ilustración 3.58; Ciclo de dificultad del juego 179
Ilustración 3.59; SoundManager.....181
Ilustración 3.60; Nave del jugador, componentes 190
Ilustración 3.61; Nave del jugador, elementos 190
Ilustración 3.62; Nave del jugador, puntos de control de la curva de Bèzier 191
Ilustración 3.63; Nave del jugador, curva de Bèzier.....192
Ilustración 3.64; Nave del jugador, apertura de cápsula 192
Ilustración 3.65; Nave del jugador, componentes de cápsula 193
Ilustración 3.66; Cúpula, componentes del cristal.....196
Ilustración 3.67; Cúpula, componentes de la base del botón central 196
Ilustración 3.68; Cúpula, elementos de la base del botón central 197
Ilustración 3.69; Cúpula, componentes del botón central.....197
Ilustración 3.70; Plano de “out of bounds” 198
Ilustración 3.71; Nave espacial, componentes.....200
Ilustración 3.72; Nave espacial, apertura de puerta.....201
Ilustración 3.73; Primera planta, primera sala.....203
Ilustración 3.74; Primera planta, muro invisible 204
Ilustración 3.75; Primera planta, ascensor.....205
Ilustración 3.76; Primera planta, componentes del elevador.....206
Ilustración 3.77; Primera planta, rotacion de puerta de elevador 207
Ilustración 3.78; Primera planta, collider de la primera puerta del elevador 208
Ilustración 3.79; Primera planta, componentes del botón del elevador 208
Ilustración 3.80; Primera planta, habitación de Darth Vader 212
Ilustración 3.81; Primera planta, componentes de Darth Vader 213
Ilustración 3.82; Primera planta, AnimatorController de DarthVader.....213
Ilustración 3.83; Primera planta, collider de Darth Vader 214
Ilustración 3.84; Primera planta, tanques de ácido 215
Ilustración 3.85; Primera planta, componentes del ácido de los tanques de ácido.....216
Ilustración 3.86; Segunda planta, sala intermedia 217
Ilustración 3.87; Segunda planta, sala izquierda 218
Ilustración 3.88; Segunda planta, sala izquierda, batería recogible 218
Ilustración 3.89; Segunda planta, sala izquierda, sección de recompensas.....219
Ilustración 3.90; Segunda planta, láser de la sección de recompensa.....220
Ilustración 3.91; Segunda planta, collider de la sala de recompensa.....221
Ilustración 3.92; Segunda planta, collider de la sala izquierda.....221
Ilustración 3.93; Segunda planta, sala lateral 224
Ilustración 3.94; Segunda planta, cascada de ácido.....225
Ilustración 3.95; Segunda planta, efecto de la cascada de ácido.....225
Ilustración 3.96; Segunda planta, shader graph completo de la cascada de ácido 226
Ilustración 3.97; Segunda planta, primera parte del shader graph de la cascada de ácido.227
Ilustración 3.98; Segunda planta, segunda parte del shader graph de la cascada de ácido 228

Ilustración 3.99; Segunda planta, primera parte explicación del efecto de la cascada de ácido 229
Ilustración 3.100; Segunda planta, segunda parte explicación del efecto de la cascada de ácido 230
Ilustración 3.101; Segunda planta, puerta de la sala de StarLord.....231
Ilustración 3.101; Segunda planta, panel de control de la puerta de la sala de StarLord 232
Ilustración 3.103; Segunda planta, no se puede interactuar con panel si jugador no tiene batería seleccionada 232
Ilustración 3.104; Segunda planta, se puede interactuar con panel si jugador tiene batería seleccionada 233
Ilustración 3.105; Segunda planta, elementos del panel de control de la puerta de la sala de StarLord 233
Ilustración 3.106; Segunda planta, puerta de la entrada de la sala de StarLord abierta 234
Ilustración 3.107; Segunda planta, sala de StarLord 234
Ilustración 3.108; Segunda planta, collider de la sala de StarLord.....235
Ilustración 3.109; Segunda planta, sala de StarLord después de que jugador pase por el collider de la sala.....236
Ilustración 3.110; Segunda planta, puerta de salida de la sala de StarLord.....236
Ilustración 3.111; Segunda planta, sala intermedia a la tercera planta 241
Ilustración 3.112; Tercera planta, primera sala.....242
Ilustración 3.113; Tercera planta, hangar 242
Ilustración 3.114; Tercera planta, collider del hangar 243
Ilustración 3.115; Tercera planta, portal del hangar.....243
Ilustración 3.116; Tierra, componentes.....245
Ilustración 3.117; Luna, componentes.....246
Ilustración 3.118; Sol, componentes.....247
Ilustración 3.119; Sol, efecto visual (VFX) 248
Ilustración 3.120; Sol, efecto visual (VFX), primera parte 249
Ilustración 3.121; Sol, efecto visual (VFX), segunda parte.....250
Ilustración 3.122; Cohetes de propulsión de la nave 251
Ilustración 3.123; Cohetes de propulsión de la nave, sistema de partículas 252
Ilustración 3.124; Cohetes de propulsión de la nave, activados.....253
Ilustración 3.125; Elementos del personaje 254
Ilustración 3.126; Componentes del personaje.....254
Ilustración 3.127; AnimatorController del personaje 255
Ilustración 3.128; Walking Blend Tree del AnimatorController del personaje 256
Ilustración 3.129; Running Blend Tree del AnimatorController del personaje 257
Ilustración 3.130; Pseudocódigo del script PlayerMovement 260
Ilustración 3.131; Pseudocódigo del script LifeBarLogic 263
Ilustración 3.132; Elementos de la cámara principal.....265
Ilustración 3.133; Arma apuntando.....266
Ilustración 3.134; Arma sin apuntar 267
Ilustración 3.135; Componentes de la cámara principal.....268
Ilustración 3.136; Colisión de la bala del arma sin apuntar 269
Ilustración 3.137; Colisión de la bala del arma apuntando.....269
Ilustración 3.138; Pseudocódigo del script CameraLook 271

Ilustración 3.139; Menú principal de Minecraft. Imagen obtenida de Minecraft Wiki. © Mojang Studios. 274
Ilustración 3.140; Componente Alfa del color del canvas del menú principal. 274
Ilustración 3.141; Posiciones de la cámara del menú principal 275
Ilustración 3.142; Componentes de la cámara del menú principal 276
Ilustración 3.143; Resultado de combinar menú principal con su cámara....277
Ilustración 3.144; Modelo del arma....278
Ilustración 3.145; Elementos del arma....279
Ilustración 3.146; Componentes del arma 279
Ilustración 3.147; Componente TrailRenderer de la bala del arma 282
Ilustración 3.148; Pseudocódigo del script WeaponRaycasting 283
Ilustración 3.149; Componentes del enemigo Banana-man....292
Ilustración 3.150; AnimatorController del enemigo Banana-man 293
Ilustración 3.151; Elementos del enemigo Banana-man 294
Ilustración 3.152; Componentes del enemigo Dron de combate 296
Ilustración 3.153; AnimatorController del enemigo Dron de combate 297
Ilustración 3.154; Área de detección del enemigo Dron de combate 299
Ilustración 3.155; Eventos dentro de la animación “Punching_Bag” del enemigo Dron de combate 299
Ilustración 3.156; Elementos de las chispas del enemigo Dron de combate 300
Ilustración 3.157; Posicion de las chispas del enemigo Dron de combate 300
Ilustración 3.158; Componentes del enemigo Dron terrestre 304
Ilustración 3.159; Dron terrestre cuando ha detectado al jugador 305
Ilustración 3.160; Pseudocódigo del script ArmedEnemyFollowsPlayer 307
Ilustración 3.161; Componentes del enemigo Dron aéreo 311
Ilustración 3.162; Dron aéreo cuando ha detectado al jugador 311
Ilustración 3.163; Componentes del enemigo StarLord 313
Ilustración 3.164; AnimatorController del enemigo StarLord 314
Ilustración 3.165; Explosión del enemigo StarLord 316
Ilustración 3.166; Muerte del enemigo StarLord 316
Ilustración 3.167; Radios de detección del enemigo StarLord 317
Ilustración 3.168; Eventos de la animacion “GunPlay” del enemigo StarLord 318
Ilustración 3.169; Eventos de la animacion “UpperCut” del enemigo StarLord 319
Ilustración 3.170; Disparo del enemigo StarLord 319
Ilustración 3.171; Pseudocódigo del script “BossController” 321
Ilustración 3.172; Cambios de materiales de StarLord en función de su estado de detección 324
Ilustración 3.173; Componentes del enemigo Pilar de invocación de enemigos 325
Ilustración 3.174; Área de detección del enemigo Pilar de invocación de enemigos....326
Ilustración 3.175; Explicación de la invocación de enemigos del enemigo Pilar de invocación de enemigos....327
Ilustración 3.176; Casos de invocación de enemigos del enemigo Pilar de invocación de enemigos....327
Ilustración 3.177; Pseudocódigo del script “SpawnEnemies” 329
Ilustración 3.178; Interacción con un NPC 331
Ilustración 3.179; Interacción con poción de curación 333

Ilustración 3.180; Interacción con batería de las salas 333
Ilustración 3.181; Interacción con panel de control de la sala de StarLord.....334
Ilustración 3.182; No se puede interactuar con panel de control.....334
Ilustración 3.183; Componentes de Kyle 336
Ilustración 3.184; AnimatorController de Kyle.....337
Ilustración 3.185; Dialogo con Kyle 338
Ilustración 3.186; Kyle quiere hablar con el jugador 339
Ilustración 3.187; Tarea de disparar a banana-man 340
Ilustración 3.188; Tarea de eliminar enemigos 340
Ilustración 3.189; Recompensa del final del tutorial.....341
Ilustración 3.190; Componentes de Freddy 347
Ilustración 3.191; AnimatorController de Freddy.....348
Ilustración 3.192; Diálogo con Freddy 349
Ilustración 3.193; Botones del menú principal 351
Ilustración 3.194; Botones y sliders del menú de opciones.....351
Ilustración 3.195; Botones de la pantalla de muerte 352
Ilustración 3.196; Botón de cambio de dificultad del menú principal 354
Ilustración 3.197; Componentes del PotionThrower 360
Ilustración 3.198; Trayectoria de lanzamiento de la poción de ácido 361
Ilustración 3.199; Pseudocódigo del script PotionThrower.....363
Ilustración 3.200; Componentes de la poción de ácido lanzable.....364
Ilustración 3.201; Componentes de la piscina de ácido creada 365
Ilustración 3.202; Componentes de la caja de munición 369
Ilustración 3.203; Apertura de la tapa de la caja de munición.....370
Ilustración 3.204; Casos de líquido de la poción de curación 372
Ilustración 3.205; Casos de líquido de la poción de velocidad.....373
Ilustración 3.206; Componentes del líquido de la poción de velocidad 373
Ilustración 3.207; ShaderGraph del líquido de las pociones 374
Ilustración 3.208; ShaderGraph del líquido de las pociones, primera parte 375
Ilustración 3.209; ShaderGraph del líquido de las pociones, segunda parte.....376
Ilustración 3.210; ShaderGraph del líquido de las pociones, tercera parte 376
Ilustración 3.211; Gravedad en diferentes casos del líquido de la poción de curación.....377
Ilustración 3.212; ShaderGraph del líquido de las pociones, cuarta parte 377
Ilustración 3.213; ShaderGraph del líquido de las pociones, quinta parte.....378
Ilustración 3.214; Componentes del líquido de la poción de curación.....379
Ilustración 3.215; Pseudocódigo del script PotionSwitching 382
Ilustración 3.216; Posibles ubicaciones de reaparición de items en la segunda planta.....385
Ilustración 3.217; Componentes del script SpawnItems.....385
Ilustración 3.218; Estructura de la clase ItemCollecionData 386
Ilustración 3.219; Ejemplo de instancia de ScriptableObject.....386
Ilustración 3.220; Componentes del bloque de luz del hueco del ascensor 389
Ilustración 3.221; Componentes de la luz del bloque de luz del hueco del ascensor 390
Ilustración 3.222; Panel de luz de las salas.....390
Ilustración 3.223; Panel de luz de las salas desglosado 391
Ilustración 3.224; Componentes de la luz del panel de luz de las salas.....391
Ilustración 3.225; Sol.....392

Ilustración 3.226; Componentes de la luz del Sol 392
Ilustración 3.227; CrawlText de los créditos 394
Ilustración 3.220; Componentes de la cámara de los créditos 394
Ilustración 3.229; Vista de rotación del texto 395
Ilustración 3.230; Botón de salir del juego 396
Ilustración 3.231; Funcionalidades de ProBuilder 398
Ilustración 3.232; Huecos del hangar 399
Ilustración 3.233; Proceso de creación de PolyShape hecha con ProBuilder 399
Ilustración 3.234; PolyShape hecha con ProBuilder 400
Ilustración 3.235; PolyShape hecha con ProBuilder rellena los huecos del hangar 400
Ilustración 3.236; Ejemplo de ShaderGraph 401
Ilustración 3.237; Ejemplo de rendimiento de una ejecución capturado por Profiler 402
Ilustración 3.238; Detalles del rendimiento de una ejecución capturado por Profiler.....402
Ilustración 7.1; apéndice: propuesta del TFG 406
Ilustración 7.2; apéndice: datos del TFG y tutor asignado 407
Ilustración 7.3; apéndice: objetivos del TFG y alumno asignado 408
Ilustración 7.4; apéndice: metodología y documentos y formatos de entrega del TFG 409
Ilustración 7.5; carpeta ejectuable del juego.....410
Ilustración 7.6; teclas de dispositivos de entrada.....411

Índice de tablas

Tabla 1.1; Relación Story points/horas de trabajo 56
Tabla 1.2; Estimación de tareas 57
Tabla 1.3; Precios de activos....60
Tabla 1.4; Precios de recursos humanos 61
Tabla 1.5; Presupuesto total....62
Tabla 2.1; Plantilla de la tabla de requisitos del sistema....64
Tabla 2.2; RF01: Iniciar juego....65
Tabla 2.3; RF02: Pausar juego....65
Tabla 2.4; RF03: Reiniciar juego 65
Tabla 2.5; RF04: Salir del juego 66
Tabla 2.6; RF05: Visualizar información de la partida....66
Tabla 2.7; RF06: Control del movimiento del jugador 66
Tabla 2.8; RF07: Complementos del movimiento del jugador....66
Tabla 2.9; RF08: Interacción 67
Tabla 2.10; RF09: Sistema de salud....67
Tabla 2.11; RF10: Manejo de armas 67
Tabla 2.12; RF11: Sistema de apuntado 67
Tabla 2.13; RF12: Disparar 68
Tabla 2.14; RF13: Comportamiento de los enemigos....68
Tabla 2.15; RF14: Ataque de los enemigos....68
Tabla 2.16; RF15: Variedad de enemigos 68
Tabla 2.17; RF16: Interactuar con NPCs aliados....69
Tabla 2.18; RF17: Efectos de sonido....69
Tabla 2.19; RF18: Efectos de sonido....69
Tabla 2.20; RNF01: FPS 70
Tabla 2.21; RNF02: Intuitividad 70
Tabla 2.22; RNF03: Documentación....70
Tabla 2.23; RNF04: Tolerancia a fallos 70
Tabla 2.24; RNF05: Pruebas del sistema 71
Tabla 2.25; RNF06: Compatibilidad con otras plataformas....71
Tabla 2.26; RNF07: Modularidad....71
Tabla 2.27; RNF08: Aclaración de cambios en el juego 72
Tabla 2.28; Plantilla de la tabla del autor....72
Tabla 2.29; Aut01: Sistema 73
Tabla 2.30; Aut02: Jugador 74
Tabla 2.31; Plantilla de tabla de casos de uso....76
Tabla 2.32; CU01: Jugar 77
Tabla 2.33; CU02: Mover al personaje 78
Tabla 2.34; CU03: Correr 79
Tabla 2.35; CU04: Saltar 79
Tabla 2.36; CU05: Pausar el juego....80
Tabla 2.37; CU06: Salir del juego....81
Tabla 2.38; CU07: Reanudar el juego 81
Tabla 2.39; CU08: Modificar el volumen....82

Tabla 2.40; CU09: Modificar la sensibilidad....83
Tabla 2.41; CU10: Interactuar con el entorno....84
Tabla 2.42; CU11: Interactuar con objetos....85
Tabla 2.43; CU12: Interactuar con NPCs....86
Tabla 2.44; CU13: Disparar....87
Tabla 2.45; CU14: Usar objetos....88
Tabla 2.46; CU15: Lanzar objetos....89
Tabla 2.47; CU16: Mejorar habilidades del personaje....90
Tabla 2.48; CU17: Recuperar vida del personaje....91
Tabla 2.49; CU18: Dañar al enemigo....92
Tabla 2.50; CU19: Recoger objetos....93
Tabla 2.51; CU20: Cargar recursos....94
Tabla 3.1; Estimación de tareas del sprint1....112
Tabla 3.2; Información sobre el sprint1....113
Tabla 3.3; Detalles del diagrama Burndown del sprint1....114
Tabla 3.4; Detalles del diagrama de velocidad del sprint1....115
Tabla 3.5; Estimación de tareas del sprint2....120
Tabla 3.6; Información sobre el sprint2....122
Tabla 3.7; Detalles del diagrama Burndown del sprint2....122
Tabla 3.8; Detalles del diagrama de velocidad del sprint2....124
Tabla 3.9; Estimación de tareas del sprint3....136
Tabla 3.10; Información sobre el sprint3....137
Tabla 3.11; Detalles del diagrama Burndown del sprint3....138
Tabla 3.12; Detalles del diagrama de velocidad del sprint3....139
Tabla 3.13; Estimación de tareas del sprint4....145
Tabla 3.14; Información sobre el sprint4....146
Tabla 3.15; Detalles del diagrama Burndown del sprint4....147
Tabla 3.16; Detalles del diagrama de velocidad del sprint4....149
Tabla 3.17; Estimación de tareas del sprint5....158
Tabla 3.18; Información sobre el sprint5....159
Tabla 3.19; Detalles del diagrama Burndown del sprint5....160
Tabla 3.20; Detalles del diagrama de velocidad del sprint5....161
Tabla 3.21; Plantilla de la tabla de pruebas....168
Tabla 3.22; Test_01_01: apertura de cápsula....175
Tabla 3.23; Test_01_02: aparición del mensaje de nueva tarea....176
Tabla 3.24; Test_01_03: parpadeo del color de los mensajes....176
Tabla 3.25; Test_02_01: aparición de las interfaces....180
Tabla 3.26; Sonidos del juego....183
Tabla 3.27; Música del juego....184
Tabla 3.28; Test_03_01: sonidos adecuados....187
Tabla 3.29; Test_03_02: música adecuada....188
Tabla 3.30; Test_03_03: momento de los sonidos....188
Tabla 3.31; Test_03_04: momento de la música....188
Tabla 3.32; Test_03_05: volumen de los sonidos....188
Tabla 3.33; Test_03_06: volumen de la música....189
Tabla 3.34; Test_04_01: apertura de cápsula....194

Tabla 3.35; Test_04_02: llegada de la nave 194
Tabla 3.36; Test_04_03: movimiento de la nave 194
Tabla 3.37; Test_04_04: ruta de movimiento de la nave 195
Tabla 3.38; Test_04_05: oscilación de la nave 195
Tabla 3.39; Test_04_06: pulsación del botón central 199
Tabla 3.40; Test_04_07: inicio de fuerza de los motores 199
Tabla 3.41; Test_04_08: regreso al caer 200
Tabla 3.42; Test_04_09: llegada de la nave espacial 202
Tabla 3.43; Test_04_10: apertura de la puerta de la nave espacial 202
Tabla 3.44; Test_04_11: fin de la fuerza de los motores 202
Tabla 3.45; Test_04_12: transparencia del muro invisible 205
Tabla 3.46; Test_04_13: llegada del elevador 209
Tabla 3.47; Test_04_14: movimiento del personaje dentro del elevador 210
Tabla 3.48; Test_04_15: límites del elevador 210
Tabla 3.49; Test_04_16: rotación de las puertas del elevador 210
Tabla 3.50; Test_04_17: Puerta de salida se abre al terminar movimiento del elevador 210
Tabla 3.51; Test_04_18: movimiento del elevador 211
Tabla 3.52; Test_04_19: puerta de la entrada al elevador 211
Tabla 3.53; Test_04_20: movimiento del botón al ser pulsado 211
Tabla 3.54; Test_04_21: cierre de la puerta del elevador al pulsar botón 212
Tabla 3.55; Test_04_22: inicio de animcación 214
Tabla 3.56; Test_04_23: flujo de animación 215
Tabla 3.57; Test_04_24: daño producido al jugador 216
Tabla 3.58; Test_04_25: ruta para llegar al ácido 217
Tabla 3.59; Test_04_26: límite de recogida de recompensas 222
Tabla 3.60; Test_04_27: momento de desactivación de los láseres 222
Tabla 3.61; Test_04_28: velocidad de desactivación de los láseres 223
Tabla 3.62; Test_04_29: cumplir tarea antes de salir de la sala 223
Tabla 3.63; Test_04_30: recoger batería del panel de control 223
Tabla 3.64; Test_04_31: recogida de las recompensas 224
Tabla 3.65; Test_04_32: apertura de cascada de ácido 231
Tabla 3.66; Test_04_33: colocación de la batería 237
Tabla 3.67; Test_04_34: aparición de las baterías colocadas 237
Tabla 3.68; Test_04_35: cantidad de vibración de la cámara principal 238
Tabla 3.69; Test_04_36: momento de vibración de la cámara principal 238
Tabla 3.70; Test_04_37: tiempo de vibración de la cámara principal 238
Tabla 3.71; Test_04_38: momento de apertura de la puerta de StarLord 239
Tabla 3.72; Test_04_39: velocidad de apertura de la puerta de la sala de StarLord 239
Tabla 3.73; Test_04_40: momento de cierre de la puerta de la sala de StarLord 239
Tabla 3.74; Test_04_41: velocidad de cierre de la puerta de la sala de StarLord 240
Tabla 3.75; Test_04_42: momento de cambio de las luces de la sala de StarLord 240
Tabla 3.76; Test_04_43: iluminación de la sala de StarLord tras el cambio de luces 240
Tabla 3.77; Test_04_44: activación del portal 244
Tabla 3.78; Test_04_45: movimiento del jugador por el hangar 244
Tabla 3.79; Test_04_46: paso a créditos 245
Tabla 3.80; Test_04_47: rotación de la Tierra 246

Tabla 3.81; Test_04_48: rotación de la Luna....247
Tabla 3.82; Test_04_49: rotación de la Luna sobre la Tierra....247
Tabla 3.83; Test_04_50: dirección de la luz direccional del Sol....251
Tabla 3.84; Test_04_51: rendimiento del Sol....251
Tabla 3.85; Test_04_52: fuerza de los cohetes....253
Tabla 3.86; Test_05_01: flujo de animaciones del personaje....258
Tabla 3.87; Test_05_02: de andar, a correr....261
Tabla 3.88; Test_05_03: de correr, a andar....261
Tabla 3.89; Test_05_04: poder saltar en cualquier situación....261
Tabla 3.90; Test_05_05: distancia de salto....262
Tabla 3.91; Test_05_06: barra de vida del jugador es visible....264
Tabla 3.92; Test_05_07: color de la barra de vida....264
Tabla 3.93; Test_05_08: barra de vida refleja daño recibido....264
Tabla 3.94; Test_06_01: precisión del arma....272
Tabla 3.95; Test_06_02: apuntado del arma....272
Tabla 3.96; Test_06_03: movimiento de la cámara....272
Tabla 3.97; Test_06_04: efecto de bobbing....273
Tabla 3.98; Test_06_05: efecto de bobbing en función del movimiento....273
Tabla 3.99; Test_06_06: rotación de la cámara del menú principal....277
Tabla 3.100; Test_06_07: velocidad de rotación de la cámara del menú principal....277
Tabla 3.101; Test_06_08: posicionamiento de la cámara del menú principal....278
Tabla 3.102; Test_07_01: activación de la linterna del arma....284
Tabla 3.103; Test_07_02: intensida de luz de la linterna del arma....284
Tabla 3.104; Test_07_03: fogonazo del bozal del arma....284
Tabla 3.105; Test_07_04: activación de partículas de colisión con el enemigo....285
Tabla 3.106; Test_07_05: aleatoriedad de las partíclas de colisión con el enemigo....285
Tabla 3.107; Test_07_06: activación de la bala del arma....285
Tabla 3.108; Test_07_07: velocidad de la bala del arma....286
Tabla 3.109; Test_07_08: rango del arma....286
Tabla 3.110; Test_07_09: candencia del arma....286
Tabla 3.111; Test_07_10: caída de la bala....287
Tabla 3.112; Test_07_11: caída controlada de la bala....287
Tabla 3.113; Test_07_12: daño al impactar con el enemigo....287
Tabla 3.114; Test_07_13: salida de la bala del arma....287
Tabla 3.115; Test_07_14: dirección de vuelo de la bala....288
Tabla 3.116; Test_07_15: impacto en cuerpo o cabeza....288
Tabla 3.117; Test_07_16: impacto en sólido....288
Tabla 3.118; Test_07_17: impacto en sólido desaparece....289
Tabla 3.119; Test_07_18: poder disparar....289
Tabla 3.120; Test_07_19: activación de chispas....289
Tabla 3.121; Test_07_20: activación de chispas en función de la vida....290
Tabla 3.122; Test_07_21: activación aleatoria de chispas....290
Tabla 3.123; Test_07_22: retroceso del arma....290
Tabla 3.124; Test_07_23: cantidad de retroceso del arma....291
Tabla 3.125; Test_07_24: momento de retroceso del arma....291
Tabla 3.126; Test_07_25: no poder disparar....291

Tabla 3.127; Test_07_26: recarga del arma 291
Tabla 3.128; Test_07_27: no poder recargar 292
Tabla 3.129; Test_08_01: animaciones adecuadas 295
Tabla 3.130; Test_08_02: animaciones en funcion del lugar del impacto 295
Tabla 3.131; Test_08_03: entidad inmortal 295
Tabla 3.132; Test_08_04: flujo de animaciones adecuado 301
Tabla 3.133; Test_08_05: enemigo va hacia el jugador 301
Tabla 3.134; Test_08_06: enemigo mira al jugador 302
Tabla 3.135; Test_08_07: movimiento sin variar Y 302
Tabla 3.136; Test_08_08: girar solo en eje Y 302
Tabla 3.137; Test_08_09: detección del jugador 303
Tabla 3.138; Test_08_10: daño por golpe 303
Tabla 3.139; Test_08_11: barra de vida mirando siempre a la cámara del jugador 303
Tabla 3.140; Test_08_12: enemigo no desaparece al morir 304
Tabla 3.141; Test_08_13: instanciar explosión 308
Tabla 3.142; Test_08_14: daño de la explosión 309
Tabla 3.143; Test_08_15: materiales adecuados 309
Tabla 3.144; Test_08_16: cambio de materiales 309
Tabla 3.145; Test_08_17: disparo al jugador 310
Tabla 3.146; Test_08_18: instancia de bala del enemigo 310
Tabla 3.147; Test_08_19: bala del enemigo daña al jugador 310
Tabla 3.148; Test_08_20: movimiento oscilatorio en Y 313
Tabla 3.149; Test_08_21: instanciación de enemigos 322
Tabla 3.150; Test_08_22: ubicación de instanciación de enemigos 323
Tabla 3.151; Test_08_23: ataque según distancia con jugador 323
Tabla 3.152; Test_08_24: ubicación de los enemigos instanciados en base a posición del jugador. 330
Tabla 3.153; Test_09_01: aparición del panel de interacción 332
Tabla 3.154; Test_09_02: acción de interactuar 332
Tabla 3.155; Test_09_03: aparición del efecto “outline” 335
Tabla 3.156; Test_10_01: completar el tutorial: guardar información 342
Tabla 3.157; Test_10_02: completar el tutorial: consecuencia 343
Tabla 3.158; Test_10_03: aparición de los paneles de la interfaz del jugador 343
Tabla 3.159; Test_10_04: momento de aparición de los paneles de la interfaz del jugador 343
Tabla 3.160; Test_10_05: aparición de ítems 344
Tabla 3.161; Test_10_06: activación de la funcionalidad de disparar 344
Tabla 3.162; Test_10_07: paso de los diálogos 344
Tabla 3.163; Test_10_08: activación del icono de interacción 345
Tabla 3.164; Test_10_09: desactivación del icono de interacción 345
Tabla 3.165; Test_10_10: movimiento del icono de interacción 345
Tabla 3.166; Test_10_11: desactivación del movimiento del jugador 346
Tabla 3.167; Test_10_12: tareas aparecen en el panel de tareas 346
Tabla 3.168; Test_10_13: momento en el que las tareas aparecen en el panel de tareas 346
Tabla 3.169; Test_10_14: cumplimiento de tareas 346
Tabla 3.170; Test_11_01: elementos interactuables 353
Tabla 3.171; Test_11_02: función de los elementos interactuables 353

Tabla 3.172; Test_12_01: efecto del cambio de dificultad 355
Tabla 3.173; Test_13_01: uso de una poción.....356
Tabla 3.174; Test_13_02: no uso de una poción.....357
Tabla 3.175; Test_13_03: efecto del uso de una poción.....357
Tabla 3.176; Test_13_04: gastar una poción.....357
Tabla 3.177; Test_13_05: duración del efecto de una poción.....358
Tabla 3.178; Test_13_06: lanzamiento de una poción 366
Tabla 3.179; Test_13_07: trayectoria de lanzamiento de una poción 366
Tabla 3.180; Test_13_08: vibración de la trayectoria de lanzamiento de una poción 366
Tabla 3.181; Test_13_09: carga del lanzamiento de una poción.....367
Tabla 3.182; Test_13_10: poción rota 367
Tabla 3.183; Test_13_11: instanciación del charco de ácido.....367
Tabla 3.184; Test_13_12: instanciación del gas tóxico.....367
Tabla 3.185; Test_13_13: aumento progresivo del tamaño del charco de ácido 368
Tabla 3.186; Test_13_14: velocidad de aumento de tamaño del charco de ácido.....368
Tabla 3.187; Test_13_15: daño del charco de ácido 368
Tabla 3.188; Test_13_16: rotación de la tapa de la caja de munición.....370
Tabla 3.189; Test_13_17: velocidad de rotación de la tapa de la caja de munición.....371
Tabla 3.190; Test_13_18: momento de rotación de la tapa de la caja de munición.....371
Tabla 3.191; Test_13_19: no volver a poder interactuar con la caja de munición.....371
Tabla 3.192; Test_13_20: munición aumenta.....372
Tabla 3.193; Test_13_21: cantidad de munición que aumenta.....372
Tabla 3.194; Test_13_22: movimiento del líquido.....380
Tabla 3.195; Test_13_23: visualización de las pociones 383
Tabla 3.196; Test_13_24: visualización del arma.....383
Tabla 3.197; Test_13_25: cambio entre arma y pociones 383
Tabla 3.198; Test_13_26: no visualizar pociones.....384
Tabla 3.199; Test_13_27: solo visualizar poción elegida.....384
Tabla 3.200; Test_14_01: aparición de ítems.....387
Tabla 3.201; Test_14_02: aparición aleatoria de ítems 387
Tabla 3.202; Test_14_03: ubicación de aparición de ítems.....388
Tabla 3.203; Test_14_04: todos los ítems indispensables aparecen.....388
Tabla 3.204; Test_14_05: la colección de ítems depende de la dificultad.....388
Tabla 3.205; Test_14_06: ítems aleatorios.....389
Tabla 3.206; Test_15_01: iluminación adecuada.....393
Tabla 3.207; Test_15_02: intensidad de luz adecuada.....393
Tabla 3.208; Test_16_01: efecto CrawlText 396
Tabla 3.209; Test_16_02: distorsión del fondo 397
Tabla 3.210; Test_16_03: aparición del botón.....397
Tabla 3.211; Test_16_04: utilidad del botón 397
Tabla 3.212; Test_16_04: no poder abrir menús 398
Tabla 7.1; tabla de tecla y su funcionamiento.....412

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 6 (Definiciones y abreviaturas).

1.1 Introducción

Este documento tiene como finalidad principal detallar el desarrollo y características de un videojuego de género **FPS (First Person Shooter)** [1], caracterizado por tener una perspectiva del mundo y de la interacción con el mismo en primera persona.

Este documento se estructura en distintos capítulos que detallan cada fase de desarrollo del proyecto, así como sus características principales, herramientas y algoritmos usados.

Con este proyecto, además de buscar demostrar las diferentes competencias y habilidades adquiridas durante el grado en el ámbito de Sistemas Gráficos [2], desarrollo y diseño de videojuegos, busca también ofrecer una propuesta innovadora dentro del género FPS de videojuegos, con la cual se pueda captar el interés tanto de jugadores aficionados como de aquellos con un enfoque más crítico y analítico.

A lo largo de este documento, se irán explicando las herramientas, algoritmos y métodos usados para conseguir la funcionalidad necesaria para completar el objetivo principal. Además de características y descripciones detalladas del entorno usado.

1.2 Objetivos del trabajo

El objetivo principal de este trabajo no es otro que el de ofrecer al mundo un juego con el que puedan disfrutar de su tiempo libre, así como de analizarlo o utilizarlo como base para otro juego.

Debido al tiempo, esfuerzo y dedicación que supone el desarrollo de un videojuego, este se encuentra aún en fase de desarrollo. No obstante, es lo suficientemente usable para probar y disfrutar de diferentes utilidades, funciones y mecánicas propias de los videojuegos de este género.

1.3 Antecedentes y estado del arte

El juego desarrollado en este proyecto ha surgido como base de la famosa serie de videojuegos “Doom” [3]. Esta franquicia es una serie de videojuegos de disparos en primera persona (FPS), desarrollada por Id Software [4] en 1993 de temática ficción y terror. En “Doom”, los jugadores luchan contra hordas de demonios invasores del infierno. Además, “Doom” fue jugado por aproximadamente 10 millones de personas en los dos años siguientes a su lanzamiento, popularizando este modo de juego y dando origen a una subcultura de jugadores.

Ilustración 1.1; Ilustración de la carátula Doom

Doom” también fue pionero en gráficos 3D inmersivos, juegos multijugador en red y soporte para adiciones y modificaciones personalizadas a través de archivos empaquetados en un archivo de datos conocido como 'WADs' [5]. Como señal de su efecto en la industria, los juegos de disparos en primera persona del auge del género en los años 90, ayudados en gran medida por el lanzamiento del juego, llegaron a ser conocidos simplemente como 'clones de Doom'. Su violencia gráfica, así como la imaginería satánica, hicieron de Doom objeto de controversia.

Ilustración 1.2; Ilustración de una partida de Doom

Además, el juego también se ha ambientado en la famosa saga de StarWars (Guerra de las Galaxias) [6]. Esta saga, la cual fue creada por George Lucas [7], es una de las franquicias cinematográficas más queridas e influyentes de la historia del cine y entretenimiento, expandiéndose a series de televisión, libros, cómics y videojuegos. Esta saga está ambientada en el espacio exterior, donde encontramos planetas, galaxias, naves espaciales, … Los distintos capítulos de StarWars se basan en que hay dos bandos, uno del mal y otro del bien, los cuales están en constante lucha por ver quien prevalece.

En el juego que desarrollamos en este proyecto, el jugador se encontrará en el espacio exterior y, asumiendo el papel de un Trooper conocido como “Doomguy”, como ocurre en Doom, el jugador, tras el fracaso de Darth Vader, se adentrará en una nave espacial para derrotar al enemigo final de la misma interactuando con el entorno y haciendo uso de objetos y armas.

Se ha decidido desarrollar el juego en tres dimensiones para mejorar la calidad gráfica del mismo.

Por todas estas razones, se ha decido que el juego se ha bautizado con el nombre de: “Space Wars”.

1.4 Requisitos iniciales

1. ¿Qué audiencia tendrá este proyecto?
2. ¿Qué motor de videojuegos se ha utilizado?
3. ¿En qué entornos o plataformas se podrá disfrutar del juego?

1.4.1 Audiencia

En la industria de los videojuegos, para estimar la edad de la audiencia que tendrá el juego, se usa un conocido sistema llamado PEGI (Pan European Game Information) [8].

Este sistema clasifica los juegos en varias categorías de edad según el contenido que ofrece el juego y según sea apropiado ese contenido para ese grupo de edad. Las categorías son:

1. **PEGI 3**: esta categoría califica al juego como apta para todas las edades, ya que, normalmente, el contenido del juego, no incluyen imágenes o sonidos que puedan asustar a los niños pequeños
2. **PEGI 7**: esta categoría es para los juegos que tengan un contenido más inapropiado para niños pequeños, pero sin incluir violencia ni contenido para adultos.

3. **PEGI 12**: el contenido de los juegos de esta categoría puede incluir algo de violencia, pero esta violencia está orientada más a personajes no reales y fantásticos, en vez de estar orientada en personajes humanos o animales reconocibles a la vista.
4. **PEGI 16**: esta clasificación agrupa los juegos cuyos contenidos contienen violencia similar al que se esperaría en la vida real
5. **PEGI 18**: los juegos en esta categoría son los juegos con los contenidos más extremamente violentos e incluso pueden incluir escenas de desnudez y contenido sexual.

Ilustración 1.3; Diferentes iconos de PEGI

El icono asociado a la categoría de PEGI del juego, debe estar incluido y visible en la portada y contraportada del juego en físico e incluso, si existe, en su formato virtual.

Este sistema se usa en todo Europa, por lo que en otras zonas del mundo se usan otros sistemas como ESRB (Entertainment Software Rating Board) [9] en Estados Unidos y Canadá o como CERO (Computer Entertainment Rating Organization) [10] en Japón.

En cuanto al juego desarrollado en este proyecto, es adecuado usar el sistema PEGI, puesto que estamos en Europa y se puede clasificar con PEGI 12, puesto que este juego incluye personajes fantásticos y criaturas extrañas, a la vez que violencia leve no muy gráficas hacía los personajes. El objetivo, el cual es destruir al jefe de la nave espacial, tampoco suaviza la violencia del juego. Además, es imprescindible mencionar que es recomendable tener un leve manejo con las diferentes mecánicas de los videojuegos para poder llegar al final del juego.

1.4.2 Herramientas utilizadas

1.4.2.1 Motor de videojuegos

Un motor de videojuegos [11] es un software que aporta una serie de herramientas y funcionalidades, las cuales están diseñadas para facilitar el proceso de desarrollar un videojuego. Algunas de las funcionalidades más importantes de los motores de videojuegos son:

1. El renderizado gráfico el cual maneja la visualización de los gráficos del juego, lo que permite a los desarrolladores crear entornos, efectos y personajes visuales a sus juegos.
2. Integrar audios, música y sonidos
3. Animacion de objetos y personajes
4. Programación de las diferentes lógicas del juego y de los distintos personajes y objetos.

Conociendo esto y tras haberlo valorarlo mucho, se ha optado por usar el motor de videojuegos de Unity [12].

Ilustración 1.4; Logo de Unity

Unity [13] es un motor de videojuegos desarrollado por Unity Technologies en 2005. Este es uno de los más famosos motores de videojuegos del mercado y no es para menos, gracias a:

1. Una gran accesibilidad mediante una interfaz de usuario facil, sencilla e intuitivas con la cual se facilita el desarrollo de los videojuegos. Además,

para programar la lógica de los distintos objetos, personajes y entornos, se usa el lenguaje de programación C SHARP (C#), considerado uno de los lenguajes de programación más intuitivos y fáciles de usar para los nuevos programadores gracias a su pequeña curva de aprendizaje.

2. Versatilidad gracias a soportar el desarrollo de los videojuegos tanto en tres dimensiones como en dos dimensiones.
3. Cuenta con una gran comunidad la cual pone a disposición de los desarrolladores una gran cantidad de tutoriales, documentación y foros. Además de aportar recursos y activos gracias a la “Asset Store”, un espacio donde se pueden encontrar una amplia variedad de assets.

No obstante, Unity también cuenta con muchas desventajas que podrían hacerte dudar a la hora de elegir este motor de videojuegos:

1. A pesar de que Unity es muy potente, es bastante poco eficiente a la hora de hablar de rendimiento en juegos de altos requisitos gráficos en comparación con otros motores de videojuegos como Unreal.
2. Poca disponibilidad de assets gratuitos, debido a que la gran mayoría de assets de la “Asset Store” son de pago.
3. Unity, aunque para empezar es bastante buena opción debido a que es fácil de aprender a usar al principio, tiene una curva de aprendizaje bastante alta cuando profundizamos en la infinidad de opciones que tiene este motor de videojuegos.

1.4.2.2 Herramienta de modelado y animación

En el mundo de los videojuegos, es casi imprescindible tener personajes y modelos en tu escena. Por este motivo, son bastante utilizadas las herramientas de creación de gráficos, animacion, renderizado e iluminación. Para este proyecto, y con este fin, se ha usado Blender [14] [15].

Ilustración 1.5; Logo de Blender

Este es un programa de software libre y código abierto creado en 1995 por Ton Roosendaal [16] como un proyecto interno en NeoGeo, una empresa de animación de origen holandés. Este software aporta varias herramientas y funcionalidades como modelado, iluminación, texturas, renderizado, animación, edición de video y muchas otras cosas.

La elección de esta herramienta ha sido principalmente por su potencia, variedad de herramientas y por ser un software gratuito. Además, Blender con respecto a otro software del mismo tipo, podemos destacar:

1. Es un software libre y gratuito, lo que permite que todo el mundo pueda modificar cualquiera de las herramientas y funcionalidades que nos aporta el programa
2. Su facilidad para exportar e importar proyectos.
3. Diversa cantidad de funcionalidades y herramientas que hace que el programa sea extremadamente completo.

Sin embargo, tantas funcionalidades y herramientas tambien conllevan una serie de desventajas:

1. Blender tiene una gran curva de aprendizaje debido a la gran variedad de herramientas y utilidades.
2. Puede enfrentar algunas limitaciones en cuanto a rendimiento en proyectos grandes

Gracias a esta herramienta, se han podido realizar animaciones como la de explosión de StarLord

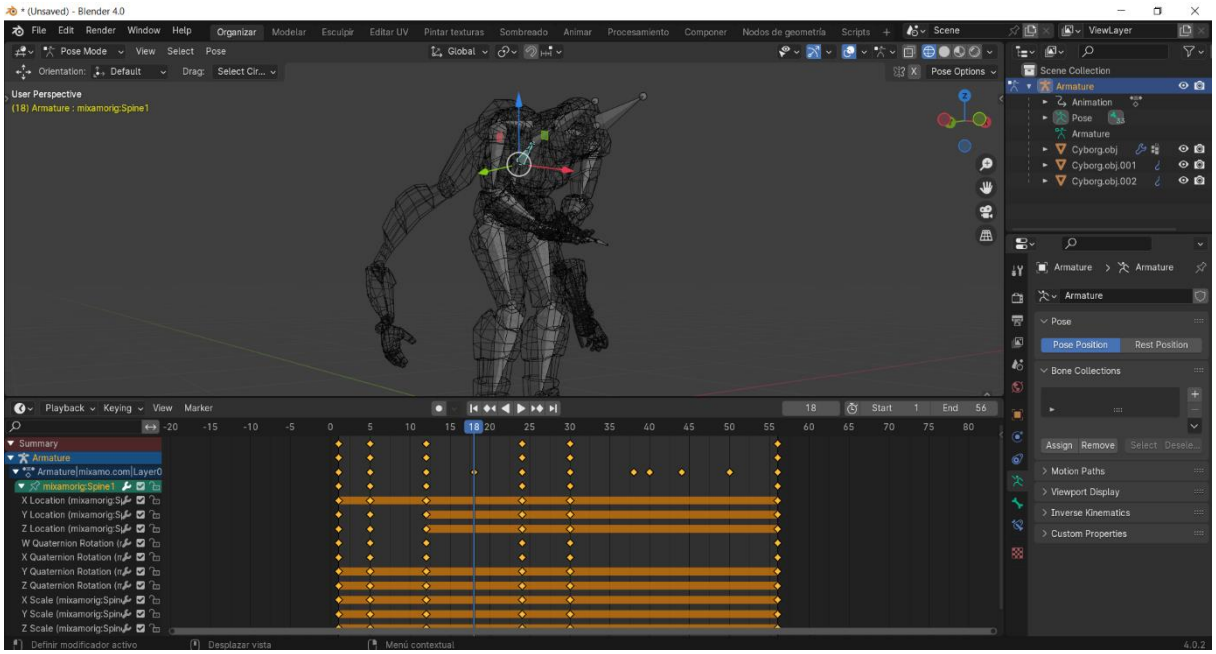


Ilustración 1.6; Animación hecha con la herramienta Blender

1.4.2.3 Herramienta de animación

Además de modelos, estos han de estar animados para realzar el realismo en el videojuego desarrollado. Para conseguir esta animación, se pueden usar herramientas como Blender, ya comentada anteriormente. No obstante, cuando se trata de animaciones humanoides, la mejor herramienta es una página web llamada Mixamo [17], donde se pueden encontrar miles de animaciones de modelos humanoides.

Ilustración 1.7; Logo de Mixamo

Además, de proporcionar animaciones de humanoides, este servicio proporciona personajes 3D y ofrece herramientas como el rigging [18] automático.

No obstante, además de todo lo que ofrece esta herramienta online, el motivo de su elección ha sido por:

1. Su gran accesibilidad
2. Facilidad de uso
3. Calidad de los resultados aportados
4. Intuitiva y clara interfaz

Sin embargo, esta herramienta también porta algunas desventajas:

1. Dependencia de una conexión a internet
2. Personalización de animación es limitada, se cogen las animaciones que hay, pudiéndolas variar un poco, pero sin cambiarlas.
3. Al haber tantas animaciones en la página, su bajo rendimiento en internet.

1.4.2.4 Herramienta de control de versiones

Otra de las herramientas indispensables para desarrollar un proyecto software es el control de versiones, Git [19] [20].

Ilustración 1.8; Logo de Mixamo

Esta herramienta aporta varias herramientas y funcionalidades como control de versiones, gestión de ramas de desarrollo, colaboración en equipo, seguimiento de cambios realizados, …

Los principales motivos de elegir esta herramienta han sido:

1. Su potencia
2. Flexibilidad
3. Software gratuito y de código abierto
4. Avanzada capacidad de gestionar los proyectos

Sin embargo, tambien existen desventajas al usar esta herramienta:

1. Una gran curva de aprendizaje por la gran cantidad de comandos y opciones disponibles que aporta la herramienta

1.4.2.5 Herramienta de diseño

Para diseñar interfaces, se ha utilizado la herramienta online UXPin [21].

Ilustración 1.9; Logo de UXPin

Los motivos de la elección de esta herramienta han sido:

1. Su gran potencia
2. Facilidad de crear interfaces
3. Intuitividad de la herramienta

Sin embargo, esta herramienta también cuenta con una serie de desventajas:

1. Es una herramienta de pago, siendo su versión gratuita muy limitada a la hora de crear prototipos de interfaces
2. Una curva de aprendizaje alta para nuevos diseñadores.

1.4.2.6 Herramienta de modelado y diseño software

Para el modelado y diseño de software, lo que conlleva la creación de diversos diagramas y diseños, se ha usado la herramienta Visual Paradigm [22].

Ilustración 1.10; Logo de VisualParadigm

Esta herramienta aporta funcionalidades como la de modelado y desarrollo software como UML, SysML, BPMN, … para diseñar el software desarrollado.

Los principales motivos de la decisión del uso de esta herramienta son:

1. Modelado de diagramas
2. Generación de código a partir de diagramas
3. Generación de diagramas a partir de código
4. Gran potencia
5. Versatilidad

Sin embargo, tantas funcionalidades y herramientas también conllevan una serie de desventajas:

1. Tiene una gran curva de aprendizaje, debido a la gran cantidad de herramientas y funcionalidades que aporta

1.4.2.7 Herramienta de gestión de proyectos

Para este proyecto, también ha sido necesaria una herramienta para gestionar proyectos. Por ello, se ha usado la herramienta GanttProject [23]

Ilustración 1.11; Logo de GanttProject

Esta herramienta permite al desarrollador por planificar su proyecto, de forma que este puede planificarlo, organizarlo e incluso seguir el progreso de las tareas que ha realizado.

Aparte de estos, existen otros motivos por los que se ha decido usar esta herramienta:

1. Es software gratuito, libre y de código abierto
2. Facilidad de uso
3. Accesibilidad
4. Intuitiva y sencilla interfaz

No obstante, también conlleva una serie de desventajas:

1. Puede llegar a complicarse la gestión de proyectos grandes

1.4.3 Plataformas disponibles

Este proyecto estará disponible en ordenadores con sistema operativos Windows.

1.5 Alcance

En este apartado, se definirá hasta dónde llega el trabajo, delimitando claramente lo que se incluye en el resultado del proyecto.

1. **Proyecto**: el proyecto como tal, realizado con el motor de desarrollo de videojuegos y aplicaciones 3D, Unity con la versión 2022.3.15f1, consta de una carpeta en la que se incluyen todos los archivos relativos al proyecto realizado. Estos archivos contienen información acerca de:
    1. Dos escenas, una principal, en la que se desarrolla el juego generalmente, y otra escena para los créditos del mismo.
    2. Código generado en scripts para las lógicas de las distintas funcionalidades desarrolladas, los cuales están debidamente organizados en carpetas y documentados de forma clara y concisa con comentarios dentro del código.

3. Assets utilizados en el software desarrollado, también organizado de manera clara en distintas carpetas.
4. Una carpeta .git, en la que se guarda información acerca del control de versiones y cambios hechos en el proyecto durante todo su desarrollo.

2. **Ejecutable del software**: este consta de una carpeta llamada "SpaceWars_ejecutable" y generada por Unity, en la que se encuentran los archivos necesarios para ejecutar el proyecto desarrollado. Entre estos archivos, se pueden encontrar:
    1. El archivo ejecutable ".exe", llamado "SpaceWars.exe" y comentado en el **apartado 7.3 Instalación y configuración del sistema**, no debe ser removido de la carpeta, puesto que necesita de los demás archivos que se pueden encontrar en la carpeta para funcionar.
    2. Carpetas con archivos de información para ejecutar el juego.
3. **Documentación del proyecto**: este se refiere a este documento, en el que se encontrará toda la información acerca del software desarrollado.

1.6 Hipótesis y restricciones

El TFT se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

1.7 Estudio de alternativas y viabilidad

En este apartado, estudiaremos las distintas alternativas que se han tenido en cuenta a nivel software y técnico, valorando la viabilidad de estas alternativas y comentando los criterios utilizados para su valoración.

1.7.1 Género del videojuego

Para este proyecto en el que se desarrolla un videojuego FPS, se han valorado muchas otras opciones, ya que, en el mundo de los videojuegos, existen diversos géneros. Entre ellos, se encuentran:

1. **TPS (Third Person Shooter)** [24]: los juegos TPS tienen como característica general que la cámara está situada en la espalda del personaje que controlas, lo que te permite ver tanto el personaje como el entorno. De este tipo de juegos son los títulos: Fortnite, Dark Souls, …
2. **RPS (Real Time Strategy)** [25]: son juegos de estrategia en tiempo real donde los jugadores deben gestionar recursos y tomar decisiones en tiempo real mientras compiten contra otros jugadores. Juegos conocidos de este género son: StarCraft, Age of Empires, …
3. **RPG (Role-Playing Game)** [26]: este tipo de juegos hacen que los jugadores asuman un rol de personajes relativos al contexto del juego. Dentro de este género, nos podremos encontrar juegos de acción o basados en turnos. Ejemplos de juegos de este género son: The Witcher, Skyrim, …
4. **Battle Royale** [27]: los battle royale son un tipo de videojuegos que está en auge actualmente. En estos juegos, los jugadores son lanzados en un mapa formando equipo con otros jugadores o en solitario. En este mapa, los jugadores deben competir contra otros jugadores hasta que solo quede uno o un equipo en función del modo de juego. La clave es que el mapa se reduce gradualmente, por lo que los jugadores van a tender a encontrarse. De los más famosos títulos de este género son: Apex Legends, PUBG: Battlegrounds.
5. **Simuladores** [28]: este tipo de juegos tratan de simular aspectos de la vida real, como vuelos (Microsoft Flights Simulator), camiones (Euro Truck Simulator), relaciones entre personas (The Sims) hasta de carreras (RaceRoom Racing Experience)

1.7.1.1 FPS vs TPS

Esta fue una difícil decisión puesto que TPS es uno de los géneros mejor valorados. No obstante, la decisión de hacer un FPS fue gracias a las siguientes razones:

1. Los juegos FPS ofrecen una perspectiva en primera persona que puede hacer que los jugadores se sientan más inmersos en el juego que con una cámara en tercera persona, ya que se considera que esta vista “a través de los ojos” potencia la experiencia.
2. Es un hecho que los FPS ofrecen mayor control a la hora de apuntar y disparar que los TPS. Este juego, al ser al fin y al cabo un juego de disparos, es idóneo tener más precisión.
3. En TPS, al tener la cámara de forma que se ven todos los elementos del escenario, el jugador puede llegar a distraerse, cosa que no buscamos puesto que es juego de disparos.

1.7.1.2 FPS vs RTS

Los RTS son un género de videojuego ligeramente distinto al de FPS. No obstante, procedía valorar la opción de desarrollar un videojuego RTS. Las razones por las que se ha optado por los FPS fueron:

1. Los juegos FPS, al ofrecer una perspectiva del juego en primera persona, colocan a los jugadores inmediatamente en la acción, lo que puede ser emocionante para ellos. Mientras que, en los RTS, no se vive en primera persona la acción.
2. La curva de aprendizaje de los RTS es bastante más alta que los juegos FPS puesto que necesitas manejar, controlar y gestionar recursos, estrategias, … Esta desventaja hace que el juego sea menos accesible para nuevos jugadores, cosa que no buscamos con este proyecto puesto que queremos que sea accesible para todo el mundo.

1.7.1.3 FPS vs RPG

Los RPG ofrecen una experiencia de juego muy distinta a los FPS, pero hay varias razones por las que se ha preferido desarrollar un FPS en este proyecto en vez de un RPG:

1. Los FPS se caracterizan por tener una acción inmediata, interrumpida y con un ritmo rápido, lo que puede atraer a jugadores que buscan emociones fuertes y constantes. Por el contrario, los RPG incluyen diálogos y exploración del mapa, lo que puede percibirse como una experiencia mucho más lenta que los FPS.
2. Al igual que con los RTS, la curva de aprendizaje de los RPG es bastante más alta que los juegos FPS, por lo que el género FPS también es preferido antes que los RPG por las mismas razones.

1.7.1.4 FPS vs Battle Royale

Los FPS y Battle Royale son similares en cuanto a los elementos de combate y acción propios de ambos juegos, pero también tienen muchas diferencias las cuales son las razones por las que un jugador se decante por un FPS antes que por un Battle Royale:

1. Los FPS consisten en escenarios más estáticos, sin cambios, lo que permite a los jugadores predecir lo que va a pasar y poder anticiparse y prepararse. Mientras que los Battle Royale tienden a variar los mapas, por lo que existe una mayor aleatoriedad.

1.7.1.5 FPS vs Simuladores

Los juegos FPS ofrecen una experiencia totalmente distinta a los simuladores. Aun así, las razones por las que se prefiere a un FPS antes que a un simulador son las siguientes:

1. Como hemos comentado antes, los FPS son más emocionantes e intensos que los simuladores, ya que estos se suelen centrar más en mostrar realismo, lo que se traduce en un ritmo mucho más lento.
2. Los FPS explorar una mayor variedad de escenarios que los simuladores. Los juegos FPS pueden estar ambientados en el espacio exterior o en escenarios fantásticos. Sin embargo, los simuladores deben ambientarse en escenarios reales.
3. Aunque a veces los FPS tienen mayor curva de aprendizaje que los simuladores, no necesitan conocimientos previos, solo deben disparar pulsando un botón y no necesitan antes saber disparar. Mientras que en los simuladores, es necesario algún conocimiento previo de la situación o actividad que simulan. Por ejemplo, en el simulador de tractores, es necesario que sepas conducir un mínimo.

1.7.2 Software

Como hemos comentado en el apartado de herramientas utilizadas, se ha decidido usar el motor de videojuegos de Unity. No obstante, hay otros softwares bastante útiles también. Entre ellos, podemos destacar:

1. Unreal Engine [29]
2. Godot Engine [30]

1.7.2.1 Unity vs Unreal

Unreal es uno de los mejores motores de videojuegos que existen en la actualidad. Este fue creado por Epic Games y ofrece herramientas para el desarrollo de juegos en 2D y 3D, así como una amplia gama de utilidades y funcionalidades para manejar y controlar física avanzada, iluminación, animación, efectos visuales, … Pero, ¿por qué se ha escogido Unity en vez de Unreal?

1. Unreal, al tener mayor cantidad de herramientas avanzadas que Unity y una interfaz mucho más compleja, tiene una curva de aprendizaje mucho más elevada
2. Unreal, a pesar de que tiene una gran comunidad de desarrolladores y usuarios activa al igual que Unity, la comunidad de Unreal es bastante pequeña en comparación con la de Unity.
3. Unity ofrece un desarrollo gratuito en todas sus fases, tanto en características, como en planes de suscripción y licencias. Sin embargo, Unreal, a pesar de que también es gratuito en la mayoría de fases de desarrollo de sus juegos, su creador, Epic Games, cobra algunas regalías sobre los ingresos generados por los juegos desarrollados con esta herramienta. No es uno de los objetivos del proyecto el vender este producto, por lo que no habría problema en este aspecto. Sin embargo, no queremos que el desarrollador que quiera expandir el producto, se vea obligado a pagar ciertos pagos a Epic Games si este quiere vender el producto.

1.7.2.2 Unity vs Godot

Godot es otro de los mejores motores de videojuegos que existen en la actualidad. Este es código abierto y gratuito y ofrece herramientas para el desarrollo de juegos en 2D y 3D. Este motor de videojuegos es conocido por estar centrado en el usuario, por su interfaz intuitiva y por su amplia gama de lenguajes de programación en los que desarrollar nuestros juegos. No obstante, la elección de Unity frente a este software, se debe a estas características:

1. Godot tiene una curva de aprendizaje muy pronunciada en comparación de Unity, lo que puede dificultar a los desarrolladores nuevos en este mundo. Además, aunque su interfaz es intuitiva para los desarrolladores de videojuegos experimentados, puede ser algo compleja y difícil de entender para los nuevos desarrolladores.

2. Godot, al igual que pasaba con Unreal, cuenta con una gran comunidad de desarrolladores y usuarios activa al igual que Unity, pero esta es bastante más pequeña en comparación con la comunidad de Unity.

1.7.3 Hardware

Hablando de las alternativas hardware, aparte del hardware con el que se ha desarrollado el proyecto, se ha estudiado la posibilidad de desarrollar el proyecto con un Macintosh de Apple. No obstante, se ha elegido el equipo usado por los siguientes motivos:

1. Los equipos Macintosh [31], normalmente son menos rentables hablando de la relación rendimiento/precio en comparación con los equipos normales. Es decir, estos equipos traen componentes hardware menos potentes comparándolo con un equipo normal con el mismo precio. Por este motivo, preferimos usar un equipo normal. De otra forma, si hubiéramos comprado un Macintosh con los componentes con potencia y rendimiento necesarios para desarrollar videojuegos, se nos hubiera disparado el costo del presupuesto.
2. Los “Mac”, con su sistema operativo macOS [32], son compatibles con muchas herramientas. No obstante, algunas de esas herramientas pueden estar más optimizadas o tener más características disponibles en otros sistemas operativos, como por ejemplo Windows, sistema operativo usado para el desarrollo de este proyecto.

1.8 Descripción de la solución propuesta

El videojuego propuesto, perteneciente al género FPS, se centra en ofrecer una experiencia inmersiva donde el jugador asume la perspectiva de primera persona. Como características clave de este género de videojuegos incluyen:

1. **Jugabilidad inmersiva** donde los jugadores experimentan acciones y toma de decisiones en tiempo real en el contexto del juego

2. Gran **accesibilidad** para que tanto jugadores nuevos como veteranos puedan disfrutar
3. **Gráficos** y multimedia de calidad para aumentar la sensación de inmersión dentro del juego.

Desde una perspectiva técnica, el juego ha sido desarrollado haciendo uso del famoso motor de videojuegos Unity, el cual nos ha permitido crear un juego agradable visualmente y divertido, así como una experiencia de usuario inmersiva y fluida. La lógica del juego y la programación de los diferentes sistemas interactivos que nos encontramos en el juego se han implementado en C#, lenguaje de programación que aporta potencia y accesibilidad en el desarrollo del código, además de ser un lenguaje bastante sencillo y con poca curva de aprendizaje.

En cuanto a la herramienta para modelar, animar y texturizar, se ha elegido Blender, la cual nos permite presentar modelos 3D con alto grado de detalle y realismo. Gracias a la importación y exportación tan eficaz que Blender ofrece, el trabajo realizado con esta herramienta ha sido fácilmente incluida en Unity.

Nuestro juego es una prueba del potencial de estas herramientas, demostrando que estas son capaces de alcanzar resultados impresionantes en el campo del desarrollo de, no solo videojuegos, sino a todo referido con entornos en tres dimensiones.

1.9 Tecnologías utilizadas

Habiendo comentado las herramientas que hemos usado para desarrollar este proyecto, se pasa a comentar la tecnología que se ha usado

Para empezar, se ha usado la versión 2022.3.15f1 de Unity. Esto se ha decido así puesto que esta versión es una de las versiones LTS (Long Term Support) [33] de Unity. Este tipo de versiones son ediciones de software estables y que ofrecen soporte por un periodo más largo de tiempo, normalmente, durante dos años después de su

lanzamiento. Gracias a este soporte, los equipos de desarrollo pueden trabajar sabiendo que pueden contar con soporte técnico durante más tiempo, una gran ventaja para los proyectos grandes y a largo plazo. Por este motivo se ha decidido utilizar esta versión LTS, porque de esta forma, los desarrolladores pueden utilizar el juego de este proyecto como base para mejorarlo o expandirlo, uno de los objetivos de este proyecto.

Además, como también se ha comentado antes, se ha usado el lenguaje de programación C SHARP (C#) para programar la lógica del juego. Haciendo uso de diferentes librerías para acceder a más funcionalidades y herramientas de trabajo de desarrollo.

Por otro lado, se ha usado la versión 4.0 de Blender puesto que es mucho más potente y aporta más herramientas y funcionalidades que otras versiones anteriores. Además, la interfaz ha mejorado significativamente con respecto a otras versiones, siendo esta más intuitiva y fácil de entender.

También es importante comentar que el equipo informático en el que se ha realizado el proyecto ha sido en un ordenador de marca ASUS, con un procesador Intel i5-11400H de 2.70GHz, 16 GB de memoria RAM y una tarjeta gráfica NVIDIA GeForce RTX 3050 dedicada.

1.10 Metodología de desarrollo de software

Uno de los primeros pasos a la hora de desarrollar un software es el de elegir la metodología de desarrollo que usar. Para este proyecto en el que se desarrolla un software, se ha optado por una **metodología de desarrollo ágil basada en iteraciones.** Entre ellas, hemos seleccionado la metodología **Scrum** [34] [35], la cual ha sido fundamental para gestionar de manera efectiva las diversas complejidades del proyecto desarrollado.

Esta metodología [36] organiza el proyecto en iteraciones cortas de entre una y cuatro semanas de duración aproximadamente cada una, conocidas como **sprints**. Como características principales del desarrollo de esta metodología:

1. Al comienzo de cada sprint, se planifica las distintas tareas a completar a lo largo del sprint. Estas tareas son elegidas según prioridad o importancia y esto nos permite adaptar el desarrollo para trabajar en lo más urgente o importante en ese momento.
2. Durante cada sprint, se realizan tareas de desarrollo y pruebas de ese desarrollo acto seguido, lo que facilita la implementación de cambios y corrección de errores en tiempo real.
3. Al final de cada sprint, se presenta a los clientes o stakeholders (en este caso, distintos compañeros, familiares y amigos) lo que se llama un incremento, la parte producida al final del sprint y que tiene como condición indispensable ser un prototipo de nuestro proyecto totalmente funcional y operativo y en condiciones de ser entregado o mostrado a terceros.

Para este proyecto, los aspectos clave de esta metodología y la forma de desarrollar el proyecto usando esta metodología son:

1. La organización y planificación de este proyecto en ciclos de trabajo de entre una y cuatro semanas aproximadamente cada uno, con los cuales al final puedo tener una versión del juego con nuevas funcionalidades o mejoras, me permite revisar y adaptar el plan de desarrollo regularmente, asegurando que el producto final cumpla las expectativas y se pueda adaptar a los requisitos cambiantes que puedan aparecer durante el desarrollo del proyecto.
2. Al final de cada sprint, reviso todo el trabajo realizado mediante pruebas de funcionalidades y jugabilidad para evaluar el progreso y determinar si es necesario incluir alguna tarea de repaso o arreglo de problemas o inconsistencias en el siguiente sprint, lo que me permite identificar y arreglar problemas desde etapas tempranas del proyecto.
3. Al conseguir un incremento, busco que los clientes o stakeholders (en este caso amigos, familiares o compañeros) prueben el producto para conseguir

feedback de ellos y así valorar mejoras o arreglos en los siguientes ciclos de trabajo.

No obstante, a parte de esta metodología, existen otras muchas totalmente válidas que también se han tenido en cuenta al elegir la metodología, pero no se han optado por ellas. Estas son:

1. **Desarrollo Lean** [37]: esta metodología se centra en maximizar el producto enfocado al cliente, a la vez que se minimiza el desperdicio de usar cualquier recurso o esfuerzo que no contribuya al desarrollo del proyecto ni al producto final.
2. **Desarrollo en cascada** [38]: este es un enfoque que se realiza de forma secuencial y se divide en varias fases de desarrollo que se deben completar antes de pasar a la siguiente. Estas fases son: análisis de requisitos, diseño, codificación e implementación, pruebas y verificación y mantenimiento del producto.
3. **Desarrollo en espiral** [39]: combina elementos del desarrollo en cascada y del prototipado iterativo, metodología la cual se centra en crear prototipos en las fases tempranas del desarrollo del proyecto para validar conceptos, funcionalidades e interfaces antes de desarrollar cualquier producto. El aspecto clave de la metodología de desarrollo en espiral es que, en cada espiral, se planifica, se determinan riesgos, se desarrolla un prototipo y por último, antes de comenzar con la siguiente iteración, se evalúa el resultado obtenido en esa espiral. Esta metodología está enfocada en identificar y mitigar riesgos.

1.10.1 Scrum VS Desarrollo Lean

El desarrollo Lean es tan buena metodología de desarrollo como el basado en iteraciones, pero las razones por las que se ha optado por el desarrollo ágil basado en iteraciones son:

1. La metodología ágil Scrum permite adaptar el desarrollo del proyecto en cada ciclo de trabajo, lo que se considera esencial y útil en el desarrollo de un videojuego. El desarrollo Lean, por el contrario, se centra más en eliminar el desperdicio de recursos y puede ser más difícil incorporar cambios y nuevas características al proyecto.
2. El desarrollo Lean se centra más en la eficiencia que en la calidad de la experiencia del usuario. Mientras que la metodología ágil permite evaluar y apreciar el progreso del proyecto mediante las entregas cíclicas, lo que hace que el producto final cumpla con los requisitos y expectativas del proyecto, cosa imprescindible para cualquier proyecto.
3. La metodología de desarrollo Lean puede eliminar algunas características que podrían parecer no esenciales desde la perspectiva de la eficiencia pero que realmente son fundamentales para la experiencia del usuario en el producto final del proyecto.

1.10.2 Scrum VS Desarrollo en cascada

Elegir entre desarrollo ágil basado en iteraciones y desarrollo en cascada fue una decisión difícil puesto que ambos son muy buenas metodologías. Finalmente, las razones por las que se decidió usar Scrum son:

1. La metodología ágil permite adaptaciones rápidas a los cambios e imprevistos surgidos en el desarrollo del proyecto, cosa que se considera una gran ventaja a la hora de desarrollar un videojuego puesto que las ideas y funcionalidades pueden evolucionar significativamente durante el desarrollo del juego. Mientras que el desarrollo en cascada es bastante rígido en cuanto a cambios ya que una vez que una etapa se completa, volver a ella posiblemente sea complicado.
2. En la metodología ágil, al trabajar en ciclos de trabajo cortos y realizar pruebas del producto continuamente, los riesgos se encuentran y se solucionan en las fases tempranas del proceso de desarrollo. No obstante, en la metodología de desarrollo en cascada, los problemas no pueden

descubrirse hasta la fase de prueba, una fase muy tardía en el desarrollo del proyecto, y encontrar fallos en esa fase resultaría en retrasos importantes en el desarrollo del proyecto.

1.10.3 Scrum VS Desarrollo en espiral

El desarrollo en espiral resulta ser una muy buena metodología para trabajar en el desarrollo de software puesto que es parecida a la de desarrollo en cascada, pero con algunos cambios. No obstante, se ha decidido usar el desarrollo ágil basado en iteraciones por estos motivos:

1. Los juegos FPS están en constante cambio de requisitos durante su desarrollo, por lo que es crucial que el proyecto se pueda adaptar a estos cambios de forma rápida y eficiente. No obstante, la metodología en espiral, al estar enfocada en la identificación y mitigación de riesgos, no está tan preparada para afrontar tantos cambios de requisitos durante su proyecto como la metodología ágil basada en iteraciones.
2. Otra razón para no usar metodología en espiral es que el desarrollo de los proyectos con esta metodología es más duradero debido a que las fases de identificación y mitigación de riesgos ocupan mucho tiempo y recursos. Los juegos FPS no pueden permitirse tener un gran delay en su fecha de lanzamiento acordada, por lo que esta metodología no es la más adecuada para el desarrollo de este tipo de proyectos.

1.11 Análisis de riesgos

Al desarrollar un videojuego FPS, podemos encontrarnos algunos problemas. Alguno de los riesgos potenciales que podrían incluir en el desarrollo de un videojuego FPS son:

1.11.1 Riesgos técnicos

Estos riesgos están relacionados con la tecnología usada en el proyecto e involucran riesgos a la hora de realizar la implementación y funcionalidades de las distintas herramientas usadas. Los riesgos más comunes son:

1. **Incompatibilidad entre distintos softwares**

   1.1. **Descripción del riesgo:** este riesgo se refiere a las dificultades encontradas a la hora de usar e intercambiar información entre distintos softwares por incompatibilidades entre ellos como motivo principal.

   1.2. **Probabilidad de riesgo:** Media

   1.3. **Impacto en el proyecto:** Alto

   1.4. **Plan de contingencia:** como plan de contingencia para este posible problema, antes de iniciar el proyecto debemos realizar pruebas de compatibilidad entre los distintos softwares que vayamos a usar. Además, se debe mantener actualizado el software o, al menos, tener una versión estable del software. Como recomendación, se deben tener preparados como alternativa algunos otros softwares.

2. **Fallos de terceros (librerías o plugins)**

   1.1. **Descripción del riesgo:** con este problema nos referimos a librerías y plugins que podrían fallar al usarse por no estar actualizados o por no ser compatibles con la versión del software que se esté usando para desarrollar el proyecto.

   1.2. **Probabilidad de riesgo:** Alta

   1.3. **Impacto en el proyecto:** Medio

   1.4. **Plan de contingencia:** para no tener problemas si surge este este problema, antes de empezar el proyecto se debe revisar que las librerías y plugins usadas estén en la última versión, así como de testear y evaluar que no hay fallos de uso al usarlos con otro

software. Como con el riesgo anterior, sería bueno tener alguna alternativa por si en algún momento falla.

1.11.2 Riesgos de gestión

Estos riesgos se refieren a los aspectos organizativos, de gestión y administrativos de un proyecto. Es decir, estos están relacionados con el control, planificación y gestión de los recursos del proyecto (recursos humanos, materiales, económicos, …). Los riesgos de este tipo más comunes son:

1. **Problemas con el trabajo en equipo y con la coordinación**

   1.1. **Descripción del riesgo:** este riesgo se refiere a las diferentes situaciones en las que se encuentran dificultades en la colaboración de forma efectiva entre los miembros del equipo los cuales usan diferentes herramientas de trabajo como Blender, Unity, Git, …

   1.2. **Probabilidad de riesgo:** Media

   1.3. **Impacto en el proyecto:** Medio

   1.4. **Plan de contingencia:** la solución para este problema es establecer unos protocolos claros de comunicación y gestión de versiones. Para ello, sería adecuado realizar regularmente reuniones de equipo con el fin de organizar y coordinar el equipo. También es fundamental que se utilicen herramientas compatibles entre sí, para facilitar sincronización y seguimiento del proceso realizado por parte de los diferentes miembros del equipo de trabajo.

2. **Desviaciones e imprevistos respecto al cronograma establecido**

   1.1. **Descripción del riesgo:** cuando se habla de desviaciones o imprevistos con respecto a un cronograma el cual hay que seguir, se está haciendo referencia a posibles retrasos en las entregas

establecidas debido a imprevistos tales como problemas técnicos o indisposición de algún trabajador, o también a mala estimación u organización de la planificación.

1.2. **Probabilidad de riesgo:** Alta

1.3. **Impacto en el proyecto:** Alto

1.4. **Plan de contingencia:** ante este problema, lo idóneo es implementar una gestión de proyectos ágil que esté presente durante todo el proyecto y que permita ajustar la planificación establecida ante cualquier desajuste. Algunas de las metodologías ágiles más importantes son:

1.4.1. **Metodología Scrum**: esta metodología es especialmente útil para el desarrollo software ya que se centra en una entrega de forma incremental e iterativa de productos donde se prioriza la colaboración y comunicación de todo el equipo de trabajo.

1.4.2. **Extreme Programming** [40]**:** esta metodología, al igual que la metodología Scrum, está enfocada en el desarrollo de software. No obstante, este tipo de gestión de proyectos intenta mejorar un software que ya esté creado, no crear un software de cero. Su funcionamiento es similar al de la metodología Scrum.

1.12 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFT, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

Como se ha comentado en el **apartado 1.12 Metodología de desarrollo de software**, se va a usar una metodología de desarrollo ágil para el desarrollo del

software de este proyecto. Por lo tanto, para llevar a cabo la estimación del tamaño y esfuerzo del proyecto, se usarán “Story points” (puntos de historia), los cuales reflejan el esfuerzo de realizar una determinada tarea.

A la hora de estimar el esfuerzo de una tarea en la metodología Scrum con Story points, se utiliza la **secuencia de Fibonacci** [41] debido a que:

1. A medida que aumenta el número en la secuencia de Fibonacci, la diferencia entre los números sucesivos se vuelve cada vez más grande, lo que refleja la naturaleza incierta y variable del esfuerzo de las tareas. Esta incertidumbre es necesaria puesto que es muy difícil acertar a la hora de estimar el esfuerzo de las tareas. Además, hay que tener en cuenta que las tareas más grandes tienden a tener una mayor incertidumbre y variabilidad en la estimación, mientras que las tareas más pequeñas son más precisas a la hora de estimar el esfuerzo necesario para realizarlas.
2. Al utilizar una escala exponencial, se reduce la posibilidad de **sobreestimar** las tareas, ya que estas se agrupan en categorías más amplias, lo que facilita la estimación y reduce la necesidad de una precisión exacta.

Por estos motivos, se utilizará la secuencia de Fibonacci para estimar las tareas con Story points. No obstante, se ha de tener en cuenta que existe una restricción temporal en este TFT, por lo que, para tener en cuenta el tiempo de desarrollo a la hora de la planificación, vamos a establecer una relación Story point (esfuerzo) / horas necesarias para realizar tarea, determinadas por la siguiente tabla:

Story point	Horas de trabajo (aproximadas)	Descripción
1	3 horas	Simple, requiere poco trabajo
2	6 horas	Un poco complejo
3	9 horas	Requiere entendimiento y algo de trabajo
5	15 horas	Bastante complejo

8	24 horas	Muy complejo
13	36 horas	Muy complejo y sobre distintas áreas y/o parcialmente incierto
21	Más de 36 horas	Totalmente incierto y/o extremadamente complejo

Tabla 1.1; Relación Story points/horas de trabajo

Antes de hacer la estimación de tareas, se debe tener en cuenta que es imprescindible seguir siempre la secuencia de Fibonacci a la hora de estimar las tareas con Story points. Por lo que, si por la suma de SP (Story points) al subdividir tareas, no da un número de la secuencia de Fibonacci, se deberá redondear al siguiente número de la secuencia de Fibonacci. Por ejemplo: si una tarea se estima 10 SP = 3 SP + 3 SP + 4SP, se debe estimar que la tarea costará 13 SP, que es el siguiente número al 10 dentro de la secuencia de Fibonacci.

Teniendo en cuenta esta condición y sabiendo la relación entre SP y horas, se pasa a ver la estimación de las tareas en la siguiente tabla:

Tarea	Story points (SP)	Horas de trabajo (aproximadas)
Desarrollo del mapa o entorno de juego (nivelado): 1. Cúpula exterior 2. Primera planta 3. Ascensor 4. Segunda planta 5. Tercera planta	11 SP -> 13 SP: 1. 1 SP 2. 3 SP 3. 2 SP 4. 3 SP 5. 2 SP	33 -> 36 horas: 1. 3 horas 2. 9 horas 3. 6 horas 4. 9 horas 5. 6 horas
Implementación del personaje principal	1 SP	3 horas
Implementación del movimiento del personaje	2 SP	6 horas
Implementación del movimiento de la cámara principal	2 SP	6 horas
Implementación de arma del personaje	1 SP	3 horas

Implementación de funcionalidad del arma del personaje (disparar)	3 SP	9 horas
Mejora del arma y la munición	3 SP	9 horas
Diseño de las interfaces: 1. Menú principal 2. Menú de pausa 3. Menú de muerte 4. PlayerUI	8 SP: 1. 2 SP 2. 2 SP 3. 2 SP 4. 2 SP	24 horas: 1. 6 horas 2. 6 horas 3. 6 horas 4. 6 horas
Implementación de enemigos**:** 1. Enemigo con ataque cuerpo a cuerpo 2. Enemigo con ataque a distancia 3. Jefe final	10 SP -> 13 SP: 1. 3 SP 2. 3 SP 3. 4 SP	33 -> 36 horas: 1. 9 horas 2. 9 horas 3. 15 horas
Interacción con el entorno	5 SP	15 horas
Implementación de NPCs (no enemigos): 1. Kyle 2. Freddy	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Menús y sus funcionalidades	3 SP	9 horas
Implementación de objetos: 1. Modelado, texturas y materiales 2. Funcionalidad	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Interacción con objetos	5 SP	15 horas
Animación de todos los modelos de la escena	5 SP	15 horas
Decoración del mundo	3 SP	9 horas
Implementación de audio y efectos sonoros del juego	3 SP	9 horas
TOTAL:	86 Story points	252 horas

Tabla 1.2; Estimación de tareas

Cabe comentar que estas tareas son de desarrollo, por lo que no se han tenido en cuenta las horas de documentación, búsqueda de información, diseño, …

Asumiendo que la etapa de desarrollo del proyecto realizado será realizada entre los meses de diciembre y mayo, teniendo en cuenta que el mes de enero es periodo de exámenes y que, por tanto, no se trabaja en la etapa de desarrollo del proyecto, contamos con 5 meses de desarrollo.

Como se ha comentado previamente, los sprints son de una duración de entre una y cuatro semanas. Además, hay que tener en cuenta que los sprints son realizados por un equipo de trabajo de entre 3 y 9 personas según Deloitte, una página web de artículos [42]. Aunque, para este trabajo, se estimará una media de 5 personas en un equipo Scrum. Teniendo en cuenta a una media de 5 personas en un equipo de desarrollo, se puede estimar que, para un sprint de 4 semanas, cada persona trabaja una semana y un poco más. Sabiendo todo esto, se estimar que, para una persona en el equipo Scrum de desarrollo de este proyecto, el autor del TFG, en vez de 5 personas de media, cada sprint durará aproximadamente un mes. Al contar con 5 meses de desarrollo, se realizarán 5 sprints. En cada sprint, al tener 86 story points totales, se realizarán 86/5 = 17.2 -> **18 SP.**

1.13 Planificación temporal

El desarrollo del proyecto, como se ha comentado en el apartado anterior, se realizó durante los meses de diciembre, febrero, marzo, abril y mayo, realizando una iteración en cada mes.

No obstante, a parte del desarrollo del proyecto, también hay otras fases que se han realizado. La primera de ellas, y con la que vino el comienzo real del proyecto, es la fase de análisis y diseño del proyecto, en el que se especificaron y se definieron los distintos requisitos mínimos del proyecto junto con el cliente. Otra de las fases indispensable de un proyecto, a parte de la del desarrollo, es la fase de pruebas, en la que se realizan pruebas de las diferentes mecánicas y técnicas del videojuego desarrollado con el fin de realizar pequeños ajustes para mejorar algunos aspectos del juego. Por último, otra de las fases indispensable de cualquier proyecto es la

documentación del mismo, en el que se detalla y se explica cada uno de los aspectos del proyecto. Esta última fase se realizará en paralelo con todas las demás fases del proyecto, pudiendo realizar de forma síncrona la documentación del proyecto y las otras fases.

Conociendo todas las fases que han comprendido la realización de este proyecto, se procede a determinar el tiempo que ocupará cada fase, con el objetivo de organizar el tiempo de realización del proyecto, teniendo en cuenta que el proyecto finalizará a mediados del mes de junio y que el mes de enero no se trabaja.

1. Análisis y diseño: 1/11/2023 – 30/11/2023
2. Desarrollo: 1/12/2023 – 31/05/2024
3. Pruebas: 01/05/2024 – 31/05/2024
4. Documentación: 01/11/2023 – 08/06/2024

A continuación, se aporta un diagrama de Gantt [43] en el que se detalla la planificación temporal de la realización del proyecto.

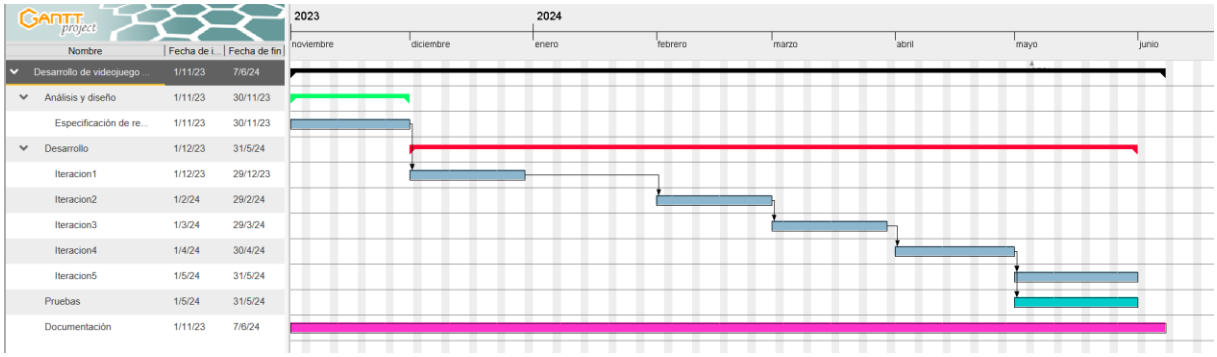


Ilustración 1.12; Diagrama de Gantt

1.14 Presupuesto

Toda actividad tiene un precio y este proyecto no iba a ser menos. Para estimar una cantidad económica presupuestaria, debemos tener en cuenta diversos factores

1.14.1 Activos fijos

Como recursos activos, vamos a valorar el equipo usado para desarrollar este proyecto. Vamos a usar la página web del comercio electrónico de PCComponentes [44] para asemejar precio a los diferentes recursos activos que se han usado en el desarrollo de este proyecto. Esta es una página web centrada en el comercio de tecnología especializada en productos informáticos, electrodomésticos y electrónicos.

Teniendo en cuenta que un equipo informático portátil, según la página web TECFYS [45], dura entre 2 y 4 años (una media de 3 años), y que le proyecto dura aproximadamente 4 meses (duración de un cuatrimestre), se debe estimar el coste proporcional al uso en el proyecto. Para ello, se usará esta fórmula:

$$\text{Coste proporcional} = \text{Coste total del componente} * \frac{\text{Duracion del proyecto en meses}}{\text{Vida útil total en meses}}$$

Siendo:

1. Duración del proyecto en meses: 4 meses
2. Vida útil total: 3 años = 36 meses

Recurso	Coste total	Coste proporcional
Procesador Intel Core i5-11400H	135.32 €	15.04 €
Memoria RAM 16 GB	47.99 €	5.33 €
SSD 512 GB M.2	39.99 €	4.44 €
NVIDIA GeForce RTX 3050	299.99 €	33.33 €
TOTAL	523.29 €	58.14 €

Tabla 1.3; Precios de activos

Gracias a esta tabla de precios, se puede ver que uno de los recursos activos, el equipo informático, costaría 58.14 euros en proporción al uso que se le estaría dando.

1.14.2 Recursos humanos

Para realizar el proyecto, se han necesitado los servicios de:

1. Un diseñador de videojuegos
2. Un programador de videojuegos en C#
3. Un modelador para conseguir los modelos del videojuego
4. Una persona encargada de conseguir sonidos acordes a acciones del videojuego, un ingeniero de sonido.

Teniendo en cuenta estos recursos humanos, se va a ver el salario necesario para contratar sus servicios. Para ello, se va a usar la página web llamada Talent [46], la cual es una página para buscar salarios, empleos y empresas en España. Cabe destacar que esta página está actualizada, por lo que los datos son considerados actuales.

Recurso	Coste promedio (mensual)
Diseñador de videojuegos	1500 €
Programador C#	2375 €
Modelador	2000 €
Ingeniero de sonido	2208 €
TOTAL	6283 €

Tabla 1.4; Precios de recursos humanos

Gracias a esta tabla de los salarios que los recursos humanos necesarios cobrarían, se puede ver que este recurso nos costaría 6283 euros mensuales.

1.14.3 Gastos indirectos

Todo este proyecto se ha desarrollado en Jaén, por lo que se debe asumir un coste extra en otras cosas como en el alquiler de un piso en la ciudad donde se desarrolla el proyecto para que el desarrollador pueda ir a trabajar. Para no complicar los cálculos y así evitar que salgan erróneos o inciertos, se va a imputar un 10% del coste total del proyecto.

1.14.4 Licencias

En cuanto a licencias, para desarrollar este proyecto se ha usado el sistema operativo Windows 10 Pro, de Microsoft. Por lo que, según PCComponentes, esta cuesta 119.99 euros.

1.14.5 Conclusión

Recurso	Coste
Recursos activos fijos	58.14 €
Recursos humanos	6283 €
Licencias	119.99 €
Gastos indirectos	10% del total
TOTAL	6461.13 € - 10% = 5812.02 €

Tabla 1.5; Presupuesto total

Habiendo analizado los diferentes aspectos a tener en cuenta para orientar el presupuesto, se puede afirmar que todo este proyecto necesitaría contar con un presupuesto de **5812.02 euros**.

2 DISEÑO INICIAL

En este capítulo se buscará definir el funcionamiento de la aplicación, así como sus requisitos iniciales. Con esta fase del desarrollo se busca conseguir un diseño preliminar del proyecto, el cual, por la metodología de desarrollo elegida, irá ajustándose en cada fase de incremento del desarrollo del proyecto.

Debido al uso de una metodología de desarrollo ágil, antes de proceder a las especificaciones del sistema, se hará una descripción del diseño general previo a la primera iteración.

El juego desarrollado en este proyecto se trata de un juego de género FPS en solitario, como hemos comentado previamente. Como tal, este juego debe cumplir estos requisitos generales básicos:

1. Una interfaz sencilla e intuitiva con
    1. Un HUD (Heads-Up Display), una interfaz que muestra información esencial para la experiencia sin obstruir la visibilidad del jugador. Esta incluye:
        - Barra de salud del jugador
        - Contador de munición del arma
        - Objetivos de la misión
        - Contador de ítems u objetos
    2. Menús de juego:
        - Menú de inicio del juego
        - Pantalla de fin del juego
        - Menú de pausa
2. Deben existir NPCs (Non-playable characters) tales como enemigos y aliados los cuales encontraremos durante nuestra experiencia.
3. Armas y objetos.

4. Entornos interactivos
5. Elementos del juego como misiones que el jugador debe superar para avanzar.
6. Una trama o “Lore” de acuerdo a la ambientación del juego.

2.1 Especificaciones del sistema

En este subcapítulo se presentarán la especificación detallada de los requisitos acordados entre el cliente (tutor del proyecto) y el desarrollador (autor del proyecto) al inicio del proyecto. Todos los requisitos serán clasificados en subapartados para aclarar. Además, para ayudar a definir los diferentes requisitos determinados, se usarán diferentes versiones de la siguiente tabla, la tabla 2.1:

Identificador del requisito*	
Título	Título descriptivo del requisito
Descripción	Información del requisito
Prioridad	Importancia del requisito*

Tabla 2.1; Plantilla de la tabla de requisitos del sistema

* El identificador del requisito será un código con el que se podrá hacer referencia a este elemento en un futuro. Este estará formado por las iniciales del tipo de requisito (funcional, f, o no funcional, n) y por el numero de la sucesión de los mismos.

* La importancia o prioridad del requisito estará clasificada como:

1. **Extra**: los requisitos con esta prioridad serán requisitos no esenciales para el trabajo pero que aportan o complementan el proyecto. Sin estos requisitos, el proyecto seguiría completo.
2. **Baja**: estos requisitos son aquellos no muy urgentes pero que deben estar en el proyecto.

3. **Media**: los requisitos de prioridad media son requisitos importantes para el funcionamiento del proyecto, por lo que deben ser atendidos con prontitud, pero no son urgentes.
4. **Alta**: estos requisitos son los más urgentes y esenciales para el proyecto, por lo que deben ser abordados con máxima prioridad.

2.1.1 Requisitos funcionales

Los requisitos funcionales [47] son aquellos que definen las funciones y tareas que el software o sistema debe realizar ante unas entradas o bajo ciertas condiciones. El proyecto desarrollado contará con los siguientes requisitos funcionales

2.1.1.1 Sistema de menús e interfaz

RF01	
Título	Iniciar juego
Descripción	El jugador podrá iniciar el juego
Prioridad	Alta

Tabla 2.2; RF01: Iniciar juego

RF02	
Título	Pausar juego
Descripción	El jugador podrá detener el juego a mitad de su partida sin que el tiempo siga transcurriendo
Prioridad	Alta

Tabla 2.3; RF02: Pausar juego

RF03	
Título	Reiniciar juego
Descripción	El jugador podrá reiniciar el juego, empezando desde el principio
Prioridad	Media

Tabla 2.4; RF03: Reiniciar juego

RF04	
Título	Salir del juego
Descripción	El jugador podrá cerrar la aplicación del juego y salir directamente
Prioridad	Baja

Tabla 2.5; RF04: Salir del juego

RF05	
Título	Visualizar información de la partida
Descripción	El jugador podrá ver en todo momento información relacionada con la partida como su vida o la de los enemigos, armas, munición, tareas o misiones, …
Prioridad	Alta

Tabla 2.6; RF05: Visualizar información de la partida

2.1.1.2 Gestión del jugador

RF06	
Título	Control del movimiento del jugador
Descripción	El jugador podrá mover al personaje principal en cualquier dirección usando los dispositivos de entrada estándar.
Prioridad	Alta

Tabla 2.7; RF06: Control del movimiento del jugador

RF07	
Título	Complementos del movimiento del jugador
Descripción	El jugador podrá realizar movimientos complementarios como correr y saltar
Prioridad	Baja

Tabla 2.8; RF07: Complementos del movimiento del jugador

RF08	
Título	Interacción
Descripción	El jugador podrá interactuar con el entorno del juego
Prioridad	Alta

Tabla 2.9; RF08: Interacción

RF09	
Título	Sistema de salud
Descripción	Tanto el jugador como algunos de los distintos NPCs del juego, dispondrán de un sistema de salud que disminuya con el daño recibido y que, en algunas situaciones, pueda regenerarse.
Prioridad	Alta

Tabla 2.10; RF09: Sistema de salud

2.1.1.3 Sistema de combate

RF10	
Título	Manejo de armas
Descripción	El jugador podrá equipar, recargar y usar armas
Prioridad	Alta

Tabla 2.11; RF10: Manejo de armas

RF11	
Título	Sistema de apuntado
Descripción	El jugador, al usar armas, podrá apuntar con ellas para mejorar la precisión.
Prioridad	Baja

Tabla 2.12; RF11: Sistema de apuntado

RF12	
Título	Disparar
Descripción	El jugador podrá disparar proyectiles usando sus armas para dañar a los enemigos del juego.
Prioridad	Alta

Tabla 2.13; RF12: Disparar

2.1.1.4 NPCs

RF13	
Título	Comportamiento de los enemigos
Descripción	Los enemigos detectarán al jugador si este entra dentro de un rango de detección determinado, indicándolo mediante algún sonido y cambio en su apariencia física.
Prioridad	Alta

Tabla 2.14; RF13: Comportamiento de los enemigos

RF14	
Título	Ataque de los enemigos
Descripción	Los enemigos perseguirán al jugador al detectarlo y le atacarán mediante armas a distancia o en combate cuerpo a cuerpo
Prioridad	Alta

Tabla 2.15; RF14: Ataque de los enemigos

RF15	
Título	Variedad de enemigos
Descripción	Existirán diferentes tipos de enemigos, con diferentes armas y formas de atacar al jugador y con diferentes características
Prioridad	Media

Tabla 2.16; RF15: Variedad de enemigos

RF16	
Título	Interactuar con NPCs aliados
Descripción	El jugador, podrá interactuar con algunos NPCs aliados, los cuales ayudarán al jugador a completar el juego.
Prioridad	Baja

Tabla 2.17; RF16: Interactuar con NPCs aliados

2.1.1.5 Sonidos y música

RF17	
Título	Efectos de sonido
Descripción	Se reproducirán sonidos para representar los distintos eventos ocurridos en el juego.
Prioridad	Alta

Tabla 2.18; RF17: Efectos de sonido

RF17	
Título	Modificación del volumen de sonidos y música
Descripción	El jugador podrá modificar el volumen de los sonidos y música de la partida en cualquier momento
Prioridad	Media

Tabla 2.19; RF18: Efectos de sonido

2.1.2 Requisitos no funcionales

Los requisitos no funcionales [48] son aquellos que abordan aspectos como el rendimiento, seguridad, usabilidad, fiabilidad, … con objetivo de que el proyecto cumpla con una experiencia óptima y satisfactoria para el usuario.

2.1.2.1 Rendimiento

RNF01	
Título	Frames Per Seconds (FPS)
Descripción	El juego debe operar con un minimo de 60 fps (fotogramas por segundo) en todo momento.
Prioridad	Alta

Tabla 2.20; RNF01: FPS

2.1.2.2 Usabilidad

RNF02	
Título	Intuitividad
Descripción	El juego debe ser intuitivo, así como sus controles y su interfaz.
Prioridad	Media

Tabla 2.21; RNF02: Intuitividad

RNF03	
Título	Documentación
Descripción	Proporcionar una documentación clara y completa, la cual incluya instrucciones y manuales para todo lo manejable por el usuario.
Prioridad	Alta

Tabla 2.22; RNF03: Documentación

2.1.2.3 Estabilidad

RNF04	
Título	Tolerancia a fallos
Descripción	Los fallos ocurridos durante el juego deben ser bien manejados por el sistema
Prioridad	Alta

Tabla 2.23; RNF04: Tolerancia a fallos

RNF05	
Título	Pruebas del sistema
Descripción	El sistema debe haber sido sometido a pruebas para asegurar su estabilidad
Prioridad	Media

Tabla 2.24; RNF05: Pruebas del sistema

2.1.2.4 Compatibilidad

RNF06	
Título	Compatibilidad con otras plataformas
Descripción	El juego debe ser totalmente compatible con las distintas plataformas a las que va dirigido.
Prioridad	Alta

Tabla 2.25; RNF06: Compatibilidad con otras plataformas

2.1.2.5 Mantenimiento

RNF07	
Título	Modularidad
Descripción	El código fuente de las distintas funciones y métodos del juego debe estar bien modulado para facilitar entendimiento y mantenimiento
Prioridad	Baja

Tabla 2.26; RNF07: Modularidad

2.1.2.6 Retroalimentación

RNF08	
Título	Aclaración de cambios en el juego
Descripción	El juego debe estar retroalimentado, de forma que este muestre, por medio de sonidos o por efectos visuales, que ha ocurrido algo en el juego.

Prioridad	Alta

Tabla 2.27; **RNF08***: Aclaración de cambios en el juego*

2.2 Análisis y diseño del sistema

Una vez determinados todos los requisitos iniciales, se procede a detallar la etapa de análisis y diseño del software desarrollado usando herramientas como los casos de uso o los diagramas de actividad, así como explicaciones detalladas de ciertos algoritmos y mecánicas.

2.2.1 Diagrama de casos de uso

2.2.1.1 Autores

Para el diagrama de casos de uso del proyecto, se debe tener en cuenta que, en el proyecto desarrollado, se pueden encontrar dos claros actores: el sistema y el jugador. Al igual que en la fase de especificación de requisitos, para facilitar la definición de los diferentes requisitos determinados, se usarán diferentes versiones de la siguiente tabla, la tabla 2.28:

Identificador del autor*	
Autor y tipo de autor	Aquí definiremos cual es el autor y el tipo de autor que es *
Descripción	Clarifica que representa el autor en el juego
Interacción	Describe cómo este autor interactúa con el sistema
Requisitos funcionales asociados	Lista de requisitos funcionales relacionados con el autor
Precondiciones	Condiciones que se deben cumplir para que el autor pueda interactuar con el sistema
Postcondiciones	Describe el estado esperado del juego después de que el autor interactúe con el sistema

Tabla 2.28; Plantilla de la tabla del autor

* Identificador del autor: este será un código único que ayudará a identificar a este autor a lo largo de la documentación. Este código estará formado por “Aut”

proveniente de la palabra “autor” concatenado con el numero de la sucesión de autores.

* Tipo de autor: los autores pueden ser de dos tipos: primarios, los cuales interactúan con el sistema directamente, usando las funciones del juego, y secundarios, los cuales son los sistemas externos o entidades que el sistema principal usa para alcanzar sus objetivos.

Aut01	
Tipo de autor	Sistema
Descripción	Es la infraestructura lógica y de software del juego, el cual procesa las entradas del jugador para controlar todo lo relacionado con el juego como el entorno, los NPCs, elementos del juego, …
Interacción	El sistema recibe entradas del jugador y procesa estas entradas, actualizando el estado del juego, y proporcionando feedback visual y auditivo. Además, gestiona eventos automáticos como ataques de enemigos y cambios ambientales.
Requisitos funcionales asociados	RF05, RF09, RF14, RF15, RF16, RF18
Precondiciones	El sistema debe estar operativo para recibir y procesar las entradas del jugador. Esto incluye tener cargados y disponibles los distintos recursos del juego como assets, NPCs, …
Postcondiciones	El sistema, después de procesar las entradas y de generar eventos, actualizará el estado del juego, así como la interfaz de usuario y otros elementos visuales y sonoros.

Tabla 2.29; Aut01: Sistema

Aut02	
Tipo de autor	Primario, Jugador
Descripción	Es el usuario final que interactúa con el juego, controlando al personaje principal usando los diferentes dispositivos de entrada.
Interacción	El jugador interactúa con el juego mediante el uso de los distintos controles para mover el personaje, interactuar con el entorno, disparar, …

Requisitos funcionales asociados	RF01, RF02, RF03, RF04, RF06, RF07, RF08, RF10, RF11, RF12, RF13, RF17
Precondiciones	El juego debe estar cargado o iniciado y en funcionamiento, con el personaje principal listo para poder controlarlo
Postcondiciones	El juego cambia en respuesta a las acciones del jugador, reflejado en nuevas posiciones y acciones del personaje principal.

Tabla 2.30; Aut02: Jugador

2.2.1.2 Ilustración del diagrama de casos de uso

Conociendo detalladamente a los autores y antes de ver los distintos casos de uso de nuestro proyecto, se va a ver el diagrama de casos de uso [49]. Este diagrama es una representación gráfica que especifica, documenta de forma visual las diferentes funciones de un sistema y sus interacciones con los autores.

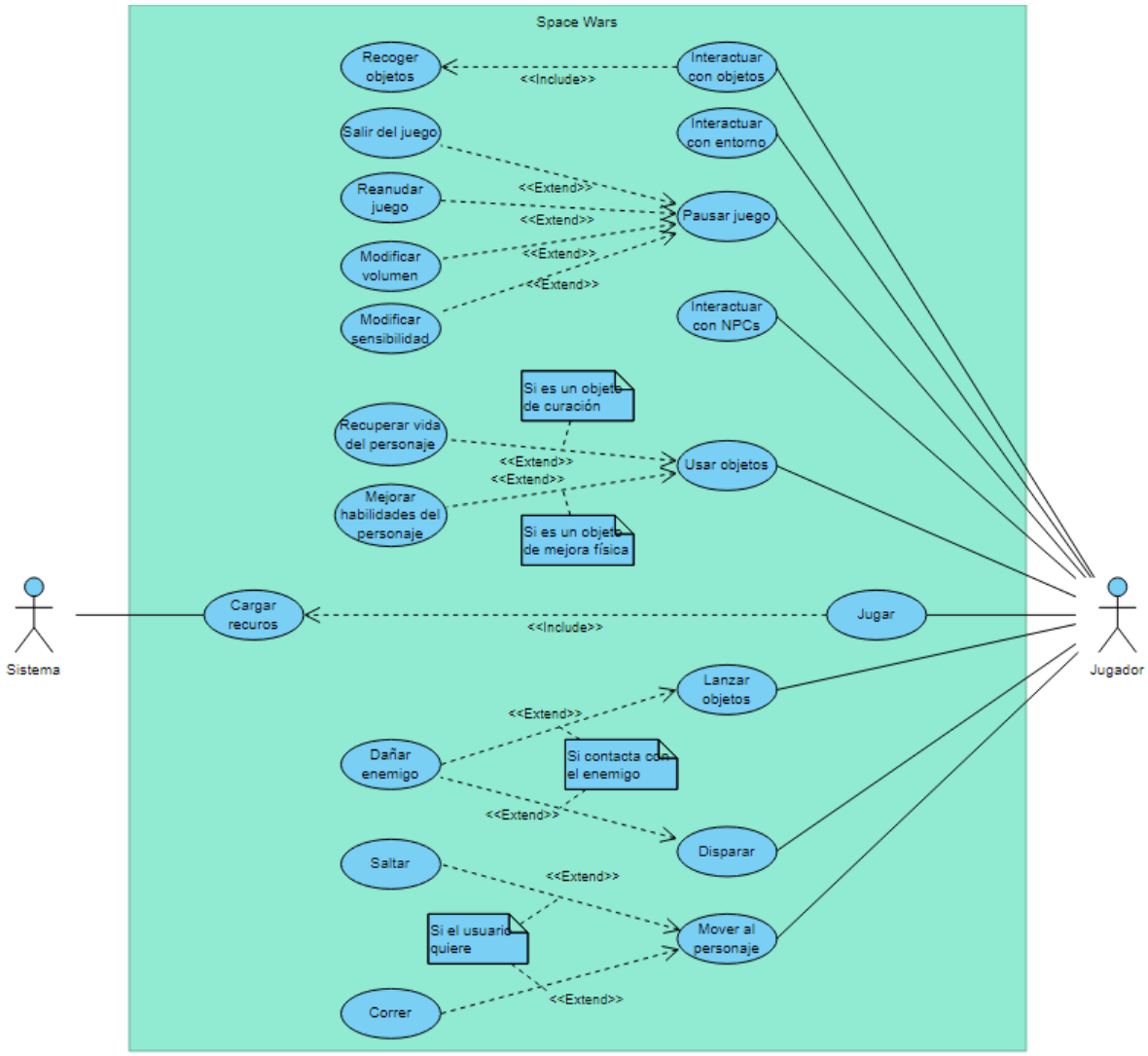


Ilustración 2.1; Diagrama de casos de uso

2.2.2 Casos de uso

A continuación, se procede a ver los casos de uso [50] de nuestro proyecto. Estos serán explicados y definidos mediante la tabla 2.31:

Identificador del caso de uso*	
Nombre del caso de uso	Nombre asociado al caso de uso
Descripción del caso de uso	Descripción detallada que aporta información adicional
Actor/es primarios	Actor/es que interactúa/n directamente con el caso de uso
Actor/es secundarios	Actor/es que interactúa/n indirectamente con el caso de uso

Nivel	Detalle con el que se describe el caso de uso en el sistema
Precondiciones	Condiciones que deben cumplirse para que se pueda ejecutar el caso de uso
Postcondiciones	Estado en el que se debería encontrar el sistema una vez que el caso de uso se complete
Frecuencia de uso	Estimación de cuan a menudo se ejecutará este caso de uso *
Importancia	Prioridad de este caso de uso *
Flujo principal	Lista paso a paso de acciones o eventos que se llevan a cabo durante el caso de uso
Flujos alternativos	Escenarios alternativos que pueden ocurrir durante la ejecución del caso de uso.

Tabla 2.31; Plantilla de tabla de casos de uso

* Identificador del caso de uso: este será un código único que facilitará la identificación del caso de uso a lo largo de la documentación. Este código estará formado por “CU”, las cuales son las letras iniciales de “Caso de Uso”, concatenado con el numero de la sucesión de autores.

* Frecuencia de uso: este campo indicará la cantidad de veces que se realiza el caso de uso durante el juego. Puede ser:

1. Alta: Este caso de uso se realiza muchas veces a lo largo del juego
2. Media: Alguna vez se usa este caso de uso en el desarrollo del juego
3. Baja: A penas se realiza este caso de uso en el juego

* Importancia: este campo indicará lo crítico que es este caso de uso para el desarrollo del juego. Puede ser:

1. **Alta**: Sin este caso de uso, el juego se vería gravemente afectado
2. **Media**: Este caso de uso aporta mucho al juego, pero no es imprescindible
3. **Baja**: El efecto de realizar este caso de uso en el juego es mínimo, por lo que el juego funcionaría sin este caso de uso.

Como aclaración, cuando se habla de “jugador” se está refiriendo al usuario, a la persona real que juega el juego. Mientras que cuando se habla de “personaje” se habla del personaje principal dentro de juego, el cual es controlado por el jugador.

CU01	
Nombre del caso de uso	Jugar
Descripción del caso de uso	Permite al jugador empezar una partida del juego
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El juego debe estar correctamente cargado. 2. El jugador debe estar en el menú principal.
Postcondiciones	El jugador está jugando al juego, por lo que puede interactuar con el entorno de juego.
Frecuencia de uso	Muy alta
Importancia	Alta
Flujo principal	1. El jugador está dentro del menú principal y selecciona la opción “Jugar” 2. El sistema carga el juego y todo lo que conlleva 3. El juego comienza
Flujos alternativos	2.A El sistema no puede cargar el juego al darle a la opción “Jugar” del menú, por lo que se muestra un mensaje de error.

Tabla 2.32; CU01: Jugar

2.2.2.1 Movimiento del personaje

CU02	
Nombre del caso de uso	Mover al personaje
Descripción del caso de uso	Permite al jugador mover el personaje dentro del entorno de juego

Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El movimiento del personaje debe estar disponible
Postcondiciones	El personaje se mueve dentro del juego
Frecuencia de uso	Alta
Importancia	Alta
Flujo principal	1. El jugador usa los controles del movimiento mediante el controlador o los dispositivos de entrada 2. El sistema actualiza la posicion del personaje dentro del juego en función de los datos de entrada recibidos.
Flujos alternativos	2.A El sistema no puede desplazar al jugador debido a algún obstáculo que le impide el movimiento.

Tabla 2.33; CU02: Mover al personaje

CU03	
Nombre del caso de uso	Correr
Descripción del caso de uso	Permite al jugador moverse más rápido
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. Debe estar disponible el movimiento del personaje
Postcondiciones	El jugador se mueve a una velocidad mayor de lo normal por el juego
Frecuencia de uso	Alta
Importancia	Baja
Flujo principal	1. El jugador usa la opción de correr mediante el controlador o los dispositivos de entrada

	2. El sistema aplica el efecto de velocidad de movimiento del personaje al correr.
Flujos alternativos	2.A El efecto de velocidad no es aplicable (por ejemplo, si el personaje intenta atravesar una pared), el sistema impide su uso.

Tabla 2.34; CU03: Correr

CU04	
Nombre del caso de uso	Saltar
Descripción del caso de uso	Permite al jugador hacer que el personaje salte
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El control del movimiento del personaje debe estar activo (tiene que estar en el suelo para poder saltar).
Postcondiciones	El personaje realice un movimiento vertical.
Frecuencia de uso	Media
Importancia	Baja
Flujo principal	1. El jugador pulsa botón de saltar usando el controlador o los dispositivos de entrada 2. El sistema verifica que se puede realizar la acción de salto 3. El personaje realiza un salto vertical
Flujos alternativos	2.A El sistema reconoce que el salto no se puede realizar y no permite que el personaje realice el movimiento

Tabla 2.35; CU04: Saltar

2.2.2.2 Opciones de los menús

CU05	
Nombre del caso de uso	Pausar el juego
Descripción del caso de uso	Permite al jugador pausar el juego sin que este siga transcurriendo. Al pausar el juego, se accede al menú de opciones.
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	El jugador debe estar jugando al juego
Postcondiciones	1. El juego se detiene 2. Se accede al menú de opciones
Frecuencia de uso	Alta
Importancia	Media
Flujo principal	1. El jugador pulsa el botón de pausa 2. El sistema pausa el juego y muestra el menú de opciones.
Flujos alternativos	2.A El sistema no puede detener el juego y/o no se muestra el menú de opciones.

Tabla 2.36; CU05: Pausar el juego

CU06	
Nombre del caso de uso	Salir del juego
Descripción del caso de uso	Permite al jugador salir del juego cerrando la aplicación del sistema
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe estar en algún menú que ofrezca la opción de salir del juego
Postcondiciones	El juego se cierra y se sale de la aplicación.
Frecuencia de uso	Baja

Importancia	Media
Flujo principal	1. El jugador usa los dispositivos de entrada para abrir el menú 2. El jugador selecciona la opción de salir del juego 3. El sistema cierra el juego y libera recursos.
Flujos alternativos	2.A El jugador no selecciona la opción de salir del juego, cierra el menú y sale del juego 2.B Por algún error, no se muestra la opción de salir del juego dentro del menú 3.A El sistema no puede cerrar el juego, por lo que muestra un mensaje de error.

Tabla 2.37; CU06: Salir del juego

CU07	
Nombre del caso de uso	Reanudar el juego
Descripción del caso de uso	Permite al jugador seguir jugando al juego desde donde se quedó después de haberlo pausado
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe haber pausado el juego
Postcondiciones	El juego continúa desde el punto donde fue pausado
Frecuencia de uso	Media
Importancia	Media
Flujo principal	1. El jugador selecciona la opción de reanudar la partida desde el menú de pausa. 2. El sistema cierra el menú de pausa y continúa el juego.
Flujos alternativos	No existen

Tabla 2.38; CU07: Reanudar el juego

CU08	
Nombre del caso de uso	Modificar volumen
Descripción del caso de uso	Permite al jugador modificar el volumen tanto de la música como de los sonidos del juego
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe haber pausado el juego
Postcondiciones	El volumen de los sonidos y/o música del juego ha sido modificado
Frecuencia de uso	Media
Importancia	Baja
Flujo principal	1. El jugador usa los dispositivos de entrada para abrir el menú 2. El jugador mueve la barra de los sonidos y/o música. 3. El sistema cambia el volumen de los sonidos y/o música del juego
Flujos alternativos	2.A El jugador no mueve ninguna barra. En tal caso, el volumen no varía 2.B Por algún error, no se muestran la barra de los sonidos ni la de la música 2.C Por algún error, no se puede variar la barra de los sonidos y/o música. 3.A Tras variar la barra de los sonidos y/o música, el sistema no puede modificar el volumen, por lo que muestra un mensaje de error.

Tabla 2.39; CU08: Modificar el volumen

CU09	
Nombre del caso de uso	Modificar sensibilidad

Descripción del caso de uso	Permite al jugador modificar la sensibilidad de movimiento de la cámara del jugador dentro del juego
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe haber pausado el juego
Postcondiciones	La sensibilidad de la cámara del jugador dentro del juego ha sido modificada
Frecuencia de uso	Media
Importancia	Media
Flujo principal	1. El jugador usa los dispositivos de entrada para abrir el menú 2. El jugador mueve la barra de la sensibilidad. 3. El sistema cambia la sensibilidad de la cámara del jugador dentro del juego
Flujos alternativos	2.A El jugador no mueve ninguna barra. En tal caso, la sensibilidad no varía 2.B Por algún error, no se muestran la barra de la sensibilidad 2.C Por algún error, no se puede variar la barra de la sensibilidad 3.A Tras variar la barra de la sensibilidad, el sistema no puede modificar la sensibilidad, por lo que muestra un mensaje de error.

Tabla 2.40; CU09: Modificar la sensibilidad

2.2.2.3 Interacción

CU10	
Nombre del caso de uso	Interactuar con el entorno

Descripción del caso de uso	Permite al jugador interactuar con elementos del entorno del juego, como pulsar un botón para abrir una puerta.
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe estar cerca y mirando a un elemento interactuable del entorno
Postcondiciones	El estado del juego se actualiza en función del efecto de la interacción
Frecuencia de uso	Media
Importancia	Alta
Flujo principal	1. El jugador usa los controles del movimiento mediante el controlador o los dispositivos de entrada para acercarse y mirar al elemento interactuable 2. El jugador interactúa con el elemento mediante el controlador o los dispositivos de entrada 3. El sistema actualiza el estado del juego.
Flujos alternativos	2.A El jugador no puede interactuar con el elemento 3.A El sistema no puede actualizar el estado del juego, por lo que muestra mensaje de error.

Tabla 2.41; CU10: Interactuar con el entorno

CU11	
Nombre del caso de uso	Interactuar con objetos
Descripción del caso de uso	Permite al jugador interactuar con los objetos repartidos por el entorno del juego
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego

	2. Debe haber objetos disponibles en el entorno 3. Los objetos disponibles deben poder ser interactuables. 4. El personaje debe estar mirando y cerca del objeto interactuable
Postcondiciones	Se realiza la acción de la interacción.
Frecuencia de uso	Media
Importancia	Media
Flujo principal	1. El jugador usa los controles del movimiento mediante el controlador o los dispositivos de entrada para acercarse y mirar al objeto 2. El sistema verifica que se puede interactuar con el objeto 3. El jugador interactúa con el objeto 4. El sistema realiza la acción de la interacción con el objeto.
Flujos alternativos	2.A No se puede interactuar con el objeto, por lo que muestra un mensaje de error. 4.A El sistema no puede realizar la acción de la interacción con el objeto, por lo que muestra mensaje de error.

Tabla 2.42; CU11: Interactuar con objetos

CU12	
Nombre del caso de uso	Interactuar con NPCs (no enemigos)
Descripción del caso de uso	Permite al jugador interactuar con los diferentes NPCs del juego
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe estar cerca y mirando al NPC

Postcondiciones	Se completa la interacción con el NPC y el juego actualiza su estado
Frecuencia de uso	Media
Importancia	Media
Flujo principal	1. El jugador usa los controles del movimiento mediante el controlador o los dispositivos de entrada para acercarse y mirar al NPC 2. El jugador usa botones de dispositivos de entrada para interactuar con el NPC 3. El sistema presenta respuesta de la interacción con el NPC (diálogo) 4. El sistema actualiza el estado del juego
Flujos alternativos	3.A No se puede interactuar con el NPC puesto que el NPC ya no tiene más diálogos disponibles 3.B No se puede interactuar con el NPC por algún error de lógica de programación, por lo que el sistema muestra un mensaje de error. 4.A El sistema no puede actualizar el estado del juego, por lo que muestra un mensaje de error.

Tabla 2.43; CU12: Interactuar con NPCs

2.2.2.4 Combate

CU13	
Nombre del caso de uso	Disparar
Descripción del caso de uso	Permite al jugador que el personaje dispare el arma equipada dentro del entorno de juego
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe tener un arma equipada

Postcondiciones	1. El arma equipada dispara 2. El sistema responde (daña a los enemigos, disminuye munición, …)
Frecuencia de uso	Alta
Importancia	Alta
Flujo principal	1. El jugador usa el controlador o los dispositivos de entrada para disparar 2. El sistema verifica que haya un arma equipada y que haya munición. 3. El sistema hace disparar el arma equipada y actualiza el estado del juego.
Flujos alternativos	2.A El sistema ve que no hay munición, por lo que no puede pasar al paso 3. 3.A El sistema no puede disparar el arma equipada por el personaje, por lo que se muestra una notificación.

Tabla 2.44; CU13: Disparar

CU14	
Nombre del caso de uso	Usar objetos
Descripción del caso de uso	Permite al jugador usar objetos dentro del juego, los cuales tendrán efectos sobre el personaje o dañarán a los enemigos.
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe tener disponible un objeto en el inventario 3. El jugador debe tener el objeto equipado
Postcondiciones	1. El objeto es usado, por lo que:

	1.1 El juego actualiza su estado en base al efecto del objeto usado 1.2 Se decrementa la cantidad de ese objeto en el inventario del usuario.
Frecuencia de uso	Media
Importancia	Media
Flujo principal	1. El jugador selecciona el objeto mediante el controlador o los dispositivos de entrada 2. El sistema verifica que el objeto seleccionado está disponible 3. El sistema aplica el efecto del objeto al personaje o al entorno del juego.
Flujos alternativos	3.A El efecto del objeto no es aplicable (por ejemplo, intentar usar poción de curación con la vida completa), el sistema impide su uso y se notifica al jugador de la razón.

Tabla 2.45; CU14: Usar objetos

CU15	
Nombre del caso de uso	Lanzar objetos
Descripción del caso de uso	Permite al jugador lanzar objetos dentro del juego para que estos tengan un efecto en el entorno del juego
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe tener disponible un objeto lanzable en el inventario 3. El jugador debe tener el objeto equipado
Postcondiciones	1. El objeto es usado, por lo que:

	1.1 El juego actualiza su estado en base al efecto del objeto usado 1.2 Se decrementa la cantidad de ese objeto en el inventario del usuario.
Frecuencia de uso	Media
Importancia	Media
Flujo principal	1. El jugador selecciona el objeto mediante el controlador o los dispositivos de entrada 2. El sistema verifica que el objeto seleccionado está disponible 3. El jugador lanza el objeto 4. El sistema aplica el efecto del objeto al entorno del juego.
Flujos alternativos	2.A El sistema puede comprobar que no hay suficiente cantidad de objetos lanzables en el inventario 3.A El jugador no puede lanzar el objeto, por lo que se muestra un mensaje de error 4.A El sistema no puede aplicar el efecto del objeto al entorno, por lo que muestra un mensaje de error.

Tabla 2.46; CU15: Lanzar objetos

CU16	
Nombre del caso de uso	Mejorar habilidades del personaje
Descripción del caso de uso	Permite al jugador usar objetos para que estos mejoren las habilidades y capacidades físicas del personaje
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe tener objetos disponibles en el inventario
Postcondiciones	Se proporciona el efecto del objeto al personaje

Frecuencia de uso	Media
Importancia	Media
Flujo principal	1. El jugador selecciona el objeto mediante el controlador o los dispositivos de entrada 2. El jugador usa el objeto 3. El sistema actualiza las habilidades y capacidades físicas del personaje en base al efecto del objeto 4. El sistema actualiza las cantidades de ese objeto en el inventario del personaje
Flujos alternativos	2.A El jugador no puede usar el objeto porque no tiene cantidad suficiente de ese objeto en el inventario 3.A El sistema no puede actualizar las habilidades del personaje puesto que ya tiene el efecto del objeto aplicado 3.B El sistema no actualiza las habilidades y capacidades físicas del personaje, por lo que muestra mensaje de error 4.A El sistema no puede actualizar las cantidades de ese objeto, por lo que muestra un mensaje de error.

Tabla 2.47; CU16: Mejorar habilidades del personaje

CU17	
Nombre del caso de uso	Recuperar vida del personaje
Descripción del caso de uso	Permite al jugador hacer que el personaje recupere vida
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe tener suficientes objetos de curación en el inventario
Postcondiciones	1. El personaje restaura parte de su vida

	2. Disminuye cantidades de ese objeto en el inventario
Frecuencia de uso	Media
Importancia	Media
Flujo principal	1. El jugador usa el objeto de curación mediante el controlador o los dispositivos de entrada 2. El sistema verifica que el jugador tiene suficientes objetos de curación en el inventario 3. El sistema actualiza la salud del personaje dentro del juego.
Flujos alternativos	2.A El sistema determina que el jugador no tiene suficientes objetos de curación en el inventario, por lo que no aplica cura al personaje. 3.A El sistema determina que la salud del personaje está al máximo, por lo que no actualiza su salud. 3.B El sistema no puede terminar la cantidad de salud del personaje, por lo que muestra mensaje de error

Tabla 2.48; CU17: Recuperar vida del personaje

CU18	
Nombre del caso de uso	Dañar al enemigo
Descripción del caso de uso	El jugador le inflige daño al enemigo por la colisión con un proyectil del arma del personaje o con un el efecto de un objeto lanzable.
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe tener disponible: 2.1 Un objeto lanzable en el inventario 2.2 Un arma 3. El jugador debe tener:

	3.1 El objeto lanzable equipado 3.2 El arma equipada
Postcondiciones	El enemigo sufre daño y su vida decrece
Frecuencia de uso	Alta
Importancia	Alta
Flujo principal	1. El jugador selecciona el objeto o arma mediante el controlador o los dispositivos de entrada 2.1 El sistema verifica que el objeto seleccionado está disponible 2.2 El sistema verifica que hay munición suficiente del arma seleccionada 3. El sistema aplica daño al enemigo y su vida se actualiza.
Flujos alternativos	2.1.A El sistema comprueba que no hay cantidad suficiente del objeto seleccionado, por lo que muestra un mensaje de error 2.2.B El sistema comprueba que no hay munición suficiente del arma seleccionada, por lo que muestra un mensaje de error 3.A El sistema no actualiza la vida del enemigo, por lo que muestra un mensaje de error.

Tabla 2.49; CU18: Dañar al enemigo

CU19	
Nombre del caso de uso	Recoger objetos
Descripción del caso de uso	Permite al jugador recoger los objetos repartidos por el entorno del juego para luego usarlos.
Actor/es primarios	Jugador
Actor/es secundarios	Sistema
Nivel	Usuario
Precondiciones	1. El jugador debe estar jugando al juego 2. El jugador debe haber interactuado con el objeto

Postcondiciones	1. La cantidad del objeto recogido aumenta en el inventario 2. El objeto recogido desaparece del entorno
Frecuencia de uso	Media
Importancia	Media
Flujo principal	1. El jugador recoge el objeto 2. El sistema verifica que el objeto puede ser recogido 3. El sistema añade el objeto al inventario del usuario 4. El sistema elimina el objeto del entorno.
Flujos alternativos	2.A El objeto no puede ser recogido, por lo que muestra un mensaje de error. 3.A El sistema no puede añadir el objeto al inventario del usuario, por lo que muestra mensaje de error. 4.A El sistema no puede eliminar el objeto del entorno del juego, por lo que muestra un mensaje de error

Tabla 2.50; CU19: Recoger objetos

2.2.2.5 Funciones del sistema

CU20	
Nombre del caso de uso	Cargar recursos
Descripción del caso de uso	Carga los recursos necesarios para que el juego funcione correctamente
Actor/es primarios	Sistema
Actor/es secundarios	Jugador
Nivel	Sistema
Precondiciones	El jugador ha iniciado el juego
Postcondiciones	El juego está listo para ser jugado puesto que todos sus recursos están cargados
Frecuencia de uso	Alta
Importancia	Alta

Flujo principal	1. El jugador inicia el juego usando el controlador o los dispositivos de entrada 2. El sistema carga los recursos del juego. 3. El juego está listo para ser usado
Flujos alternativos	2.A El sistema no puede cargar los recursos del juego, por lo que el juego no inicia y se muestra un mensaje de error.

Tabla 2.51; CU20: Cargar recursos

2.2.3 Diseño arquitectónico

En este apartado se especifica el diseño arquitectónico del proyecto, con el cual se puede apreciar en detalle el proyecto que se espera desarrollar. Con este diagrama se intenta aclarar la estructura del proyecto y lo que se espera conseguir.

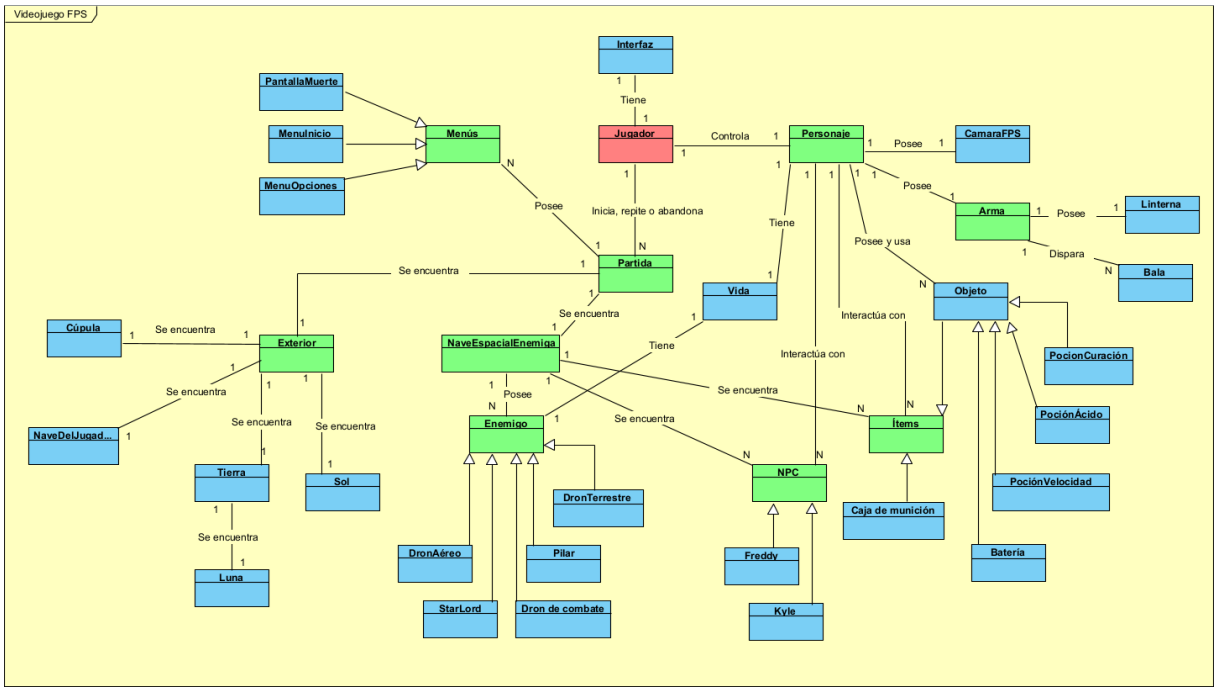


Ilustración 2.2; Diseño arquitectónico del proyecto

En este diagrama, se pueden apreciar que las entidades están de diferente color. Estos colores tienen significados:

1. **Verde**: Las clases más importantes y las bases desde la que partirán otras clases.
2. **Rojo**: Es el elemento indispensable del juego, el jugador. Sin él, no se podría hacer nada.
3. **Azul**: Las clases que forman parte del proyecto pero que no son imprescindibles para él.

2.2.4 Diseño de la interfaz

En este apartado se va a detallar el diseño de las distintas interfaces que se pueden encontrar en el juego. Estas son:

1. Menú principal
2. Menú de opciones
3. Pantalla de muerte
4. Interfaz del jugador

Para comentar el diseño de las distintas interfaces, se ha realizado, para cada una, su sketch, wireframe y mockup. Para organizar la explicación, se hablará de cada interfaz por separado y, dentro de la explicación de cada interfaz, se detallarán los distintos elementos de diseño.

Cabe mencionar que, tanto para la realización del wireframe como para del mockup de cada interfaz, se han realizado para una pantalla de resolución 800 píxeles de ancho y 600 de alto. Además, para seguir una norma entre todos los diseños, estos se han regido a una cuadrilla de 12 columnas de 48 píxeles cada una con una separación de 2 píxeles entre cada columna. Para que se pueda apreciar que todos los wireframe y mockups tienen el mismo tamaño y que está todo correcto, se aporta, para cada wireframe y mockup, una versión con rejillas y otra sin rejillas.

Por último, al final de la explicación de cada interfaz, se proporcionará un diagrama que representa el flujo de navegación entre interfaces.

2.2.4.1 Menú principal

A continuación, se muestra los elementos del diseño del menú principal.

2.2.4.1.1 Sketch

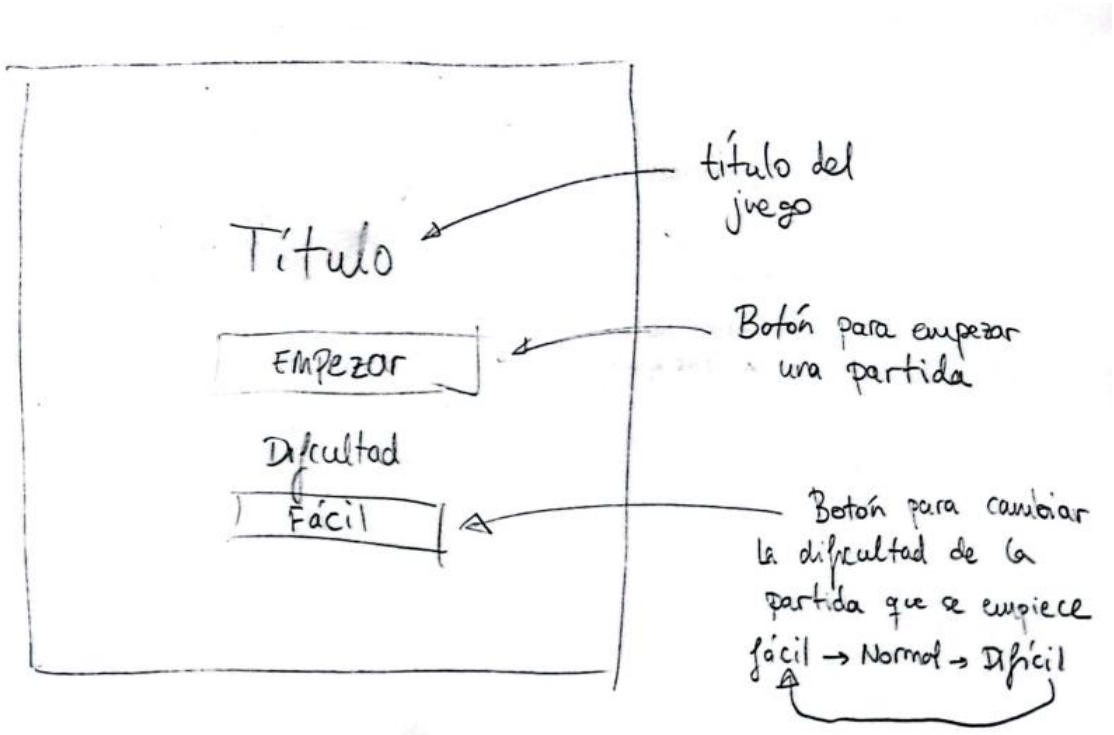


Ilustración 2.3; Sketch del menú principal

2.2.4.1.2 Wireframe

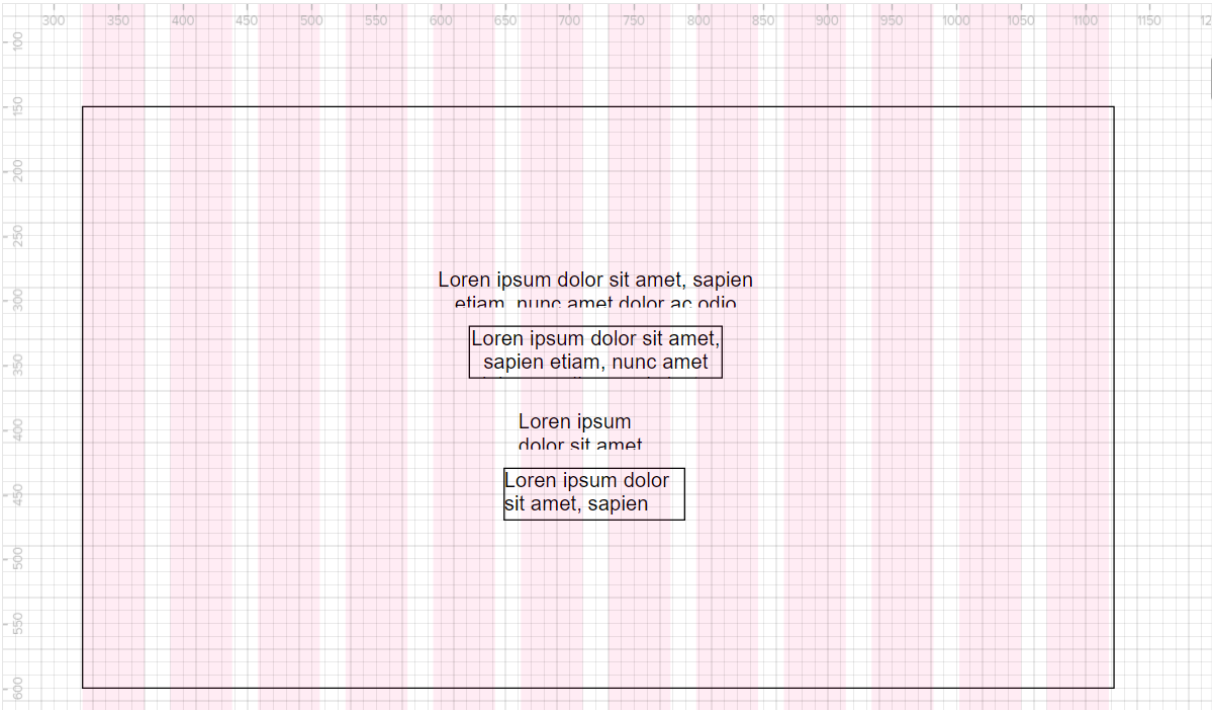


Ilustración 2.4; Wireframe del menú principal, con rejilla

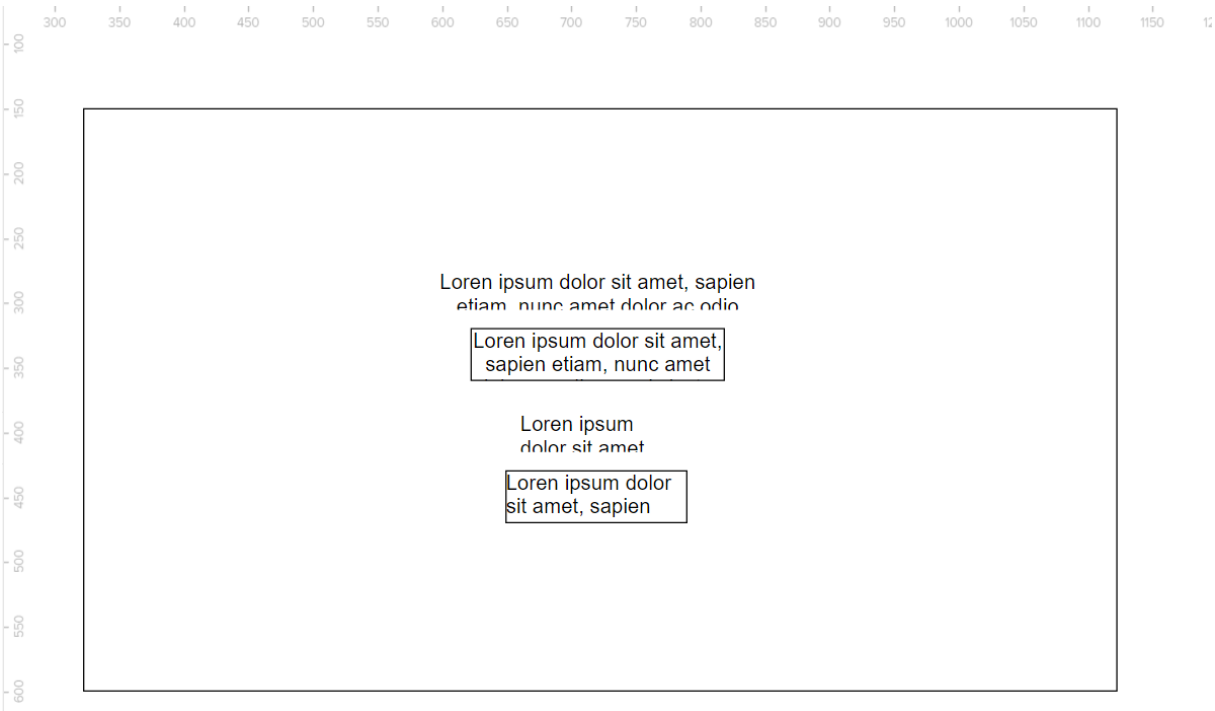


Ilustración 2.5; Wireframe del menú principal, sin rejilla

2.2.4.1.3 Mockup

Ilustración 2.6; Mockup del menú principal, con rejilla

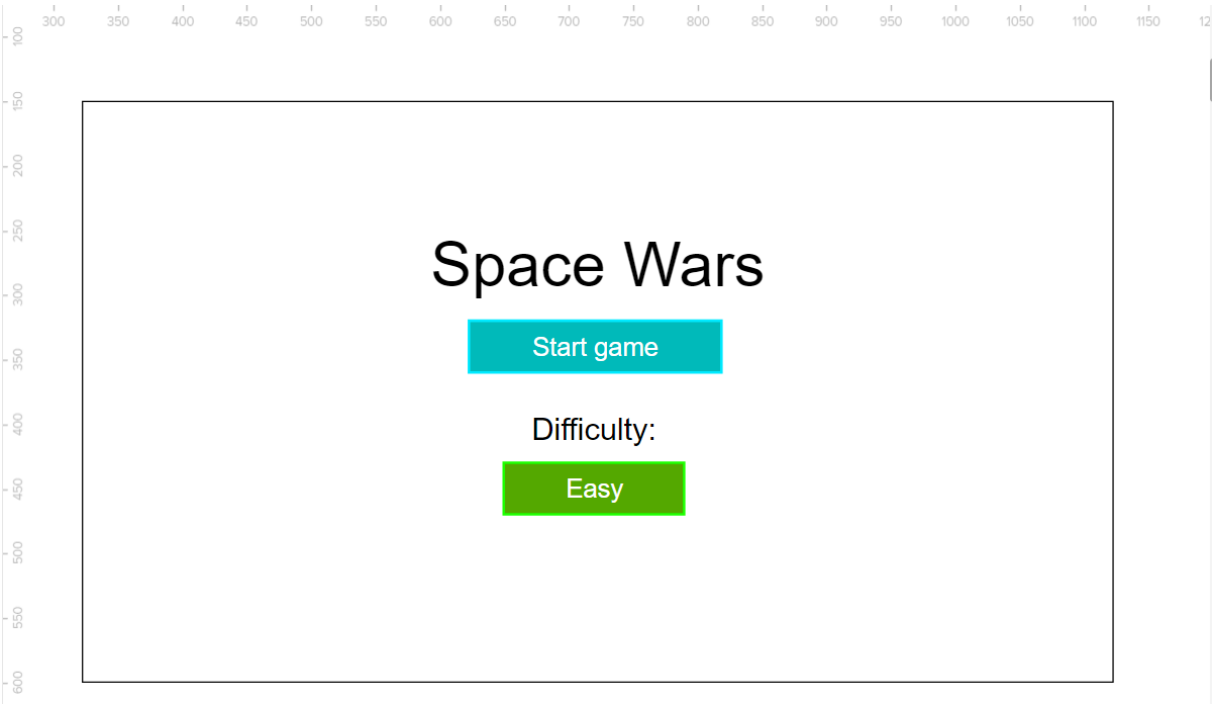


Ilustración 2.7; Mockup del menú principal, sin rejilla

2.2.4.2 Menú de opciones

A continuación, se muestra los elementos del diseño del menú de opciones

2.2.4.2.1 Sketch

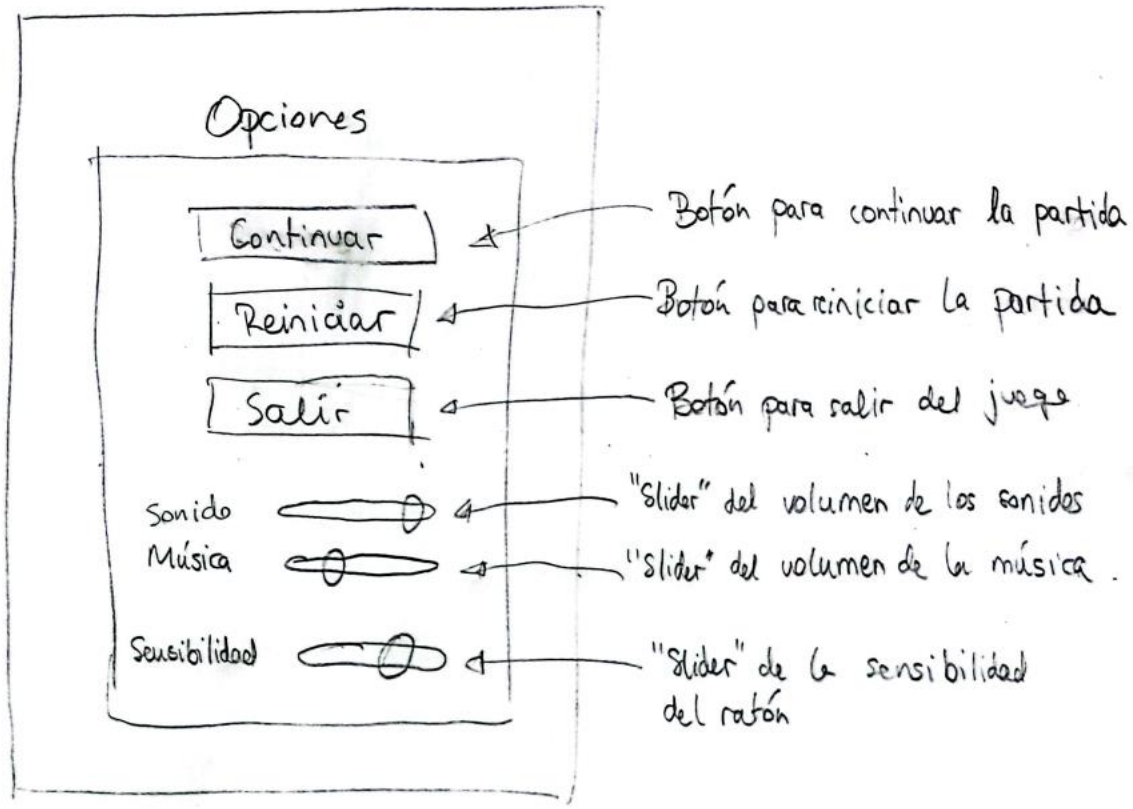


Ilustración 2.8; Sketch del menú de opciones

2.2.4.2.2 Wireframe

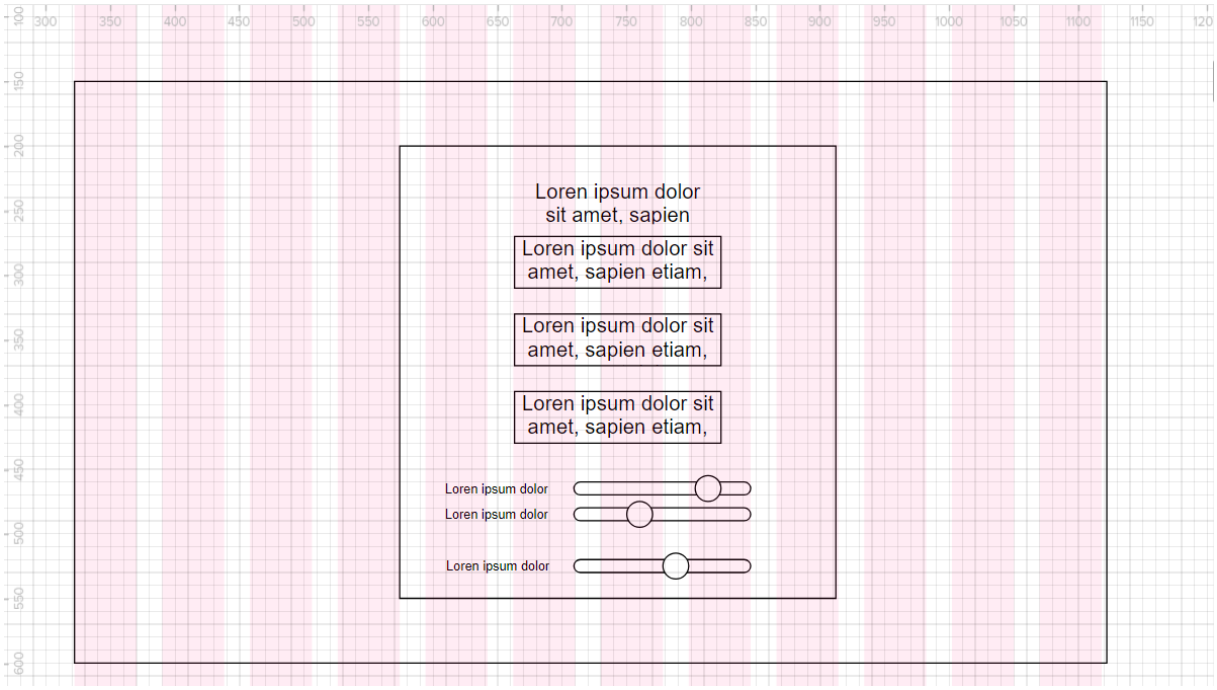


Ilustración 2.9; Wireframe del menú de opciones, con rejilla

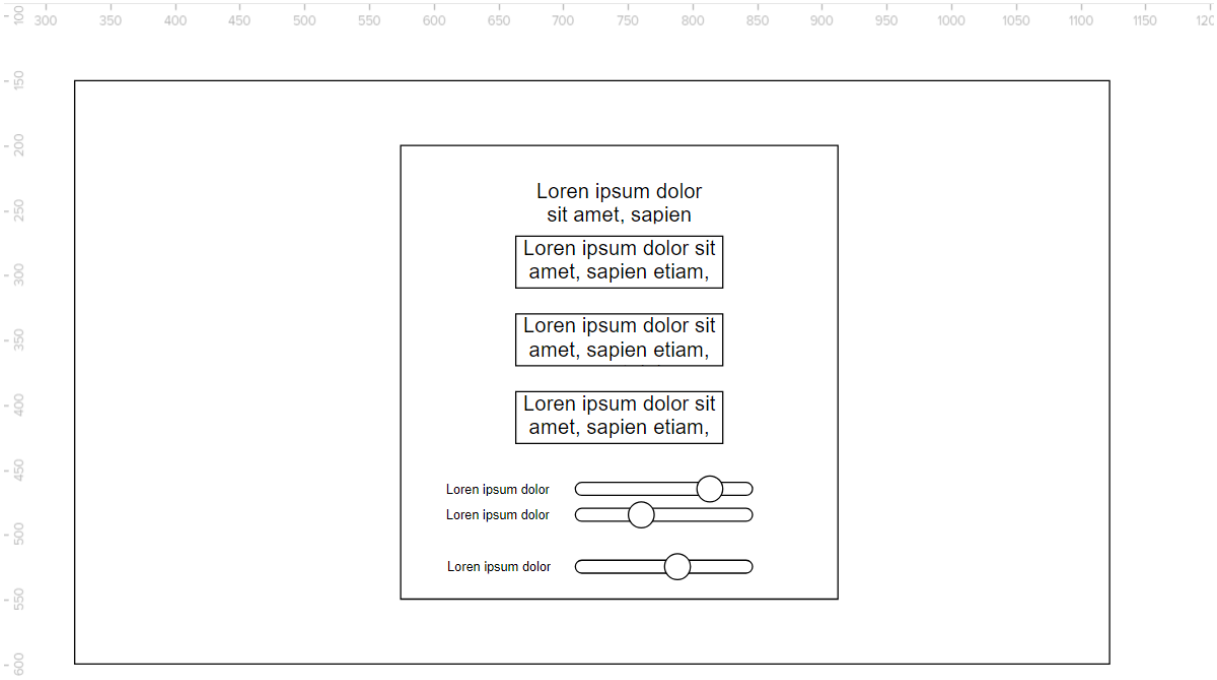


Ilustración 2.10; Wireframe del menú de opciones, sin rejilla

2.2.4.2.3 Mockup

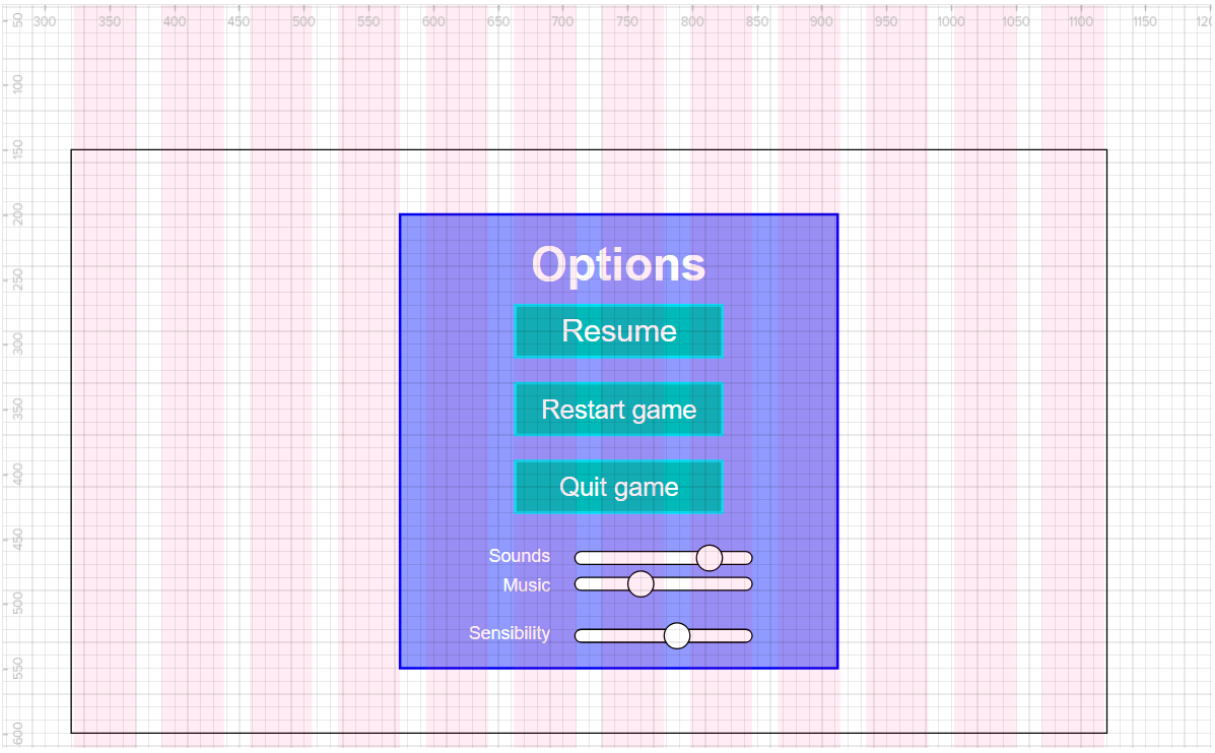


Ilustración 2.11; Mockup del menú de opciones, con rejilla

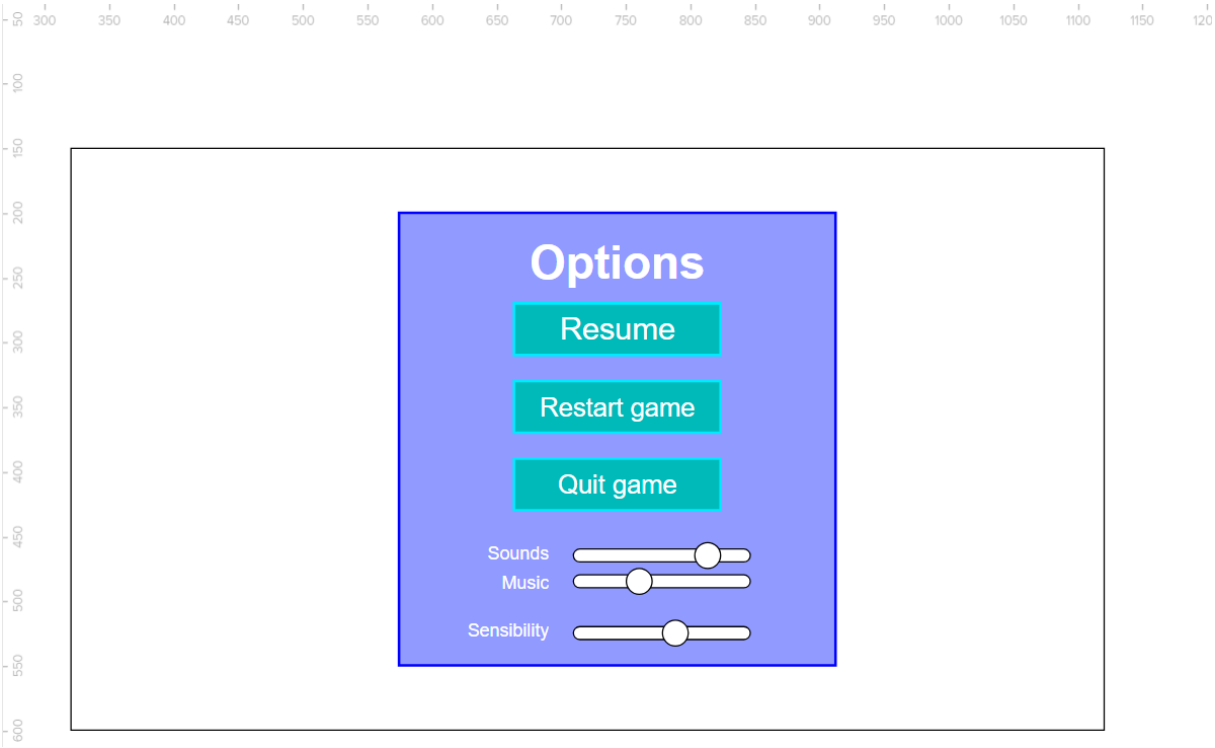


Ilustración 2.12; Mockup del menú de opciones, sin rejilla

2.2.4.3 Pantalla de muerte

A continuación, se muestra los elementos del diseño de la pantalla de muerte

2.2.4.3.1 Sketch

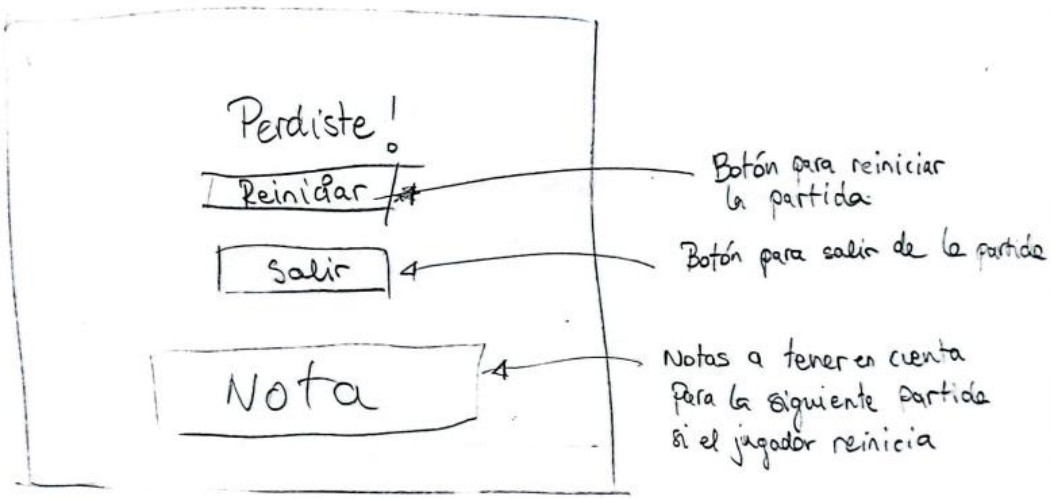


Ilustración 2.13; Sketch de la pantalla de muerte

2.2.4.3.2 Wireframe

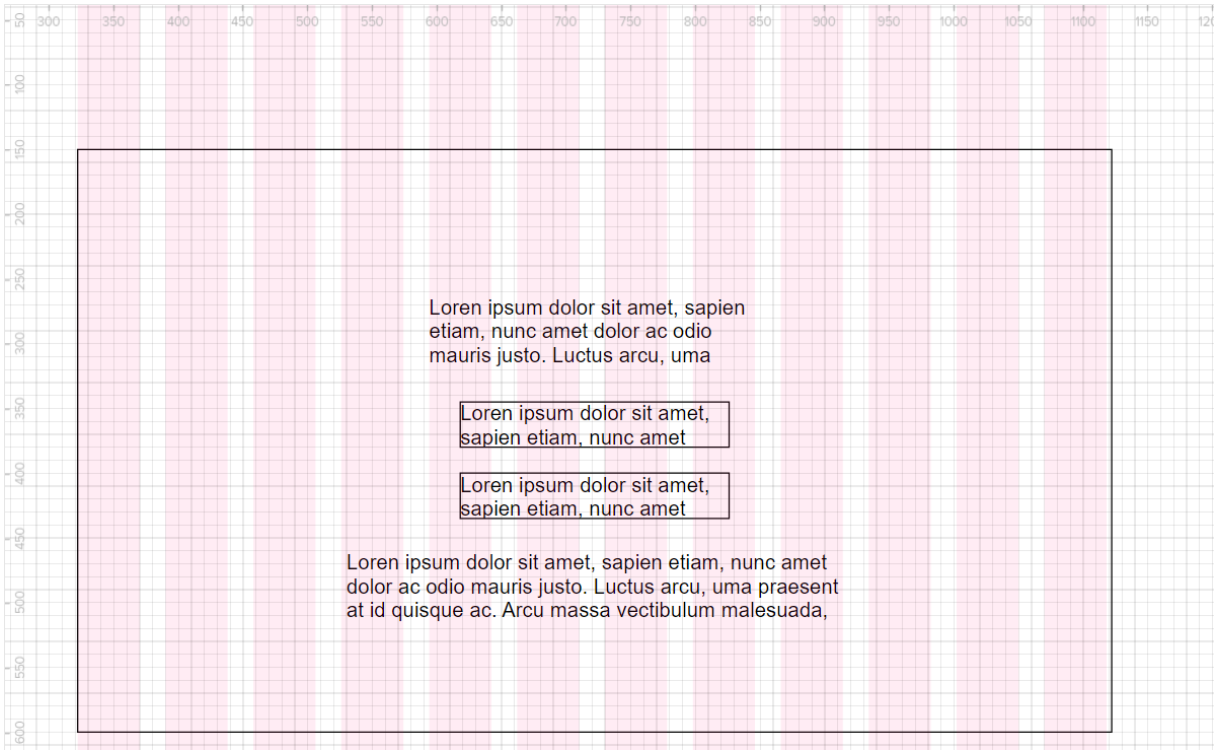


Ilustración 2.14; Wireframe de la pantalla de muerte, con rejilla

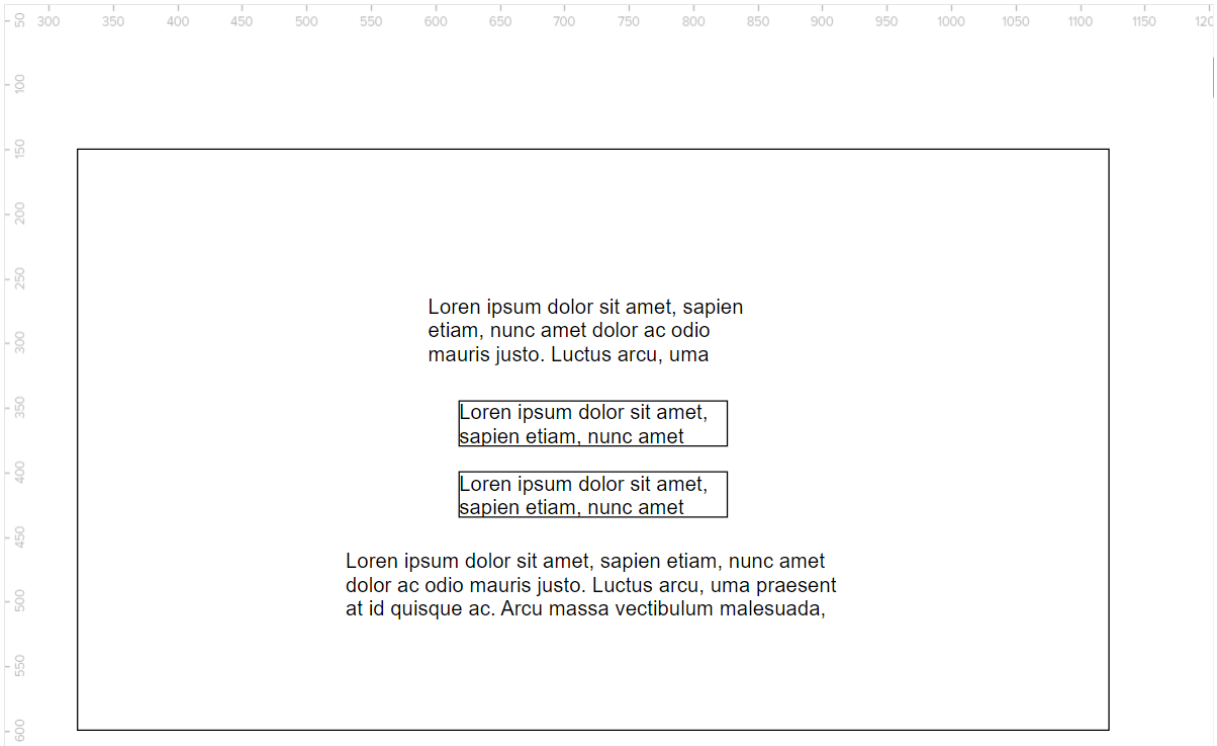


Ilustración 2.15; Wireframe de la pantalla de muerte, sin rejilla

2.2.4.3.3 Mockup

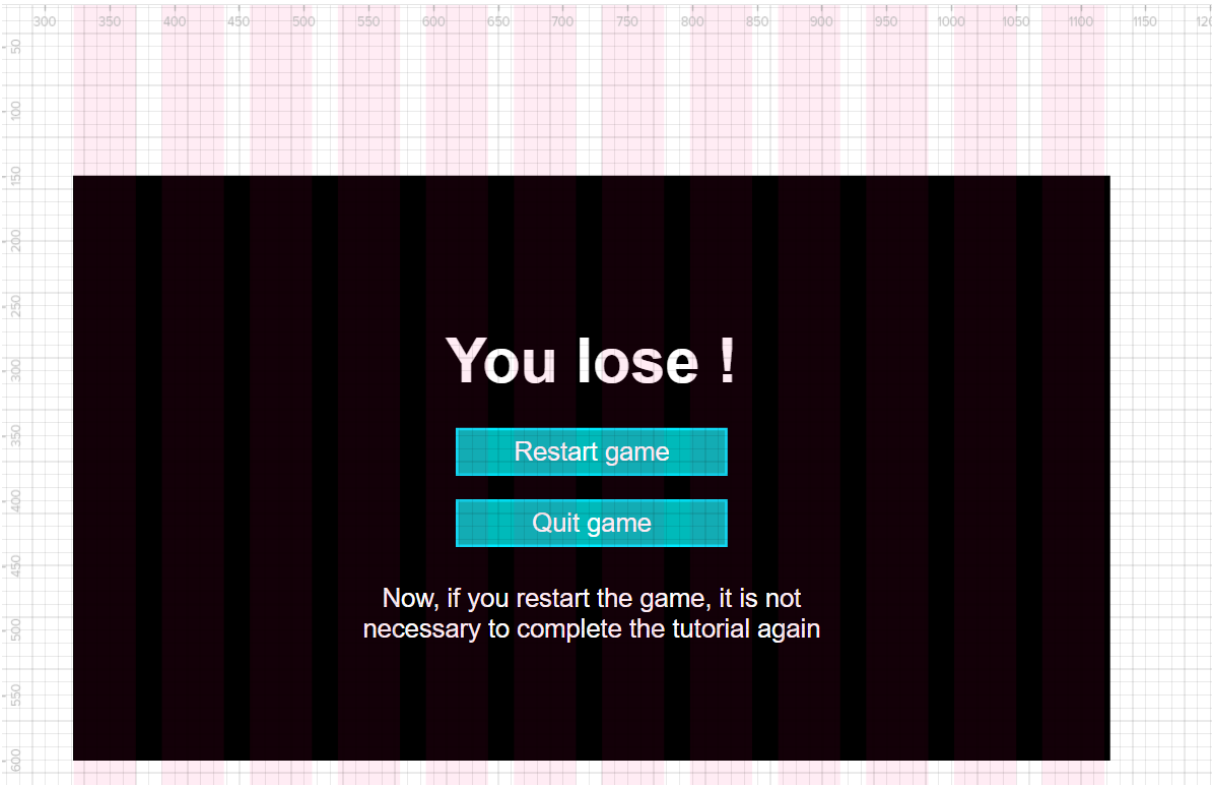


Ilustración 2.16; Mockup de la pantalla de muerte, con rejilla

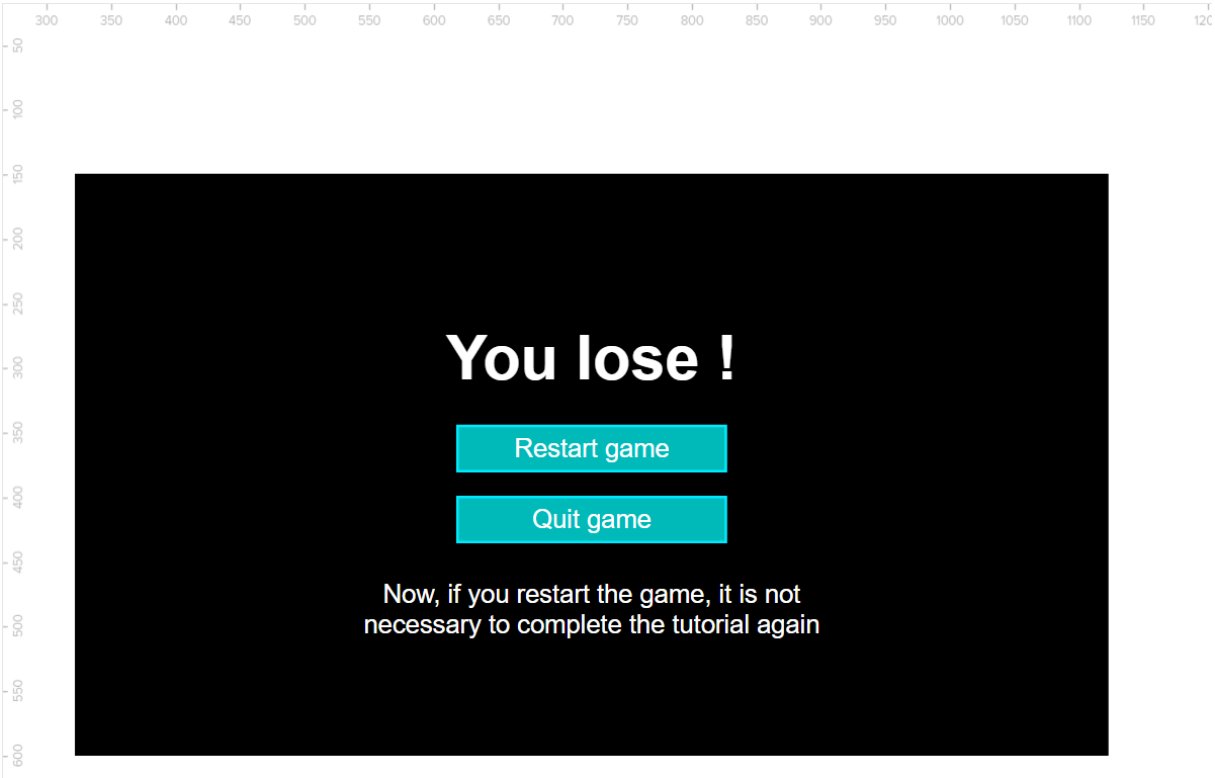


Ilustración 2.17; Mockup de la pantalla de muerte, sin rejilla

2.2.4.4 Interfaz del jugador

A continuación, se muestra los elementos del diseño de la interfaz del jugador

2.2.4.4.1 Sketch

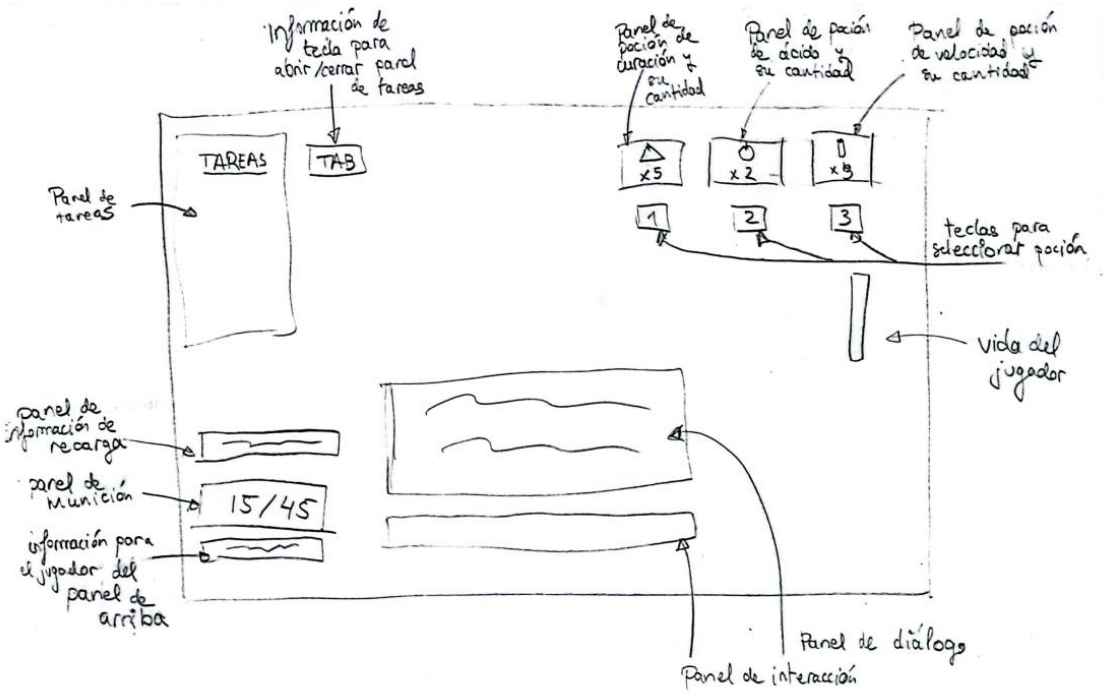


Ilustración 2.18; Sketch de la interfaz del jugador

2.2.4.4.2 Wireframe

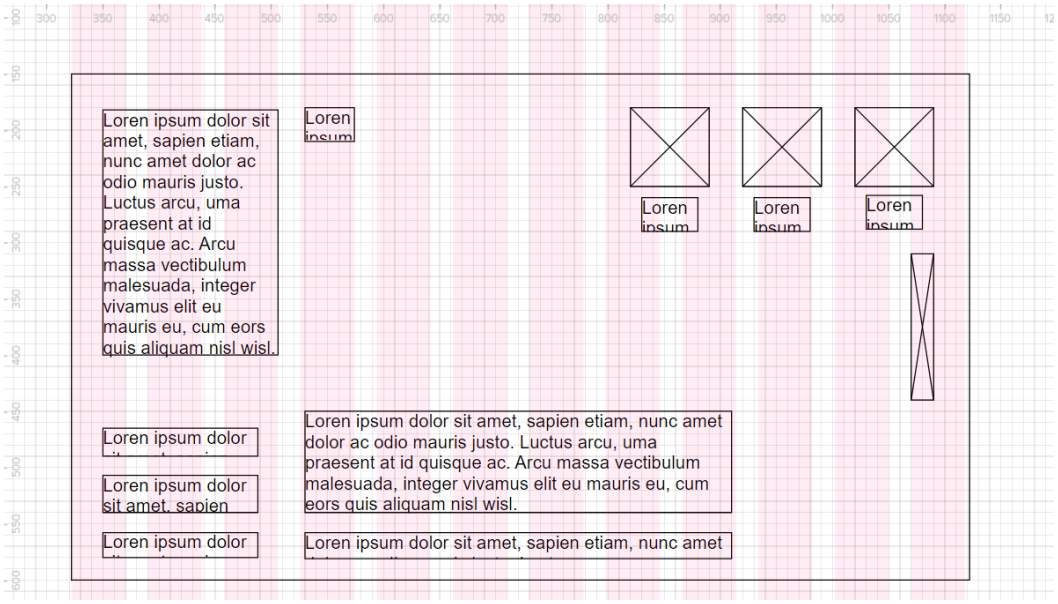


Ilustración 2.19; Wireframe de la interfaz del jugador, con rejilla

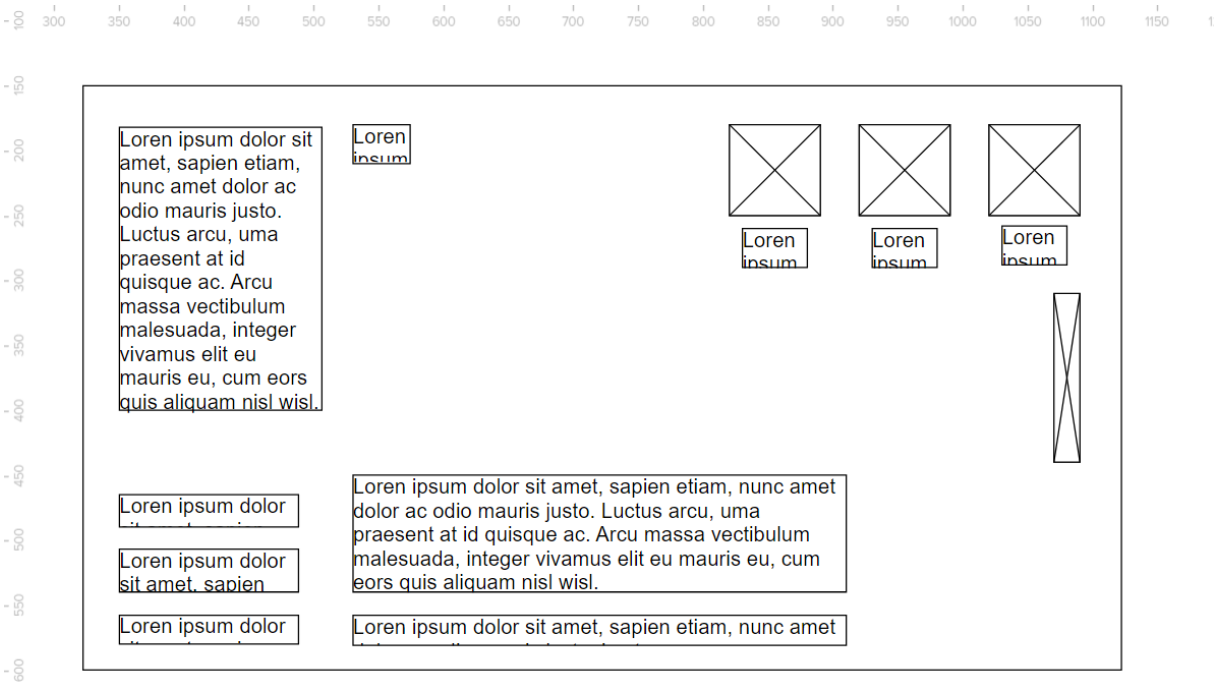


Ilustración 2.20; Wireframe de la interfaz del jugador, sin rejilla

2.2.4.4.3 Mockup

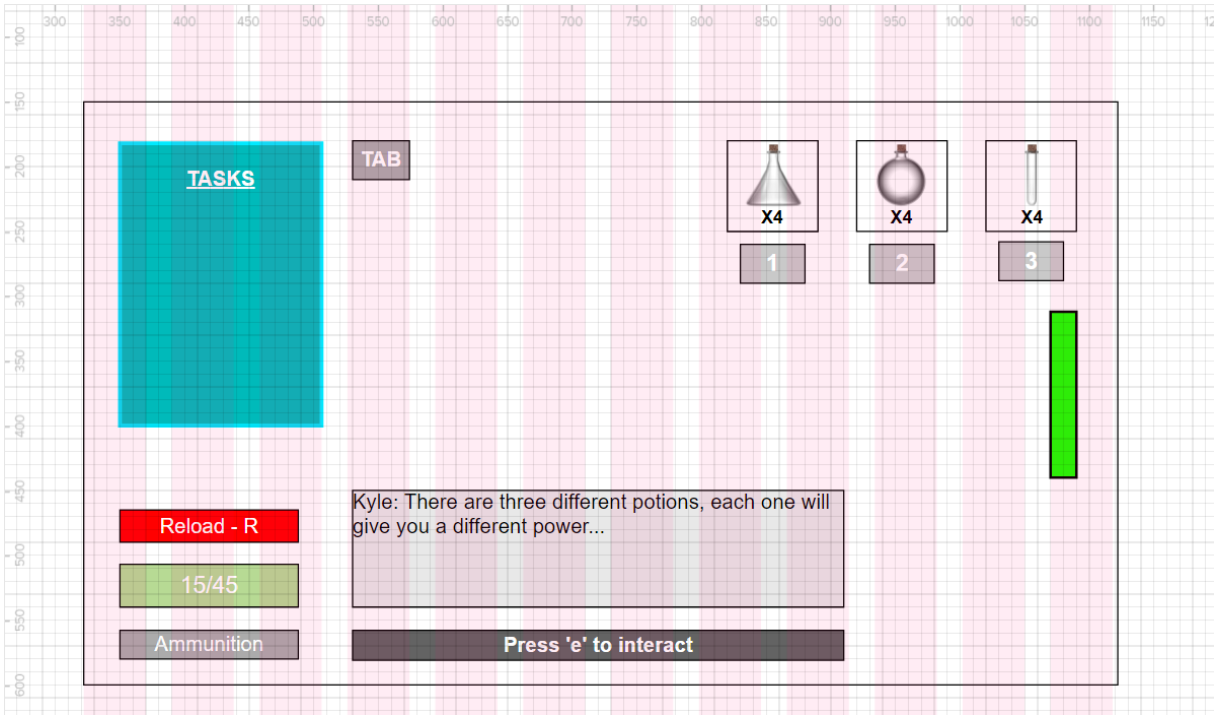


Ilustración 2.21; Mockup de la interfaz del jugador, con rejilla

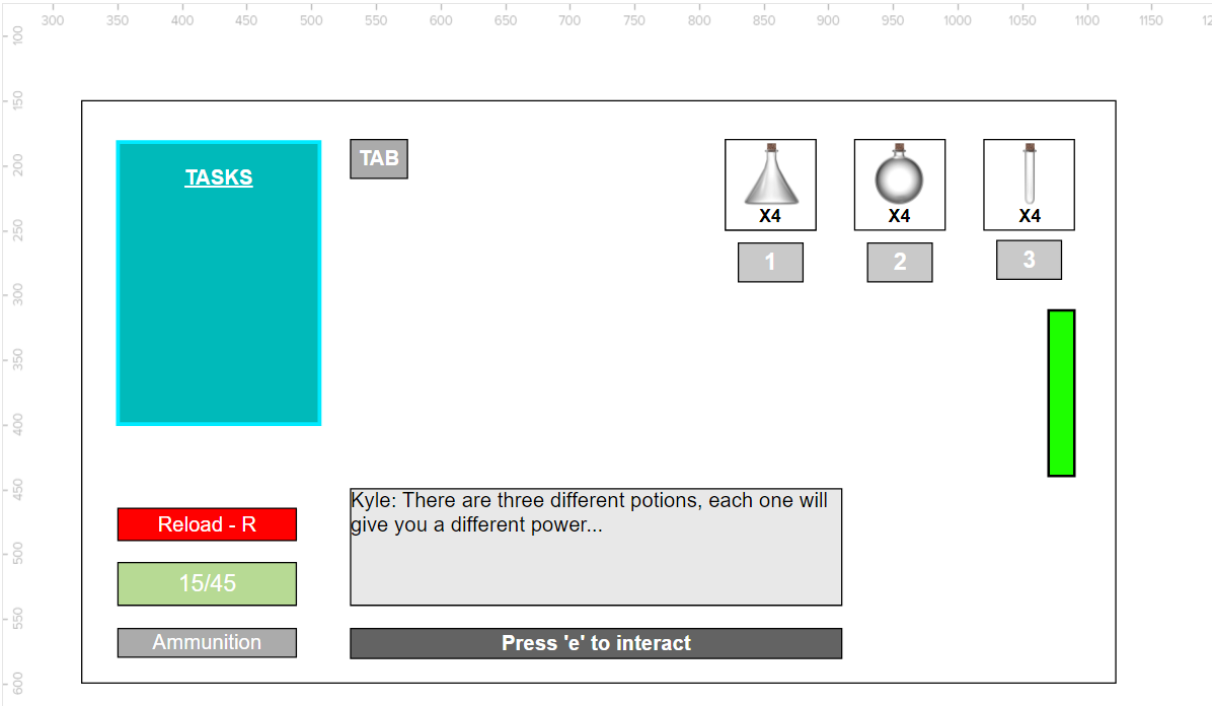


Ilustración 2.22; Mockup de la interfaz del jugador, sin rejilla

2.2.4.1 Diagrama de flujo de navegación

Todos diseños hechos se pueden interconectar mediante el diagrama de flujo de navegación, el cual une las distintas interfaces mediante la acción necesaria a realizar en una interfaz para llegar a otra interfaz.

Cabe destacar que se han usado los diseños de mockup de cada interfaz para realizar el diagrama de flujo de navegación puesto que es el diseño más claro y limpio visualmente, y además, es el diseño con el que el diagrama queda mucho más fácil de entender.

Una cosa a mencionar del diagrama es que, cada interfaz está enumerada por números del 1 al 4, con orden en función de la aparición de la misma en una partida normal. Estos son:

1. Menú principal
2. Interfaz del usuario

3. Menú de opciones
4. Pantalla de muerte

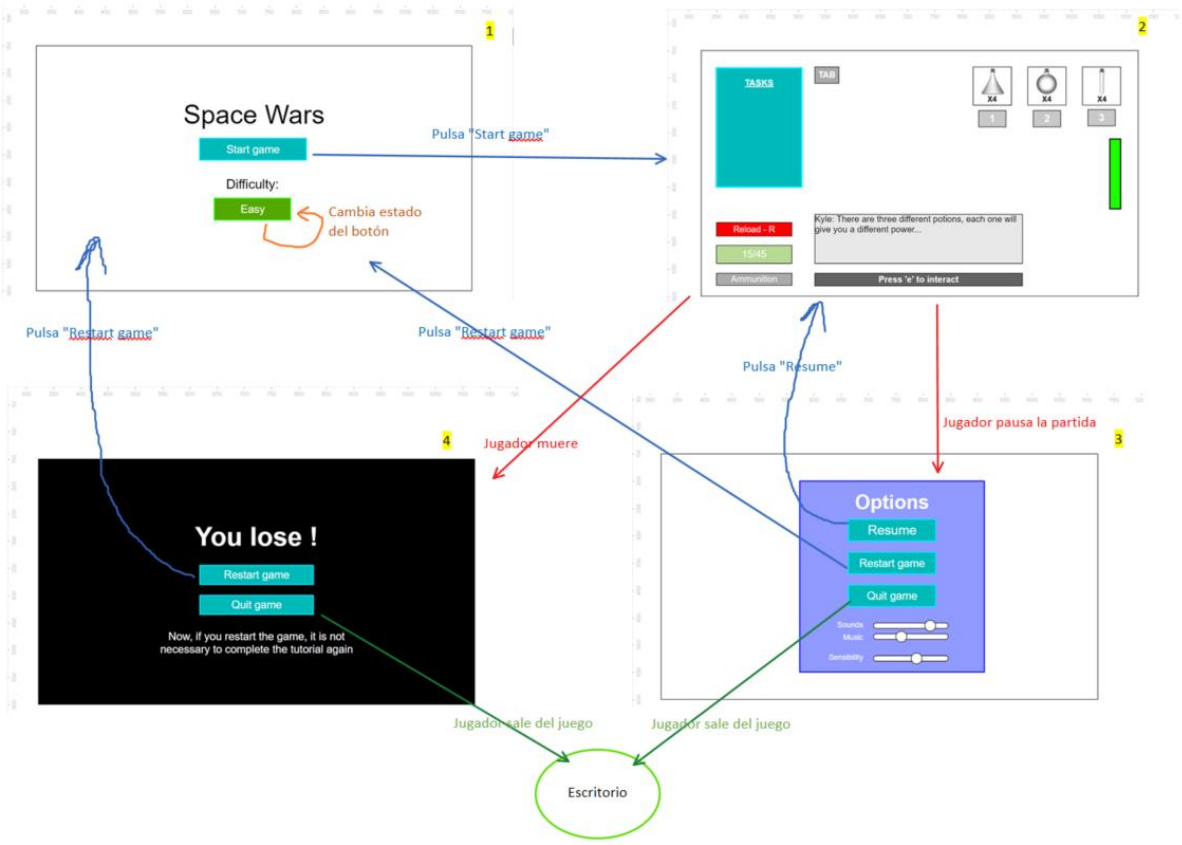


Ilustración 2.23; Diagrama de flujo de navegación entre interfaces

3 DESARROLLO

En este proyecto, como se ha comentado anteriormente, se ha optado por ad optar una metodología de desarrollo ágil basada en iteraciones, que se caracteriza por su enfoque iterativo e incremental. Teniendo en cuenta esto y teniendo acordados todos los requisitos iniciales, se puede empezar el desarrollo del proyecto.

Como se sabe, la metodología de desarrollo ágil basada en iteraciones se caracteriza por realizar el desarrollo del proyecto en iteraciones llamadas **sprints** con una duración de entre una y cuatro semanas.

En este capítulo, el **apartado 3.1 Iteraciones**, solo se detallarán las tareas a realizar en cada sprint y sobre qué trata cada tarea, hablando del contenido de cada tarea sin profundizar demasiado. Mientras que, en el siguiente capítulo, el **apartado 3.2 Funcionalidades desarrolladas**, se profundizará sobre las distintas funcionalidades desarrolladas en el proyecto.

3.1 Iteraciones

En esta sección, se van a ver cada uno de los sprints o iteraciones. En cada sprint, se tendrá:

1. Se verán las tareas a realizar durante el sprint, las restantes y las que ya se han realizado.
    1. Para determinarlas, se usará la tabla 1, ubicada en el **apartado 1.15 Estimación de tamaño y esfuerzo**, la cual indica la estimación de tareas estudiada anteriormente en ese apartado. Con esa tabla, las tareas a realizar en el sprint estarán marcadas con un color Amarillo, mientras que las tareas completadas estarán marcadas con Verde. Además, para detallar aún más, se marcarán con un color Gris las tareas realizadas en el sprint justo anterior. Haciendo esta diferencia de tareas por colores, se puede determinar que el **Backlog** del proyecto, serán las tareas no marcadas por ningún color. Esta aclaración visual con colores, facilitará el entendimiento del reparto de tareas para cada sprint.
2. Se tendrá un resumen temporal del sprint con un **diagrama Burndown** para saber el ritmo que se lleva a la hora de desarrollar en comparación con lo planteado, ya que este diagrama muestra la cantidad de trabajo **restante** en el sprint a lo largo del tiempo.
3. Además, se tendrá otro resumen temporal del sprint con un **diagrama de velocidad**, el cual aportará la información del trabajo del sprint **realizado** a lo largo del tiempo.

4. Por último, se tendrá una demostración del trabajo conseguido al final de cada sprint

Sabiendo el contenido de cada sprint, y que cada sprint comprenderá 18 puntos de historia aproximadamente, se puede empezar a detallar cada iteración.

Cabe destacar que, en las tablas de información de los diagramas de velocidad y Burndown, el día 0 “no cuenta”, ya que se considera el punto inicial, no un día de trabajo dentro del sprint.

3.1.1 Primera iteración

3.1.1.1 Introducción

Siempre, en todos los proyectos cuyo desarrollo se ha realizado siguiendo la metodología ágil Scrum, el sprint inicial es muy importante. Esto es así, puesto que sirve para saber si el ritmo de trabajo que se ha planificado está correctamente estimado o no.

3.1.1.2 Tareas

Como se ha comentado en el **apartado 1.15 Estimación del tamaño y esfuerzo**, todos los sprints deben suponer 18 puntos de historia. Por lo tanto, teniendo la tabla 1, la cual informa de la estimación del esfuerzo de las tareas, también determinada en el **apartado 1.15 Estimación del tamaño y esfuerzo**, se puede decir con certeza que las tareas a realizar en esta primera iteración son las que están marcadas en amarillo en la siguiente tabla:

Tarea	Story points (SP)	Horas de trabajo (aproximadas)
Desarrollo del mapa o entorno de juego (nivelado)**:** 1. Cúpula exterior	11 SP -> 13 SP: 1. 1 SP 2. 3 SP	33 -> 36 horas: 1. 3 horas 2. 9 horas

2. Primera planta 3. Ascensor 4. Segunda planta 5. Tercera planta	3. 2 SP 4. 3 SP 5. 2 SP	3. 6 horas 4. 9 horas 5. 6 horas
Implementación del personaje principal	1 SP	3 horas
Implementación del movimiento del personaje	2 SP	6 horas
Implementación del movimiento de la cámara principal	2 SP	6 horas
Implementación de arma del personaje	1 SP	3 horas
Implementación de funcionalidad del arma del personaje (disparar)	3 SP	9 horas
Mejora del arma y la munición	3 SP	9 horas
Diseño de las interfaces: 1. Menú principal 2. Menú de pausa 3. Menú de muerte 4. PlayerUI	8 SP: 1. 2 SP 2. 2 SP 3. 2 SP 4. 2 SP	24 horas: 1. 6 horas 2. 6 horas 3. 6 horas 4. 6 horas
Implementación de enemigos**:** 1. Enemigo con ataque cuerpo a cuerpo 2. Enemigo con ataque a distancia 3. Jefe final	10 SP -> 13 SP: 1. 3 SP 2. 3 SP 3. 4 SP	33 -> 36 horas: 1. 9 horas 2. 9 horas 3. 15 horas
Interacción con el entorno	5 SP	15 horas
Implementación de NPCs (no enemigos): 1. Kyle 2. Freddy	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Menús y sus funcionalidades	3 SP	9 horas
Implementación de objetos:	6 SP -> 8 SP: 1. 3 SP	18 -> 24 horas: 1. 9 horas

1. Modelado, texturas y materiales 2. Funcionalidad	2. 3 SP	2. 9 horas
Interacción con objetos	5 SP	15 horas
Animación de todos los modelos de la escena	5 SP	15 horas
Decoración del mundo	3 SP	9 horas
Implementación de audio y efectos sonoros del juego	3 SP	9 horas
TOTAL:	86 Story points	252 horas

Tabla 3.1; Estimación de tareas del sprint1

Estas han sido las tareas seleccionadas puesto que la sumatoria de sus story points son: 13 + 1 + 2 + 2 = 18 SP, lo anteriormente planeado para un sprint

3.1.1.3 Desarrollo del sprint

Una vez determinadas las tareas del primer sprint del desarrollo, se empezaron a realizar. Las tareas no fueron para nada complejas, simplemente fueron tediosas puesto que el nivelado y las demás implementaciones no conllevan un gran conocimiento del área ni técnicas avanzadas, solo era cuestión de dedicarle bastante tiempo.

Tras la realización del primer sprint, se obtuvo una versión bastante jugable del juego, en la que nos encontraríamos a Doomboy (el personaje principal) el cual es controlable y, además, la base de la nave espacial, en la cual se desarrollará toda la historia.

Por último, a partir del estado del sprint:

INFORMACIÓN SOBRE EL SPRINT 1	
Campo	**Valor**
Fecha de comienzo	01/12/23
Fecha de fin de sprint	31/12/23

Longitud de sprint en días	21
Días de vacaciones	10
Tamaño del equipo	1
Ocupación máxima del equipo	100%
Jornada de trabajo diaria en horas	2
Horas de trabajo totales	42
Número de sprints realizados hasta la fecha	0
Puntos de historia totales en todo el desarrollo	88
Puntos de historia restantes en todo el desarrollo	88
Puntos de historia realizados en todo el desarrollo	0
Número de historias totales del sprint	**18**
Número de historias del sprint completadas	**18**

Tabla 3.2; Información sobre el sprint1

Se puede determinar el diagrama Burndown y el diagrama de velocidad del primer sprint

3.1.1.4 Diagrama Burndown

Para realizar el diagrama Burndown, es necesario determinar los puntos de historia **restantes** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 1		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	18,00	18,00
1	17,14	18,00
2	16,28	17,00
3	15,43	17,00
4	14,57	16,00
5	13,71	15,00
6	12,86	13,00
7	12,00	12,00
8	11,14	11,00
9	10,29	10,00
10	9,43	9,00
11	8,57	8,00
12	7,71	7,00
13	6,86	7,00
14	6,00	6,00
15	5,14	5,00
16	4,29	3,00

17	3,43	2,00
18	2,57	2,00
19	1,71	1,00
20	0,86	1,00
21	0,00	0,00
Resultado (SP restantes)	**0,00**	**0,00**

Tabla 3.3; Detalles del diagrama Burndown del sprint1

Con estos datos, el diagrama Burndown queda:

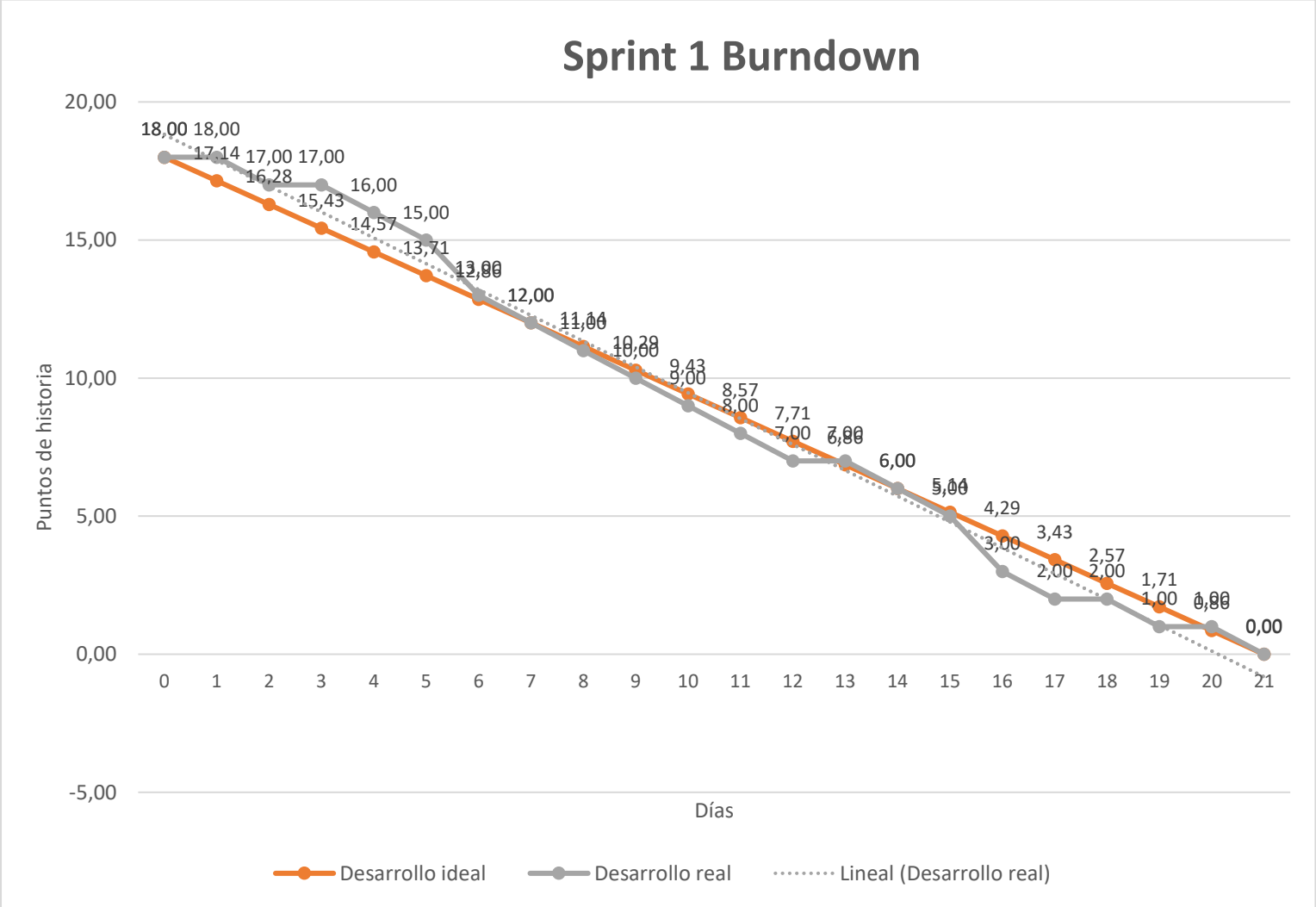


Ilustración 3.1; Diagrama Burndown del sprint1

3.1.1.5 Diagrama de velocidad

Para realizar el diagrama de velocidad, es necesario determinar los puntos de historia **realizados** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 1		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	0,00	0,00
1	0,86	0,00
2	1,71	1,00
3	2,57	1,00
4	3,43	2,00
5	4,29	3,00
6	5,14	5,00
7	6,00	6,00
8	6,86	7,00
9	7,71	8,00
10	8,57	9,00
11	9,43	10,00
12	10,29	11,00
13	11,14	11,00
14	12,00	12,00
15	12,86	13,00
16	13,71	14,00
17	14,57	16,00
18	15,43	16,00
19	16,28	17,00
20	17,14	17,00
21	18,00	18,00
Resultado (SP restantes)	**18,00**	**18,00**

Tabla 3.4; Detalles del diagrama de velocidad del sprint1

Con estos datos, el diagrama de velocidad queda:

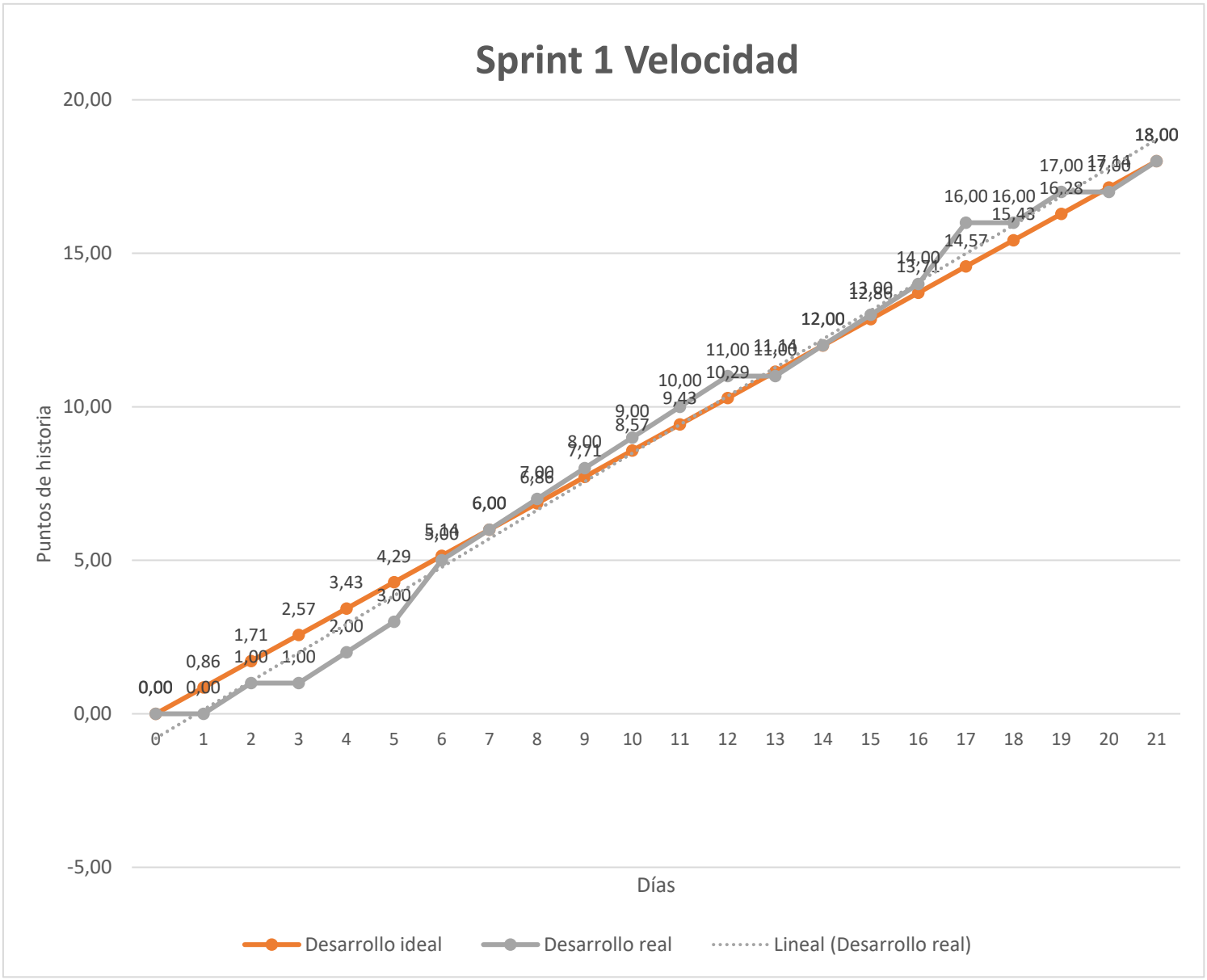


Ilustración 3.2; Diagrama de velocidad del sprint1

3.1.1.6 Comentarios finales

Gracias a los diagramas de velocidad y Burndown mostrados en los subcapítulos anteriores, se puede ver que las tareas se han completado en el tiempo establecido sin problema. Esto indica que la planificación realizada ha sido acertada y que, en situaciones normales, el autor del proyecto puede realizar la estimación de 18 puntos de historia en la duración de un sprint sin problemas.

Cabe destacar que, en este sprint, si se observan los diagramas, más o menos se han realizado la misma cantidad de puntos de historia en cada día. Esto se debe a que, en el mes de diciembre del año 2023, el autor del TFG tenía clases todos los días, por lo que la disponibilidad de trabajo durante ese mes ha sido un poco mermada por otras actividades fuera del desarrollo del proyecto.

A continuación, se van a mostrar imágenes para validar el trabajo realizado durante este sprint:

Para empezar, se colocará el Skybox de la escena:

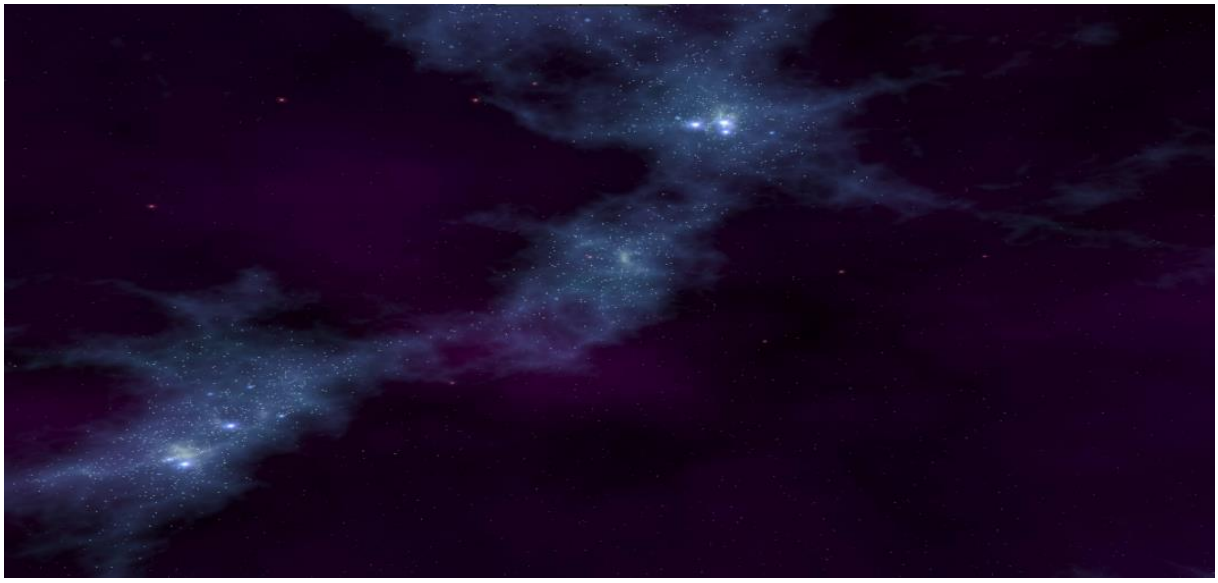


Ilustración 3.3; Skybox de la escena

Esto se ha conseguido asociando el material del Skybox deseado al componente “Skybox material” de la ventana de “Lighting” de la escena.

Luego, se tiene el nivelado:

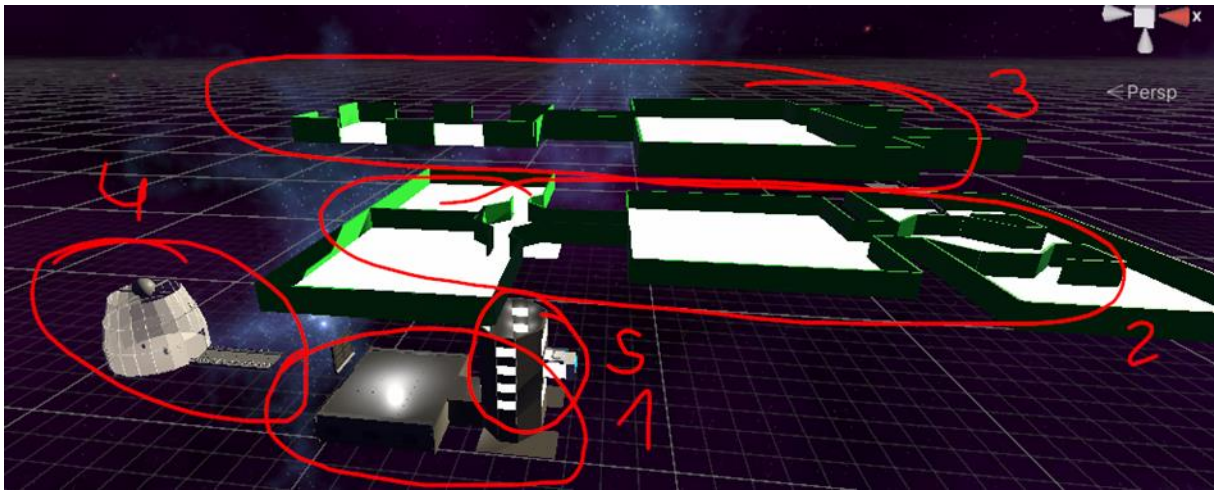


Ilustración 3.4; Plantas de la nave espacial enemiga

Donde se podrá encontrar:

1. La primera planta
2. La segunda planta
3. La tercera planta
4. La cúpula exterior
5. El ascensor

Se puede ver la implementación del personaje y de la cámara principal en esta imagen:

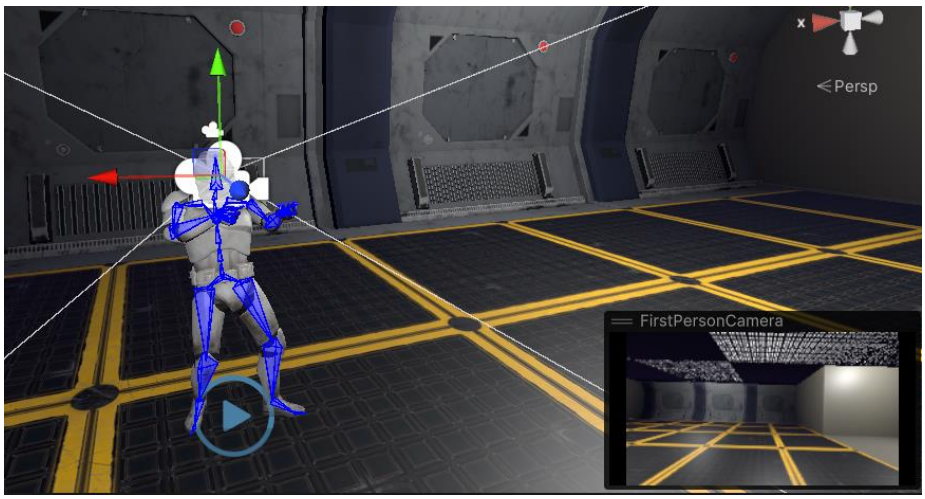


Ilustración 3.5; Personaje principal y cámara

3.1.2 Segunda iteración

3.1.2.1 Introducción

Este sprint es realizado el mes de febrero de 2024. Este mes es especial, puesto que tiene menos días. Por este motivo, a pesar de que sabemos del primer sprint que 18 puntos de historia por sprint es una cantidad aceptable para el desarrollador, se va a intentar reducir un poco el número de Story Points a realizar en este sprint.

3.1.2.2 Tareas

Como se ha comentado antes, se va a intentar reducir un poco el número de SP en este sprint por ser un mes especial. Por lo que, las tareas definidas para realizarlas en este sprint son las marcadas en amarillo en la siguiente tabla:

Tarea	Story points (SP)	Horas de trabajo (aproximadas)
Desarrollo del mapa o entorno de juego (nivelado): 1. Cúpula exterior 2. Primera planta 3. Ascensor 4. Segunda planta 5. Tercera planta	11 SP -> 13 SP: 1. 1 SP 2. 3 SP 3. 2 SP 4. 3 SP 5. 2 SP	33 -> 36 horas: 1. 3 horas 2. 9 horas 3. 6 horas 4. 9 horas 5. 6 horas
Implementación del personaje principal	1 SP	3 horas
Implementación del movimiento del personaje	2 SP	6 horas
Implementación del movimiento de la cámara principal	2 SP	6 horas
Implementación de arma del personaje	1 SP	3 horas
Implementación de funcionalidad del arma del personaje (disparar)	3 SP	9 horas
Mejora del arma y la munición	3 SP	9 horas

Diseño de las interfaces: 1. Menú principal 2. Menú de pausa 3. Menú de muerte 4. PlayerUI	8 SP: 1. 2 SP 2. 2 SP 3. 2 SP 4. 2 SP	24 horas: 1. 6 horas 2. 6 horas 3. 6 horas 4. 6 horas
Implementación de enemigos**:** 1. Enemigo con ataque cuerpo a cuerpo 2. Enemigo con ataque a distancia 3. Jefe final	10 SP -> 13 SP: 1. 3 SP 2. 3 SP 3. 4 SP	33 -> 36 horas: 1. 9 horas 2. 9 horas 3. 15 horas
Interacción con el entorno	5 SP	15 horas
Implementación de NPCs (no enemigos): 1. Kyle 2. Freddy	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Menús y sus funcionalidades	3 SP	9 horas
Implementación de objetos: 1. Modelado, texturas y materiales 2. Funcionalidad	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Interacción con objetos	5 SP	15 horas
Animación de todos los modelos de la escena	5 SP	15 horas
Decoración del mundo	3 SP	9 horas
Implementación de audio y efectos sonoros del juego	3 SP	9 horas
TOTAL:	86 Story points	252 horas

Tabla 3.5; Estimación de tareas del sprint2

Estas han sido las tareas seleccionadas y la sumatoria de sus story points son: 1 + 3 + 3 + 8 = 15 SP. Aunque no son los puntos de historia acordados en la

estimación, como se ha comentado antes, no se van a incluir 18 SP en este sprint, por ser este mes un mes especial.

3.1.2.3 Desarrollo del sprint

Las tareas en este sprint ya fueron un poco más complejas que las del sprint anterior, sobre todo, la tarea de implementación de la funcionalidad de disparar del arma y su mejora. No obstante, la implementación del arma y su funcionalidad no fue muy difícil. Probando y ajustando valores, se pudo conseguir un resultado aceptable. La tarea realmente difícil de este sprint fue la tarea de mejorar el arma. Esto fue así puesto que, para mejorar el arma y su funcionalidad de disparar, fue necesario usar tecnologías y métodos con los que el autor del proyecto no estaba acostumbrado, viendo imprescindible usar documentación de internet. Sin embargo, finalmente se consiguió un resultado bastante bueno.

En cuanto a las tareas de la implementación de la munición y de las diferentes interfaces, no fueron muy complicadas, sino tediosas.

Tras la realización de todas las tareas del sprint, se obtuvo una versión en la que el juego se iba pareciendo bastante a un juego FPS.

Por último, a partir del estado del sprint:

INFORMACIÓN SOBRE EL SPRINT 2	
Campo	**Valor**
Fecha de comienzo	01/02/24
Fecha de fin de sprint	29/02/24
Longitud de sprint en días	21
Días de vacaciones	8
Tamaño del equipo	1
Ocupación máxima del equipo	100%
Jornada de trabajo diaria en horas	2,5
Horas de trabajo totales	52,50
Número de sprints realizados hasta la fecha	1
Puntos de historia totales en todo el desarrollo	88
Puntos de historia restantes en todo el desarrollo	70
Puntos de historia realizados en todo el desarrollo	18

Número de historias totales del sprint	**15**
Número de historias del sprint completadas	**15**

Tabla 3.6; Información sobre el sprint2

Se puede determinar el diagrama Burndown y el diagrama de velocidad del segundo sprint

3.1.2.4 Diagrama Burndown

Para realizar el diagrama Burndown, es necesario determinar los puntos de historia **restantes** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 2		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	15,00	15,00
1	14,28	14,00
2	13,57	13,00
3	12,86	11,00
4	12,14	11,00
5	11,43	11,00
6	10,71	10,00
7	10,00	10,00
8	9,28	9,00
9	8,57	9,00
10	7,86	9,00
11	7,14	7,00
12	6,43	7,00
13	5,71	5,00
14	5,00	5,00
15	4,29	5,00
16	3,57	4,00
17	2,86	3,00
18	2,14	2,00
19	1,43	1,00
20	0,71	1,00
21	0,00	0,00
Resultado (SP restantes)	**0,00**	**0,00**

Tabla 3.7; Detalles del diagrama Burndown del sprint2

Con estos datos, el diagrama Burndown queda:

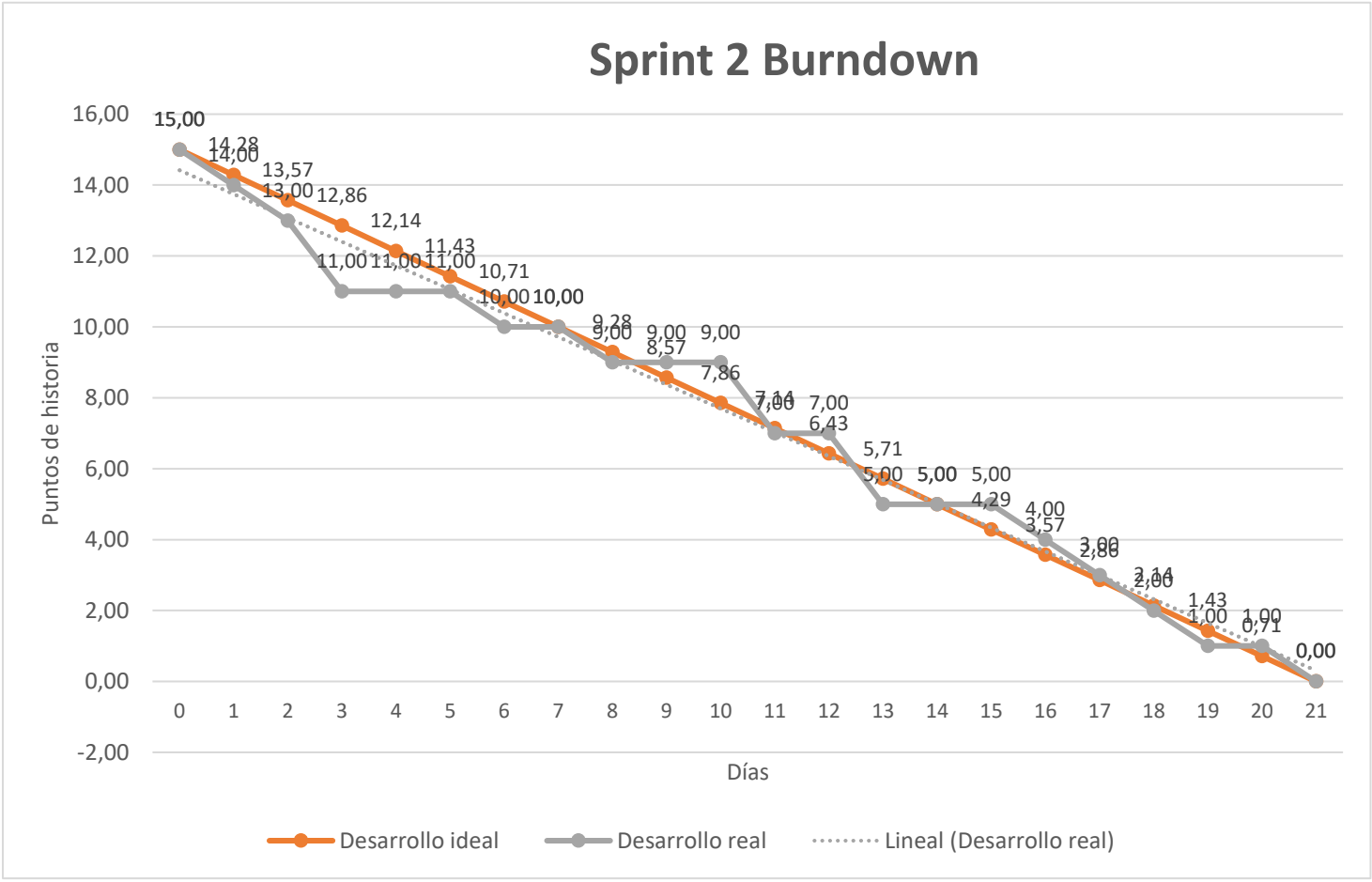


Ilustración 3.6; Diagrama Burndown del sprint2

3.1.2.5 Diagrama de velocidad

Para realizar el diagrama de velocidad, es necesario determinar los puntos de historia **realizados** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 2		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	0,00	0,00
1	0,71	1,00
2	1,43	2,00
3	2,14	4,00
4	2,86	4,00
5	3,57	4,00

6	4,29	4,00
7	5,00	5,00
8	5,71	5,00
9	6,43	6,00
10	7,14	6,00
11	7,86	6,00
12	8,57	8,00
13	9,28	8,00
14	10,00	10,00
15	10,71	10,00
16	11,43	11,00
17	12,14	12,00
18	12,86	13,00
19	13,57	14,00
20	14,28	14,00
21	15,00	15,00
Resultado (SP restantes)	**15,00**	**15,00**

Tabla 3.8; Detalles del diagrama de velocidad del sprint2

Con estos datos, el diagrama de velocidad queda:

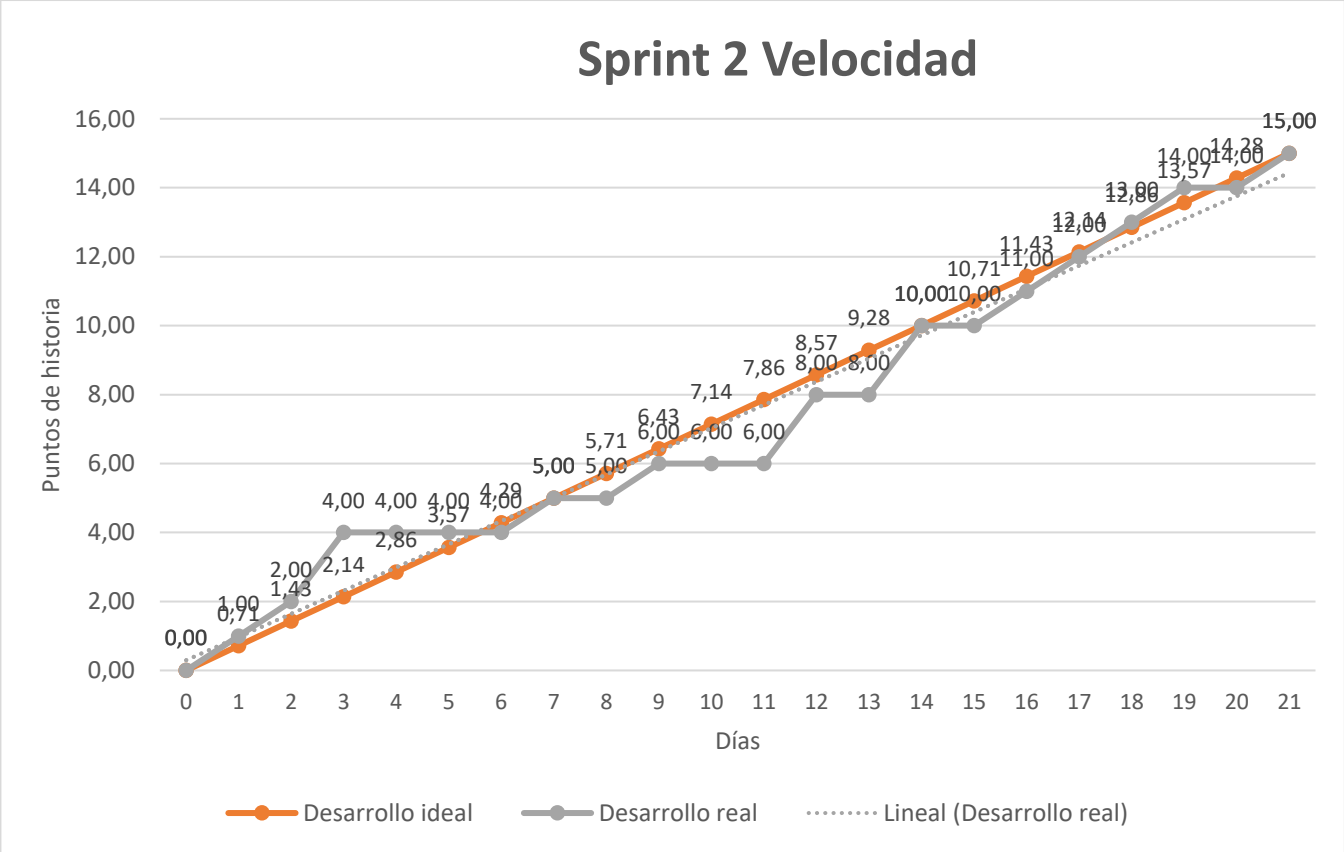


Ilustración 3.7; Diagrama de velocidad del sprint2

3.1.2.6 Comentarios finales

Gracias a los diagramas de velocidad y Burndown mostrados en los subcapítulos anteriores, se pueden ver que las tareas se han completado en el tiempo establecido sin problema, aunque hubo partes como en la mitad de los días, en los que se dificultaron algunas tareas, en concreto, la tarea de implementar las mejoras del arma. También se puede observar cómo, al principio del sprint, las primeras tareas, las cuales fueron la implementación del arma simplemente, se consiguieron realizar con bastante rapidez.

A continuación, se van a mostrar imágenes para validar el trabajo realizado durante este sprint:

Para empezar, en la siguiente imagen se puede ver la implementación del arma:

Ilustración 3.8; Modelo del arma del jugador

3.1.2.6.1 Interfaz de usuario

Seguidamente, se muestra el resultado final de la interfaz del usuario:

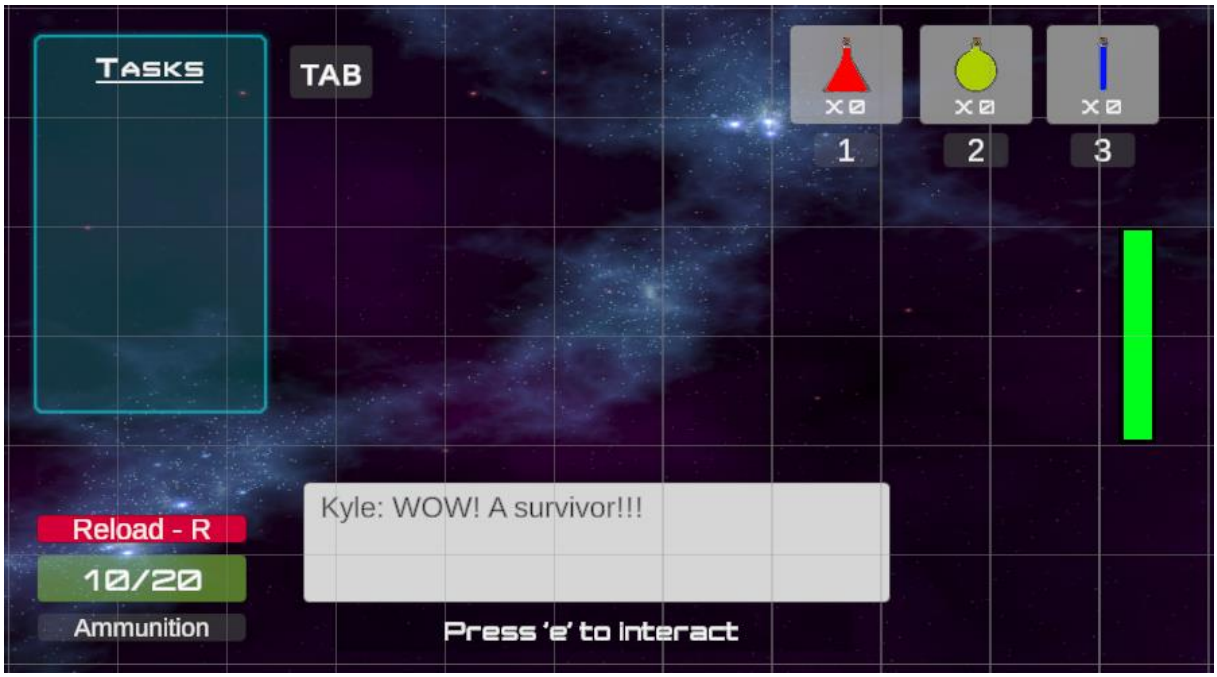


Ilustración 3.9; Interfaz del usuario, vista general

En las siguientes imágenes se detallará la interfaz del usuario. En esta se puede ver:

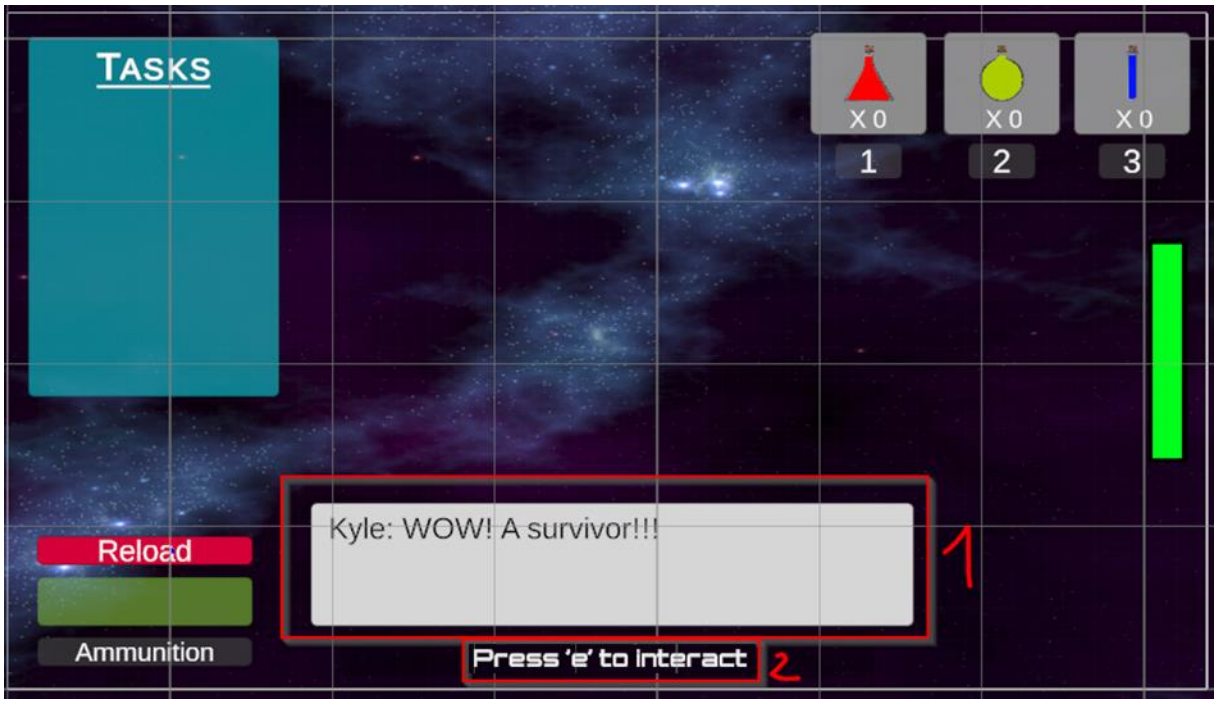


Ilustración 3.10; Interfaz del usuario, panel de dialogo y de interacción

1. Cuadro de diálogo con los NPCs
2. Panel de interacción que se mostrará cuando sea posible la interacción con algo del entorno

En la siguiente imagen:

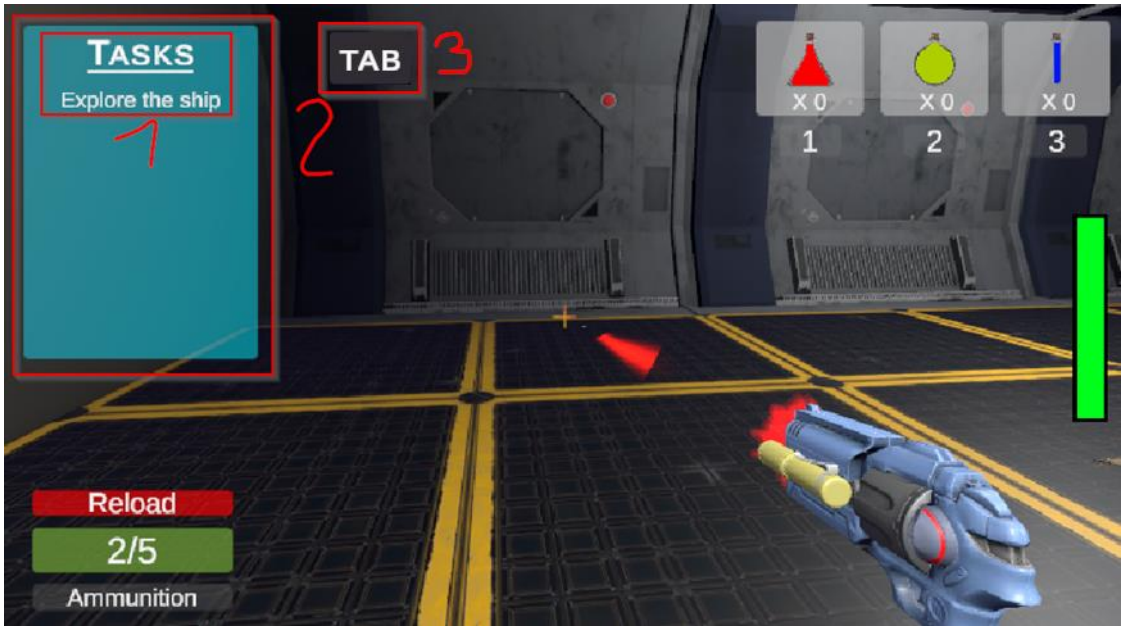


Ilustración 3.11; Interfaz del usuario, panel de tareas

Se puede ver:

1. Cuadro de tareas
2. Tareas a realizar
3. Tecla para abrir/cerrar panel de tareas

La siguiente imagen:

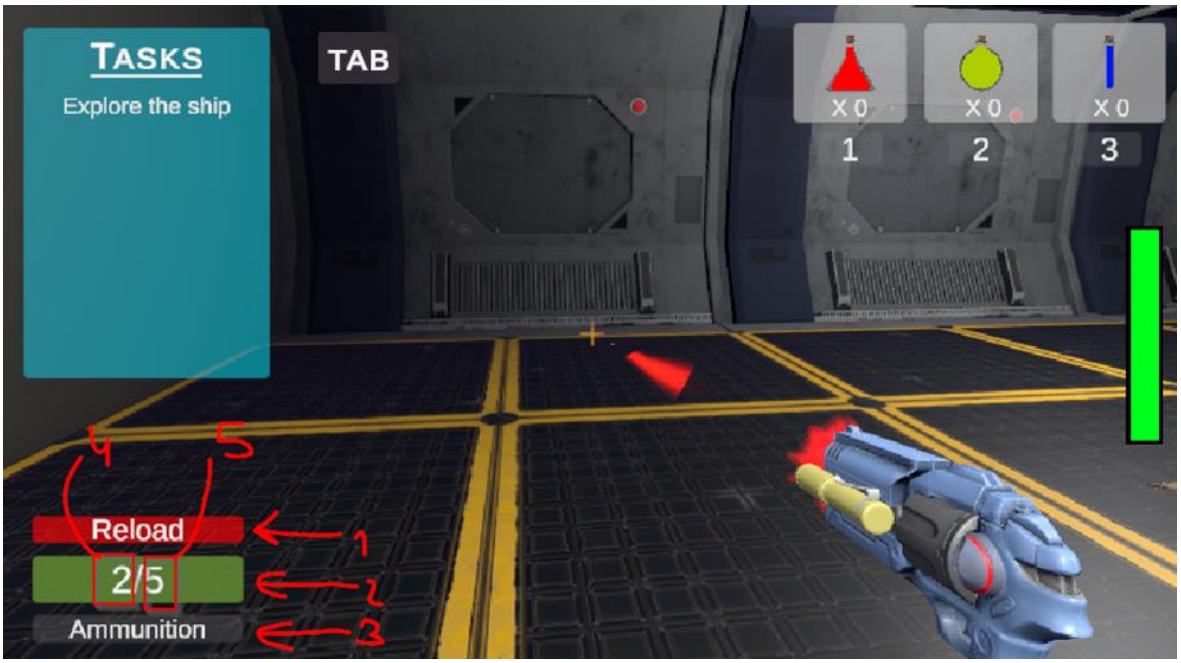


Ilustración 3.12; Interfaz del usuario, panel de munición

Muestra:

1. Mensaje de advertencia de que tenemos que recargar el arma puesto que estamos sin balas
2. Cuadro de información de la munición
3. Mensaje que muestra que el cuadro de arriba se refiere a la munición del arma
4. Munición del cargador
5. Munición “guardada” o “de reserva”

Con la siguiente imagen:

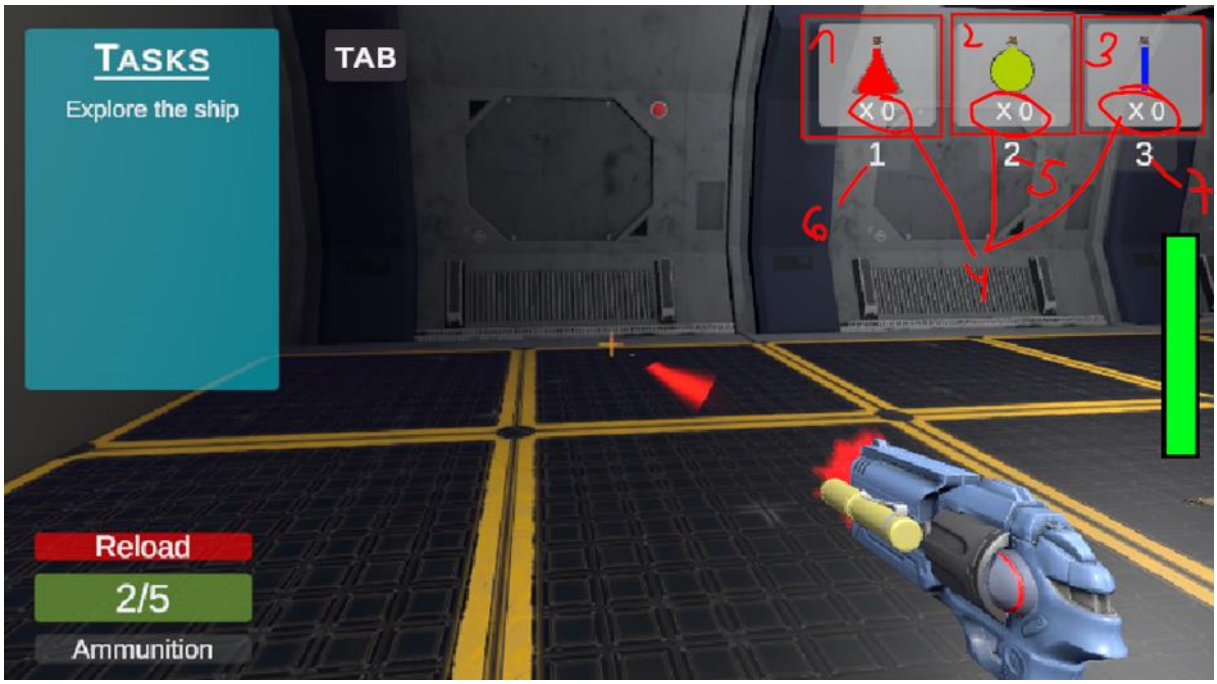


Ilustración 3.13; Interfaz del usuario, panel de pociones

Se puede ver:

1. Cuadro de poción de salud
2. Cuadro de poción de ácido
3. Cuadro de poción de velocidad
4. Indicador de la cantidad de pociones se tienen de cada tipo
5. Indicador de la tecla para seleccionar la poción de ácido
6. Indicador de la tecla para seleccionar la poción de curación
7. Indicador de la tecla para seleccionar la poción de velocidad

En la siguiente imagen:

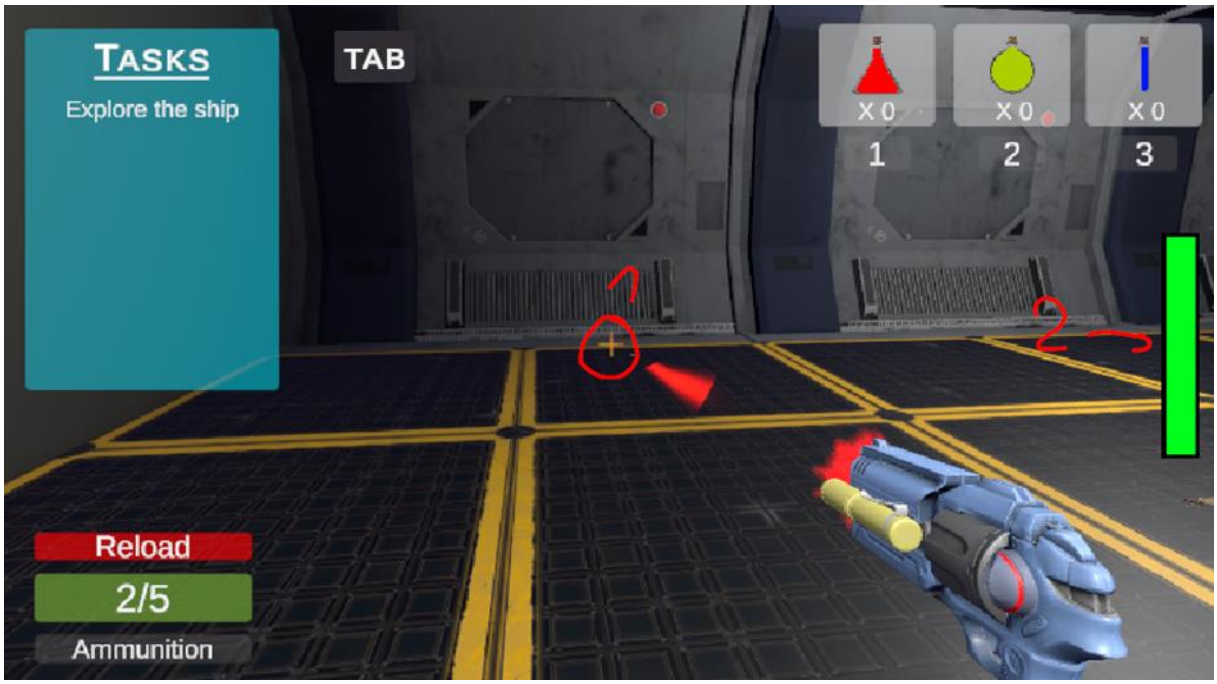


Ilustración 3.14; Interfaz del usuario, puntero y vida del jugador

Se muestra:

1. Cursor o “mirilla” que indica hacia donde está apuntando el jugador
2. Barra de vida del jugador

En la siguiente imagen:

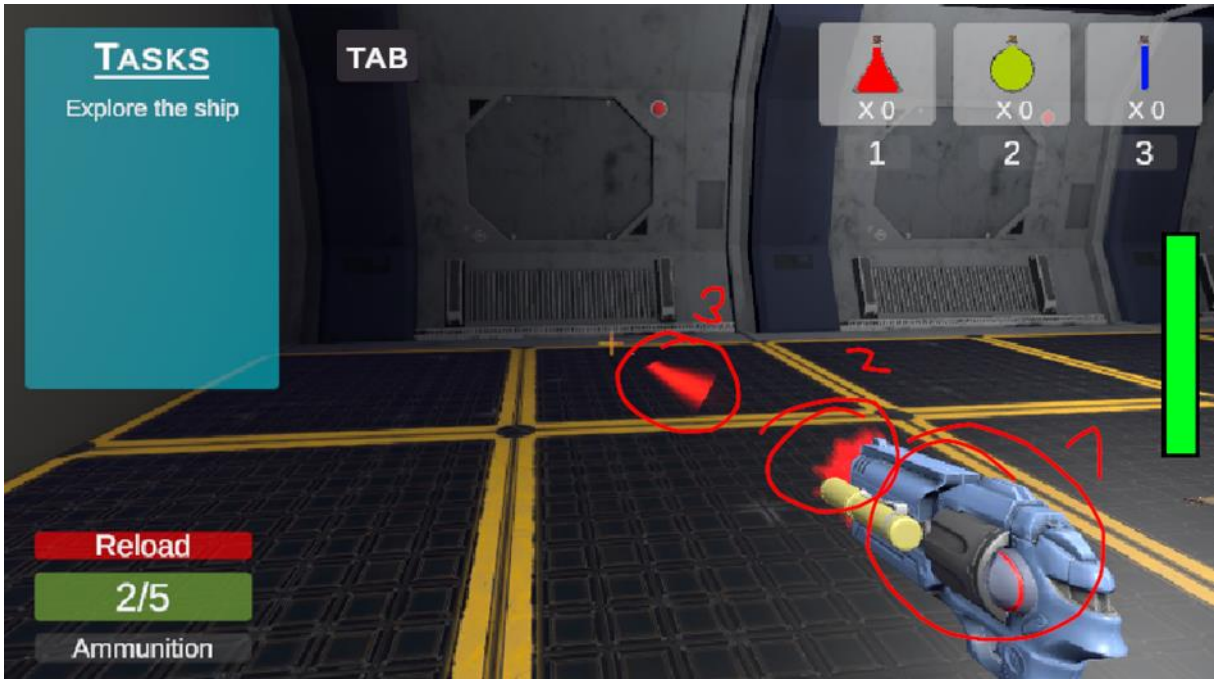


Ilustración 3.15; Interfaz del usuario, bala y arma

Se puede ver:

1. Arma del personaje
2. “Muzzle effect”: efecto de partículas en la boca del arma que indica que la bala ha salido.
3. Bala del arma.

3.1.2.6.2 Menú de inicio

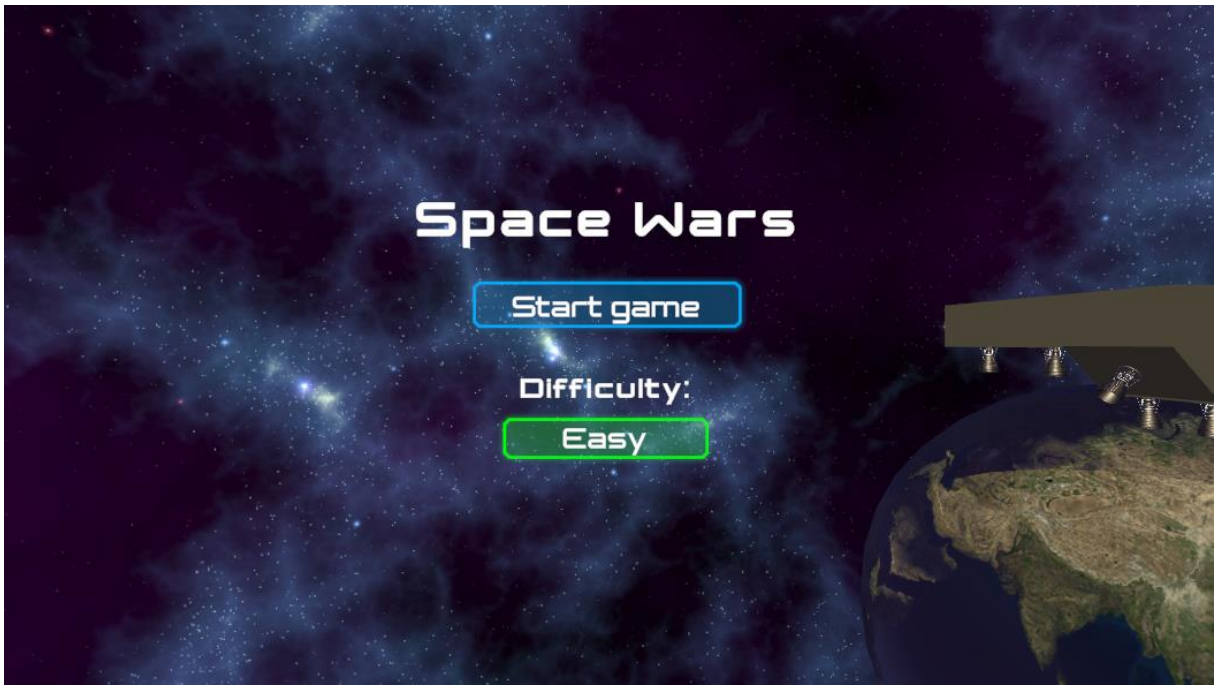


Ilustración 3.16; Menú principal

3.1.2.6.3 Menú de opciones

Ilustración 3.17; Menú de opciones

3.1.2.6.4 Pantalla de muerte

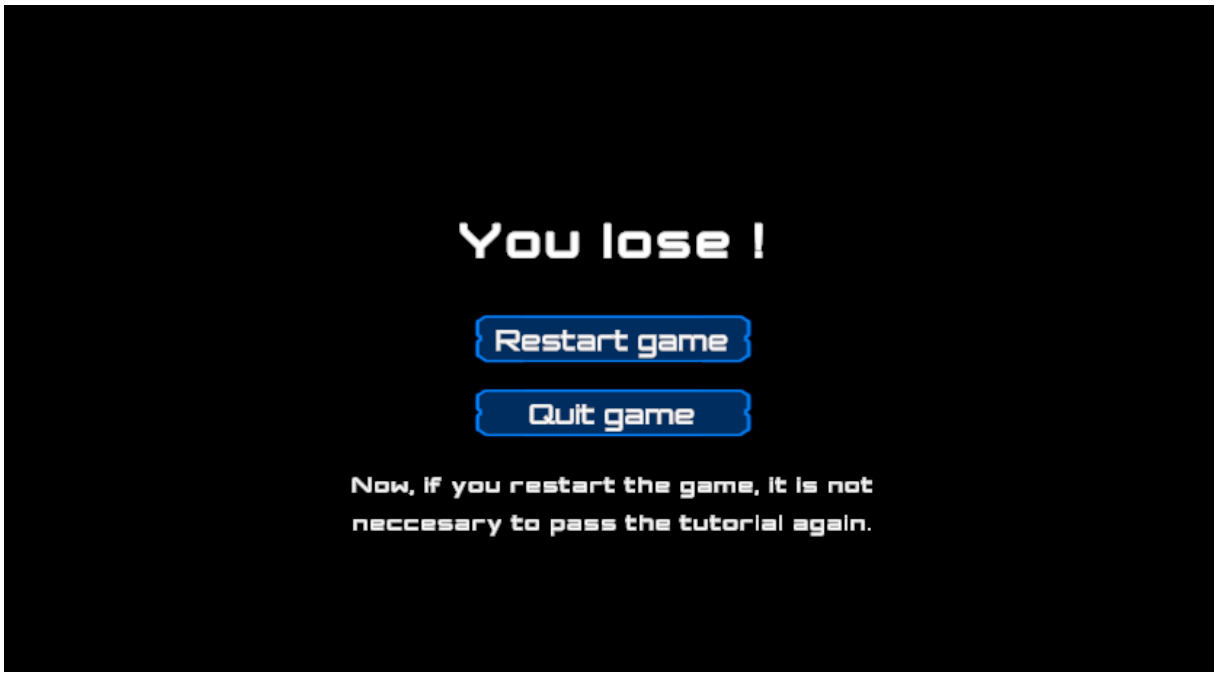


Ilustración 3.18; Pantalla de muerte

3.1.2.6.5 Munición

En las siguientes imágenes se pueden ver las distintas formas del ítem de munición:

1. Cerrada, interactuable para coger munición

Ilustración 3.19; Caja de munición, sin abrir

2. Abierta, no interactuable, la munición de su interior ha sido cogida

Ilustración 3.20; Caja de munición, abierta

3.1.3 Tercera iteración

3.1.3.1 Introducción

Del sprint anterior, se ha aprendido que las tareas pueden ser que sean más difíciles de lo que se piensa, primeramente, como la tarea del sprint anterior de mejorar el arma. No obstante, con la experiencia del primer sprint, se sabe que la planificación original para los sprints es suficiente para completar uno en un mes. Por lo que, para este tercer sprint volvemos a la planificación original. Es decir, este sprint comprenderá un esfuerzo de 18 puntos de historia.

3.1.3.2 Tareas

Habiendo determinado que este sprint comprenderá 18 puntos de historia, las tareas a realizar en este sprint son:

Tarea	Story points (SP)	Horas de trabajo (aproximadas)
Desarrollo del mapa o entorno de juego (nivelado)**:** 1. Cúpula exterior 2. Primera planta 3. Ascensor 4. Segunda planta	11 SP -> 13 SP: 1. 1 SP 2. 3 SP 3. 2 SP 4. 3 SP	33 -> 36 horas: 1. 3 horas 2. 9 horas 3. 6 horas 4. 9 horas

5. Tercera planta	5. 2 SP	5. 6 horas
Implementación del personaje principal	1 SP	3 horas
Implementación del movimiento del personaje	2 SP	6 horas
Implementación del movimiento de la cámara principal	2 SP	6 horas
Implementación de arma del personaje	1 SP	3 horas
Implementación de funcionalidad del arma del personaje (disparar)	3 SP	9 horas
Mejora del arma y la munición	3 SP	9 horas
Diseño de las interfaces: 1. Menú principal 2. Menú de pausa 3. Menú de muerte 4. PlayerUI	8 SP: 1. 2 SP 2. 2 SP 3. 2 SP 4. 2 SP	24 horas: 1. 6 horas 2. 6 horas 3. 6 horas 4. 6 horas
Implementación de enemigos**:** 1. Enemigo con ataque cuerpo a cuerpo 2. Enemigo con ataque a distancia 3. Jefe final	10 SP -> 13 SP: 1. 3 SP 2. 3 SP 3. 4 SP	33 -> 36 horas: 1. 9 horas 2. 9 horas 3. 15 horas
Interacción con el entorno	5 SP	15 horas
Implementación de NPCs (no enemigos): 1. Kyle 2. Freddy	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Menús y sus funcionalidades	3 SP	9 horas
Implementación de objetos: 1. Modelado, texturas y materiales 2. Funcionalidad	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas

Interacción con objetos	5 SP	15 horas
Animación de todos los modelos de la escena	5 SP	15 horas
Decoración del mundo	3 SP	9 horas
Implementación de audio y efectos sonoros del juego	3 SP	9 horas
TOTAL:	86 Story points	252 horas

Tabla 3.9; Estimación de tareas del sprint3

3.1.3.3 Desarrollo del sprint

Este sprint tuvo pocas tareas, pero muy tediosas y concretas. La implementación de los modelos y funcionalidades de los enemigos no fueron muy difíciles, sino tediosas. Los enemigos con ataque a distancia, una vez hecho el disparo del arma del personaje principal no fueron muy complejos, llevó más tiempo el hecho de que fueran varios enemigos con ataques a distancia. En cuanto al enemigo con ataque cuerpo a cuerpo, la mayor parte del tiempo fue a la hora de encontrar el modelo en la comunidad de assets antes que su funcionalidad. Por último, hablando del jefe final, este resultó conllevar más tiempo, como estuvo planeado. Esto fue debido a su gran nivel de detalle por su importancia dentro de la historia, y a la fusión de todos los tipos de ataques.

En cuanto a la interacción con el entorno, fue una tarea bastante compleja puesto que, primero, fue difícil adaptar la jugabilidad para que el jugador pueda detectar el objeto interactuable debido al Raycast, y segundo, fue difícil ajustar la implementación de la acción de interactuar con el objeto.

Tras la realización de todas las tareas del sprint, se obtuvo una versión en la que el juego en la que el entorno cambiaba su estado en función de las acciones tomadas y los objetos con los que se interactuaba.

Por último, a partir del estado del sprint:

INFORMACIÓN SOBRE EL SPRINT 3	
Campo	**Valor**
Fecha de comienzo	01/03/24
Fecha de fin de sprint	31/03/24
Longitud de sprint en días	21
Días de vacaciones	10
Tamaño del equipo	1
Ocupación máxima del equipo	100%
Jornada de trabajo diaria en horas	2,5
Horas de trabajo totales	52,50
Número de sprints realizados hasta la fecha	2
Puntos de historia totales en todo el desarrollo	88
Puntos de historia restantes en todo el desarrollo	55
Puntos de historia realizados en todo el desarrollo	33
Número de historias totales del sprint	**18**
Número de historias del sprint completadas	**18**

Tabla 3.10; Información sobre el sprint3

Se puede determinar el diagrama Burndown y el diagrama de velocidad del tercer sprint

3.1.3.4 Diagrama Burndown

Para realizar el diagrama Burndown, es necesario determinar los puntos de historia **restantes** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 3		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	18,00	18,00
1	17,14	17,00
2	16,28	15,00
3	15,43	15,00
4	14,57	15,00
5	13,71	14,00
6	12,86	14,00
7	12,00	13,00
8	11,14	13,00
9	10,29	13,00
10	9,43	12,00
11	8,57	11,00
12	7,71	9,00
13	6,86	9,00

14	6,00	9,00
15	5,14	7,00
16	4,29	6,00
17	3,43	4,00
18	2,57	3,00
19	1,71	2,00
20	0,86	1,00
21	0,00	0,00
Resultado (SP restantes)	**0,00**	**0,00**

Tabla 3.11; Detalles del diagrama Burndown del sprint3

Con estos datos, el diagrama Burndown queda:

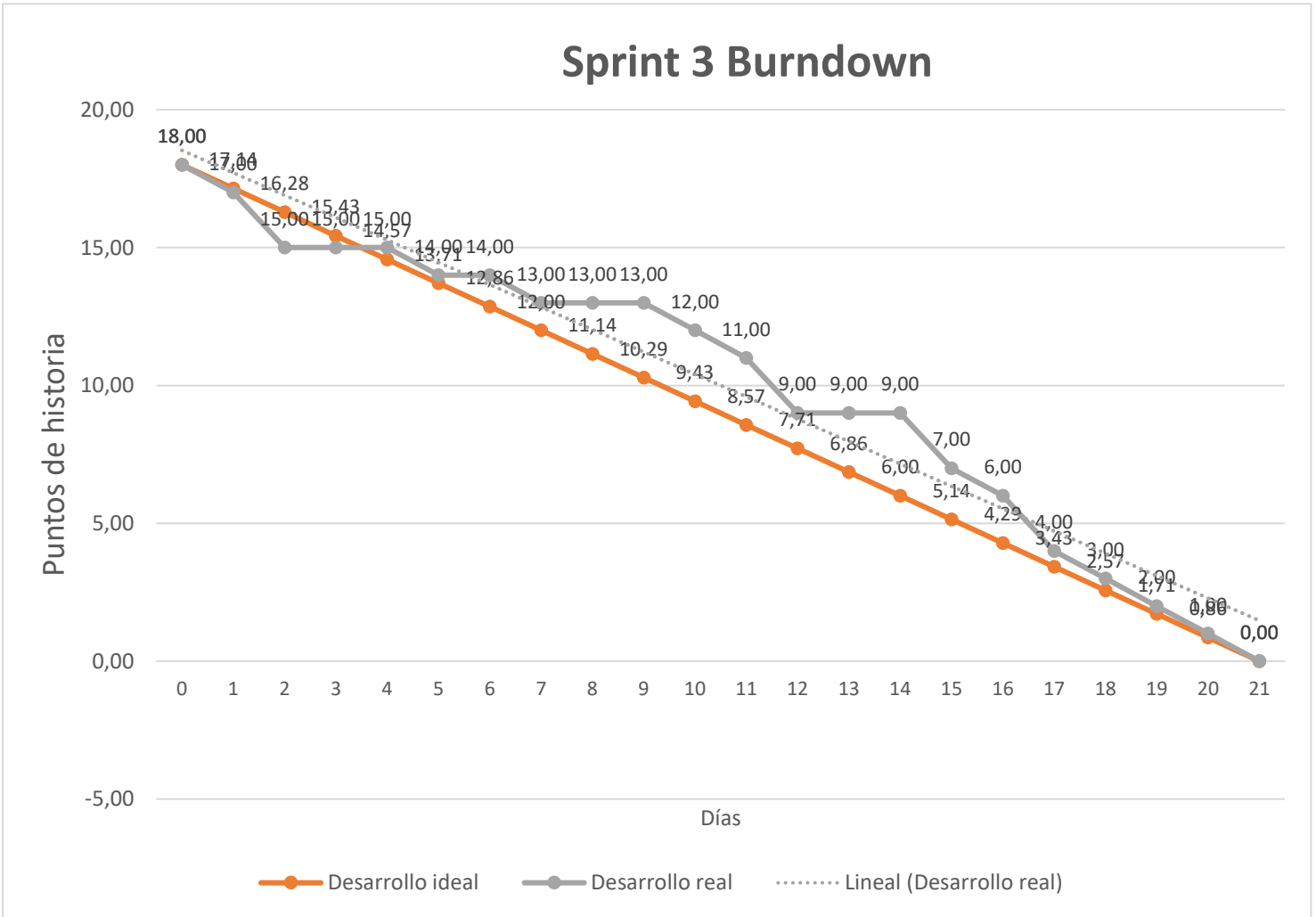


Ilustración 3.21; Diagrama Burndown del sprint3

3.1.3.5 Diagrama de velocidad

Para realizar el diagrama de velocidad, es necesario determinar los puntos de historia **realizados** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 3		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	0,00	0,00
1	0,86	1,00
2	1,71	3,00
3	2,57	3,00
4	3,43	3,00
5	4,29	4,00
6	5,14	4,00
7	6,00	5,00
8	6,86	5,00
9	7,71	5,00
10	8,57	6,00
11	9,43	7,00
12	10,29	9,00
13	11,14	9,00
14	12,00	9,00
15	12,86	11,00
16	13,71	12,00
17	14,57	14,00
18	15,43	15,00
19	16,28	16,00
20	17,14	17,00
21	18,00	18,00
Resultado (SP realizados)	**18,00**	**18,00**

Tabla 3.12; Detalles del diagrama de velocidad del sprint3

Con estos datos, el diagrama de velocidad queda:

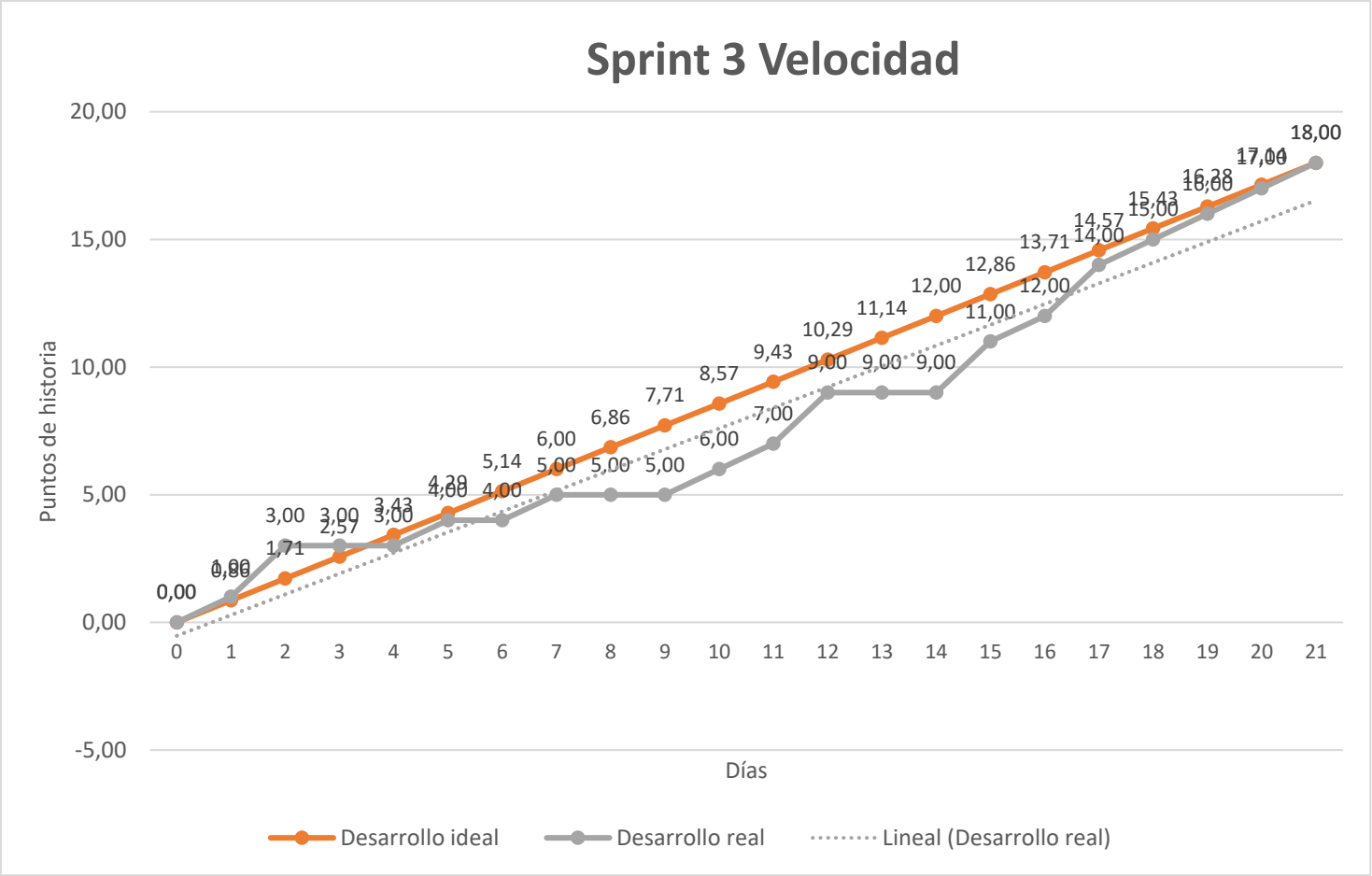


Ilustración 3.22; Diagrama de velocidad del sprint3

3.1.3.6 Comentarios finales

Gracias a los diagramas de velocidad y Burndown mostrados en los subcapítulos anteriores, se puede ver que las tareas, aunque hayan sido completadas en el tiempo establecido, han dado algunos problemas, haciendo que existan partes del sprint en las que el ritmo decayó, lo que propicia que no se alcance ritmo adecuado.

Tambien comentar que, al estar ya dentro del segundo cuatrimestre, el horario del autor ha cambiado, teniendo estas clases los días martes, miércoles y viernes de todas las semanas, días en los que ha sido difícil avanzar en el desarrollo y por ello decaer el ritmo de trabajo en estos días.

A continuación, se van a mostrar imágenes para validar el trabajo realizado durante este sprint:

3.1.3.6.1 Enemigos

3.1.3.6.1.1 Dron volador

Ilustración 3.23; Dron aéreo

3.1.3.6.1.2 Dron terrestre

Ilustración 3.24; Dron terrestre

3.1.3.6.1.3 Dron de ataque

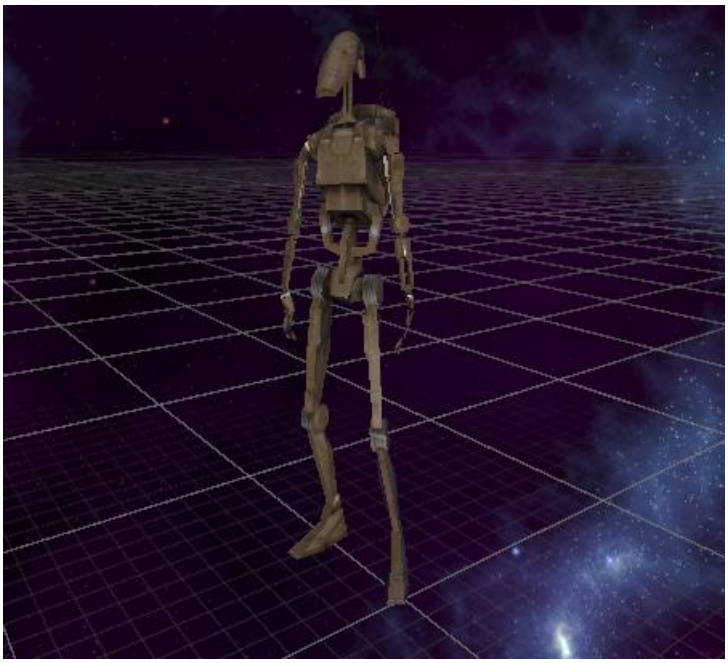


Ilustración 3.25; Dron de combate

3.1.3.6.1.4 Pilar de invocación de enemigos

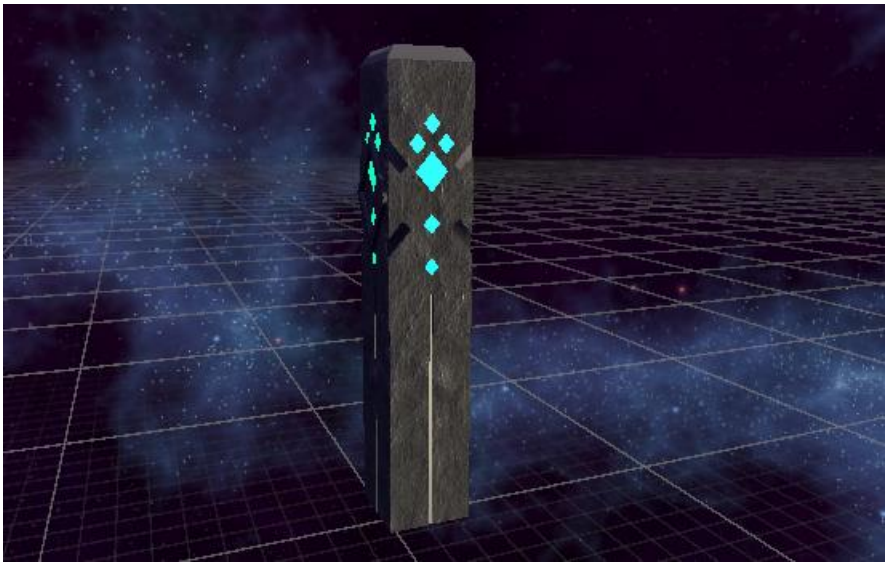


Ilustración 3.26; Pillar de invocación de enemigos

3.1.3.6.1.5 StarLord

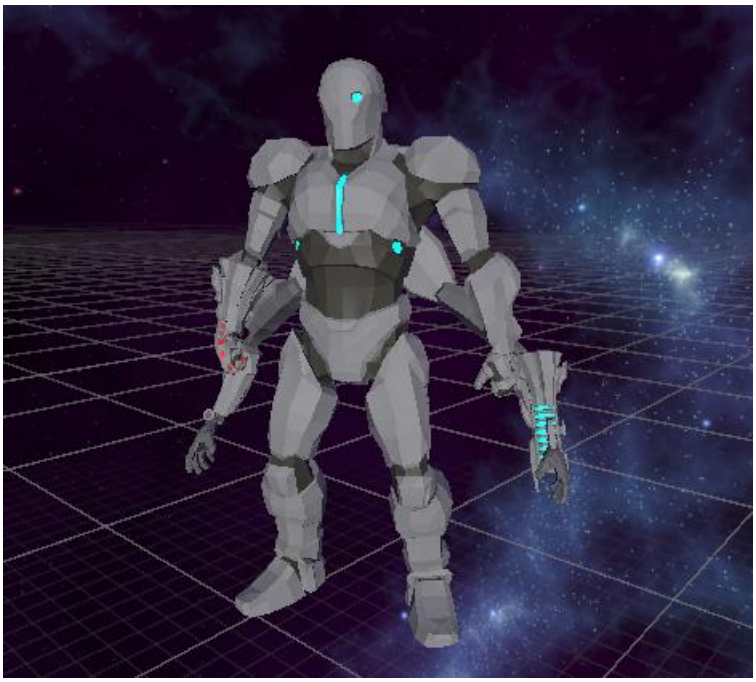


Ilustración 3.27; StarLord

3.1.3.6.2 Interacción con el entorno

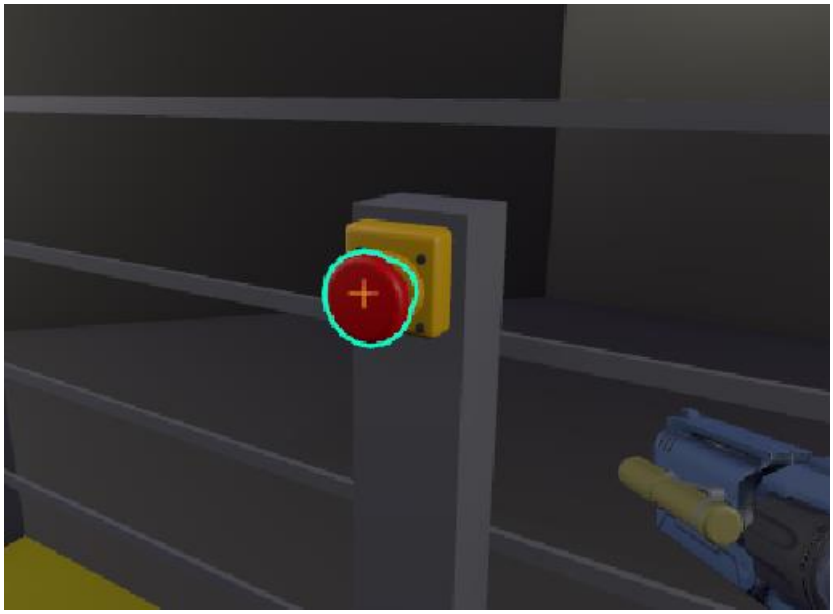


Ilustración 3.28; Botón de interacción

3.1.4 Cuarta iteración

3.1.4.1 Introducción

Este sprint es realizado en el mes de abril de 2024. Este mes tiene bastantes días de trabajo, por lo que es el sprint ideal para recuperar algunos de los puntos de historia no realizados en el segundo sprint. No obstante, teniendo de experiencia los sprints anteriores, no hay que pasarse puesto que es muy posible que las tareas resulten más difíciles de lo previamente pesando. Por estos motivos, se ha decidido que este sprint tenga 19 puntos de historia.

3.1.4.2 Tareas

Para determinar las tareas a realizar en este sprint, teniendo en cuenta que, en este sprint, a diferencia de otros, se realizarán 19 puntos de historia, las tareas a realizar en este sprint son:

Tarea	Story points (SP)	Horas de trabajo (aproximadas)
Desarrollo del mapa o entorno de juego (nivelado): 1. Cúpula exterior 2. Primera planta 3. Ascensor 4. Segunda planta 5. Tercera planta	11 SP -> 13 SP: 1. 1 SP 2. 3 SP 3. 2 SP 4. 3 SP 5. 2 SP	33 -> 36 horas: 1. 3 horas 2. 9 horas 3. 6 horas 4. 9 horas 5. 6 horas
Implementación del personaje principal	1 SP	3 horas
Implementación del movimiento del personaje	2 SP	6 horas
Implementación del movimiento de la cámara principal	2 SP	6 horas
Implementación de arma del personaje	1 SP	3 horas
Implementación de funcionalidad del arma del personaje (disparar)	3 SP	9 horas

Mejora del arma y la munición	3 SP	9 horas
Diseño de las interfaces: 1. Menú principal 2. Menú de pausa 3. Menú de muerte 4. PlayerUI	8 SP: 1. 2 SP 2. 2 SP 3. 2 SP 4. 2 SP	24 horas: 1. 6 horas 2. 6 horas 3. 6 horas 4. 6 horas
Implementación de enemigos**:** 1. Enemigo con ataque cuerpo a cuerpo 2. Enemigo con ataque a distancia 3. Jefe final	10 SP -> 13 SP: 1. 3 SP 2. 3 SP 3. 4 SP	33 -> 36 horas: 1. 9 horas 2. 9 horas 3. 15 horas
Interacción con el entorno	5 SP	15 horas
Implementación de NPCs (no enemigos): 1. Kyle 2. Freddy	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Menús y sus funcionalidades	3 SP	9 horas
Implementación de objetos: 1. Modelado, texturas y materiales 2. Funcionalidad	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Interacción con objetos	5 SP	15 horas
Animación de todos los modelos de la escena	5 SP	15 horas
Decoración del mundo	3 SP	9 horas
Implementación de audio y efectos sonoros del juego	3 SP	9 horas
TOTAL:	86 Story points	252 horas

Tabla 3.13; Estimación de tareas del sprint4

Estas son las tareas a realizar en este sprint puesto que 8 + 8 + 3 = 19 SP.

3.1.4.3 Desarrollo del sprint

Las tareas de este sprint fueron bastante bien estimadas. Empezando por la implementación de NPCs, lo más difícil fueron sus lógicas, puesto que fue necesario controlar diversas cosas y cuadrar todo para que no haya errores fuera una tarea meticulosa, en especial con la lógica de Kyle, ya que es el NPC que te ofrece el tutorial del juego. No obstante, no fue muy complejo, sino tedioso más bien.

En cuanto a la tarea de diseño de los menús y funcionalidades, a diferencia de la anterior, fue bastante sencilla, sin mucha complejidad. Hablando de la tarea de implementación de objetos, puede aparentar ser una tarea simple. No obstante, se quiso modelar y texturizar de forma muy realista y detallada las pociones y objetos, puesto que iban a estar presentes en la mayor parte del juego. Este gran realismo de los objetos fue lo que dificultó más de lo normal a la tarea. No obstante, se previó esta posible dificultad y se estimó bien la tarea.

Tras la realización de todas las tareas del sprint, se obtuvo una versión del juego que, prácticamente, podría decirse que es el juego final, aunque quedasen muchas tareas pendientes aún.

Por último, a partir del estado del sprint:

INFORMACIÓN SOBRE EL SPRINT 4	
Campo	**Valor**
Fecha de comienzo	01/04/24
Fecha de fin de sprint	30/04/24
Longitud de sprint en días	22
Días de vacaciones	8
Tamaño del equipo	1
Ocupación máxima del equipo	100%
Jornada de trabajo diaria en horas	2,5
Horas de trabajo totales	55,00
Número de sprints realizados hasta la fecha	3
Puntos de historia totales en todo el desarrollo	88
Puntos de historia restantes en todo el desarrollo	37
Puntos de historia realizados en todo el desarrollo	51
Número de historias totales del sprint	**19**
Número de historias del sprint completadas	**19**

Tabla 3.14; Información sobre el sprint4

Se puede determinar el diagrama Burndown y el diagrama de velocidad del cuarto sprint

3.1.4.4 Diagrama Burndown

Para realizar el diagrama Burndown, es necesario determinar los puntos de historia **restantes** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 4		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	19,00	19,00
1	18,14	18,00
2	17,27	17,00
3	16,41	17,00
4	15,54	15,00
5	14,68	14,00
6	13,82	13,00
7	12,95	13,00
8	12,09	13,00
9	11,23	11,00
10	10,36	11,00
11	9,50	10,00
12	8,64	10,00
13	7,77	10,00
14	6,91	8,00
15	6,05	7,00
16	5,18	6,00
17	5,18	5,00
18	3,45	4,00
19	2,59	2,00
20	1,73	1,00
21	0,86	0,00
22	0,00	0,00
Resultado (SP restantes)	**0,00**	**0,00**

Tabla 3.15; Detalles del diagrama Burndown del sprint4

Con estos datos, el diagrama Burndown queda:

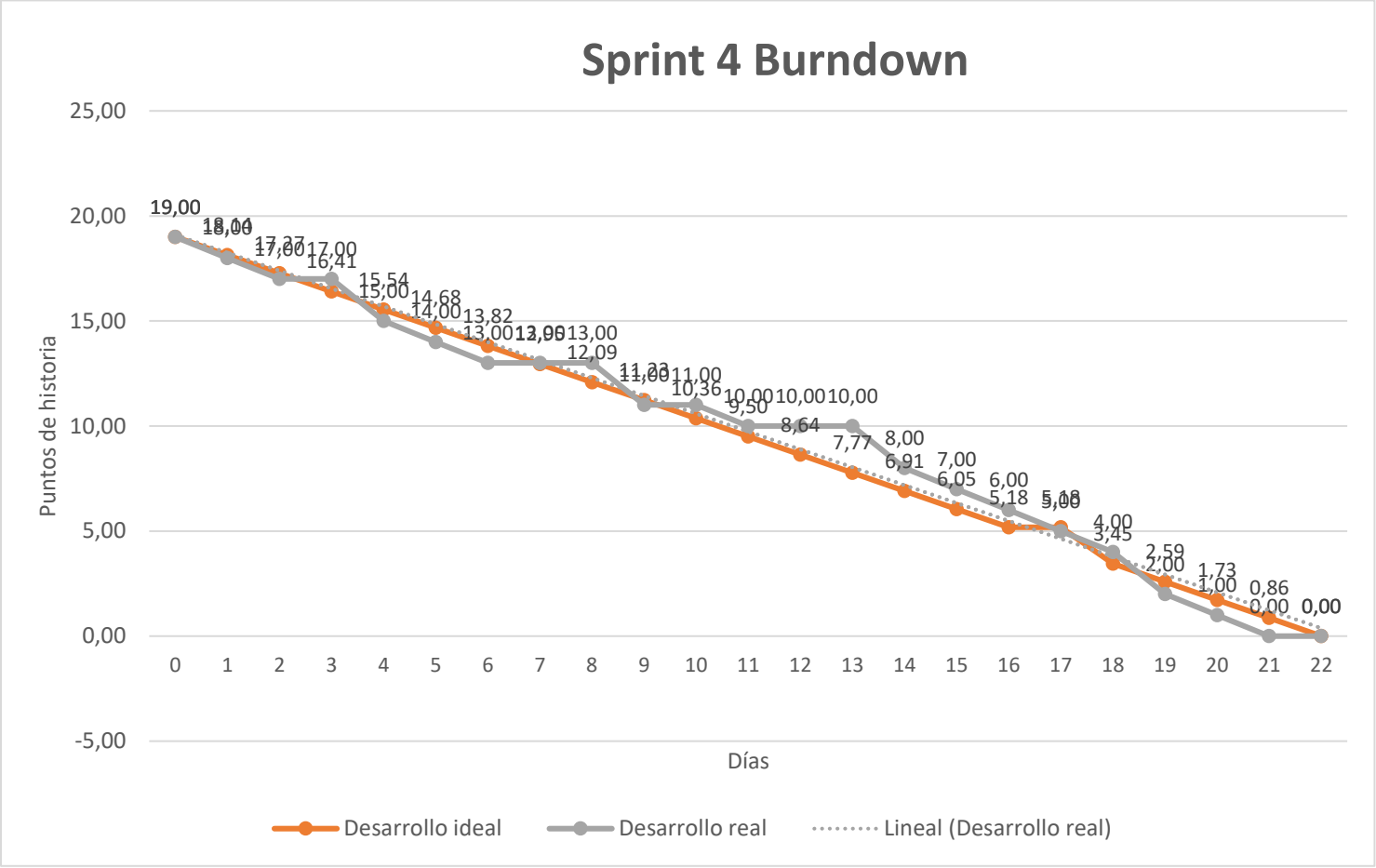


Ilustración 3.29; Diagrama Burndown del sprint4

3.1.4.5 Diagrama de velocidad

Para realizar el diagrama de velocidad, es necesario determinar los puntos de historia **realizados** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 4		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	0,00	0,00
1	0,86	1,00
2	1,73	2,00
3	2,59	2,00
4	3,45	4,00
5	4,32	5,00

6	5,18	6,00
7	6,05	6,00
8	6,91	7,00
9	7,77	7,00
10	8,64	9,00
11	9,50	9,00
12	10,36	10,00
13	11,23	10,00
14	12,09	10,00
15	12,95	12,00
16	13,82	13,00
17	14,68	14,00
18	15,54	15,00
19	16,41	17,00
20	17,27	18,00
21	18,14	19,00
22	19,00	19,00
Resultado (SP realizados)	**19,00**	**19,00**

Tabla 3.16; Detalles del diagrama de velocidad del sprint4

Con estos datos, el diagrama de velocidad queda:

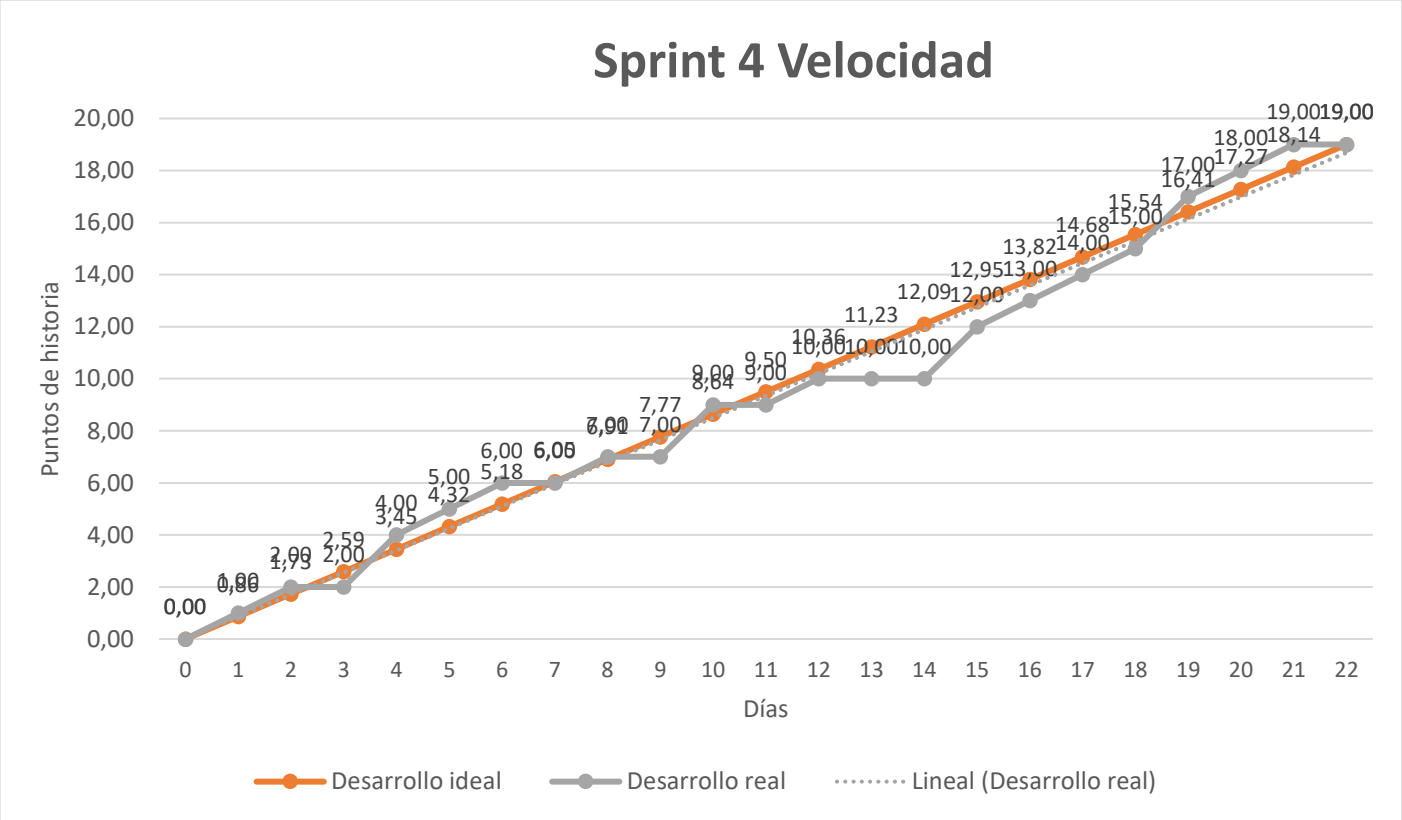


Ilustración 3.30; Diagrama de velocidad del sprint4

3.1.4.6 Comentarios finales

Atendiendo a los diagramas de velocidad y Burndown mostrados en los subcapítulos anteriores, se puede ver que al principio se aceleró el ritmo hasta igualar la curva ideal de esfuerzo. Esto fue así para remediar un posible atasco en alguna tarea en un futuro, ya que se tenía en cuenta que este sprint contaba con más story points de lo normal.

Además, se puede apreciar también que, en la parte final, a pesar de que la tarea de implementación de objetos estuvo bien estimada, se tuvieron algunos problemas con la tarea de modelado, texturas y materiales de los objetos (parte inicial del final), aunque no tanto como su funcionalidad (parte final del final).

A continuación, se van a mostrar imágenes para validar el trabajo realizado durante este sprint:

3.1.4.6.1 NPCs (no enemigos)

Primero, tenemos a Kyle, el NPC robot que ayudará al jugador enseñándole a jugar con un tutorial.

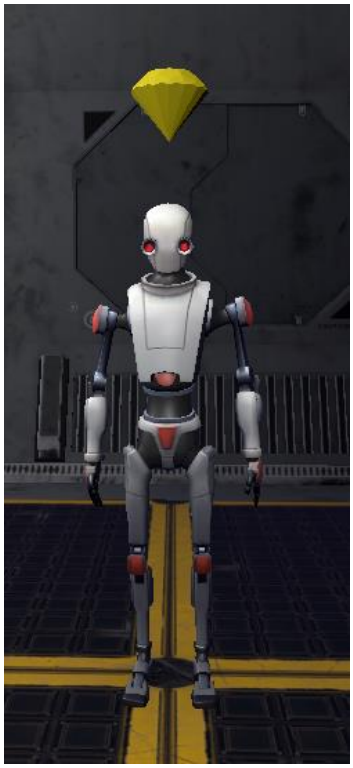


Ilustración 3.31; Kyle

Luego tendremos a Freddy, otro NPC robot que ayudará dando consejos para avanzar en la segunda planta.

Ilustración 3.32; Freddy

Por último, tendremos a Darth Vader, un NPC que se podrá encontrar encerrado en una habitación en el hueco del ascensor y con el que no se podrá interactuar, simplemente estará ahí para decorar y aportar a la historia.

Ilustración 3.33; Darth Vader

3.1.4.6.2 Objetos

3.1.4.6.2.1 Poción de curación

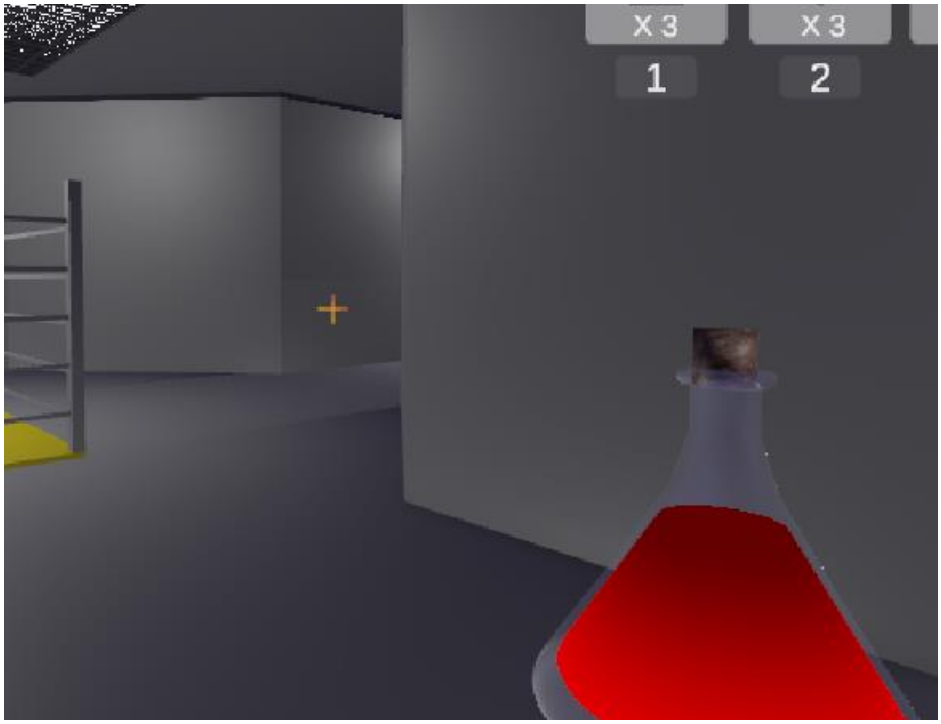


Ilustración 3.34; Poción de curación

3.1.4.6.2.2 Poción de ácido

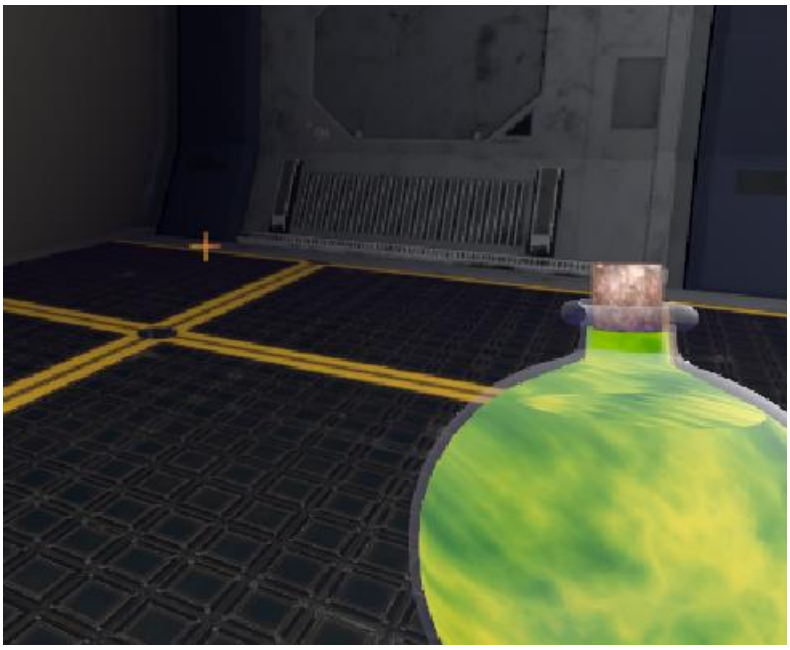


Ilustración 3.35; Poción de ácido

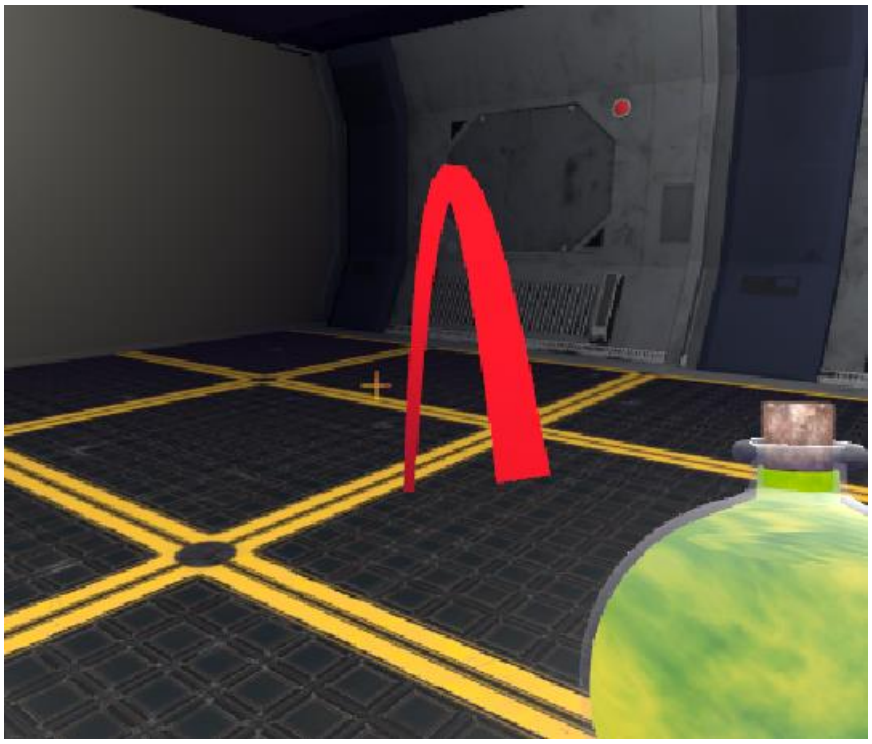


Ilustración 3.36; Poción de ácido, trayectoria de lanzamiento

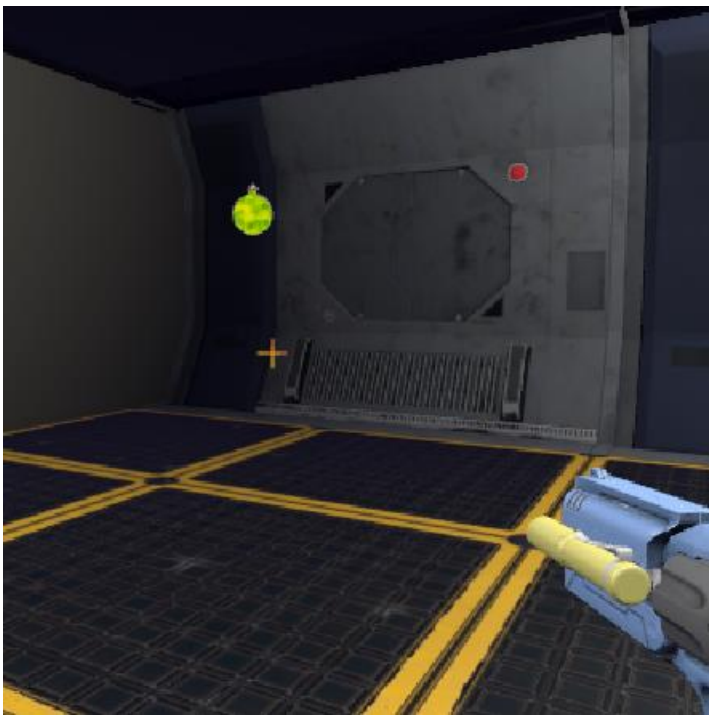


Ilustración 3.37; Poción de ácido, lanzamiento

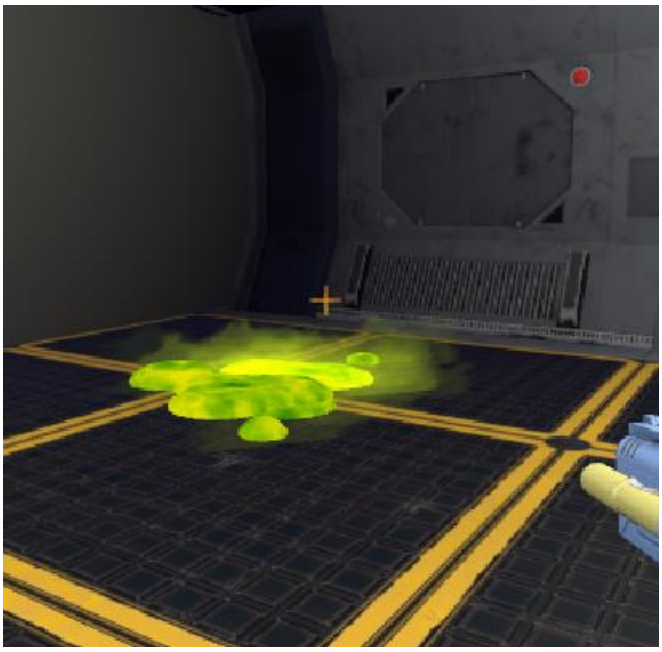


Ilustración 3.38; Poción de ácido, piscina de ácido

3.1.4.6.2.3 Poción de velocidad

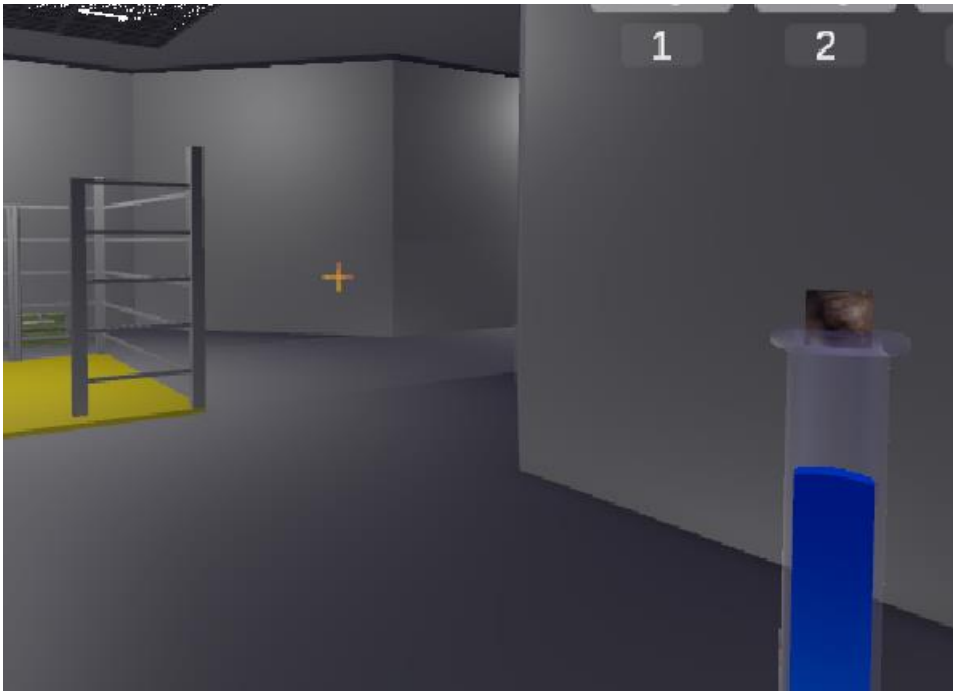


Ilustración 3.39; Poción de velocidad

3.1.4.6.2.4 Batería de la puerta de la sala de StarLord

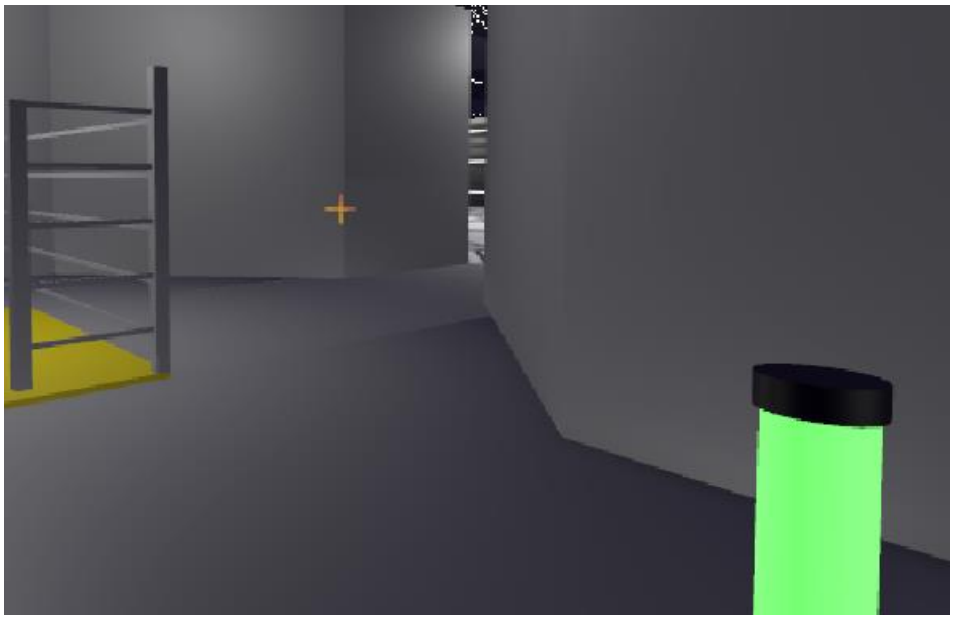


Ilustración 3.40; Batería

3.1.5 Quinta iteración

3.1.5.1 Introducción

Para este último sprint, como todas las tareas de los sprints anteriores se han completado, no queda más que realizar las tareas restantes en este sprint.

3.1.5.2 Tareas

Las tareas restantes en el backlog, y por tanto las tareas a realizar en el último sprint del desarrollo del proyecto, son:

Tarea	Story points (SP)	Horas de trabajo (aproximadas)
Desarrollo del mapa o entorno de juego (nivelado): 1. Cúpula exterior 2. Primera planta 3. Ascensor 4. Segunda planta 5. Tercera planta	11 SP -> 13 SP: 1. 1 SP 2. 3 SP 3. 2 SP 4. 3 SP 5. 2 SP	33 -> 36 horas: 1. 3 horas 2. 9 horas 3. 6 horas 4. 9 horas 5. 6 horas

Implementación del personaje principal	1 SP	3 horas
Implementación del movimiento del personaje	2 SP	6 horas
Implementación del movimiento de la cámara principal	2 SP	6 horas
Implementación de arma del personaje	1 SP	3 horas
Implementación de funcionalidad del arma del personaje (disparar)	3 SP	9 horas
Mejora del arma y la munición	3 SP	9 horas
Diseño de las interfaces: 1. Menú principal 2. Menú de pausa 3. Menú de muerte 4. PlayerUI	8 SP: 1. 2 SP 2. 2 SP 3. 2 SP 4. 2 SP	24 horas: 1. 6 horas 2. 6 horas 3. 6 horas 4. 6 horas
Implementación de enemigos**:** 1. Enemigo con ataque cuerpo a cuerpo 2. Enemigo con ataque a distancia 3. Jefe final	10 SP -> 13 SP: 1. 3 SP 2. 3 SP 3. 4 SP	33 -> 36 horas: 1. 9 horas 2. 9 horas 3. 15 horas
Interacción con el entorno	5 SP	15 horas
Implementación de NPCs (no enemigos): 1. Kyle 2. Freddy	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Menús y sus funcionalidades	3 SP	9 horas
Implementación de objetos: 1. Modelado, texturas y materiales 2. Funcionalidad	6 SP -> 8 SP: 1. 3 SP 2. 3 SP	18 -> 24 horas: 1. 9 horas 2. 9 horas
Interacción con objetos	5 SP	15 horas

Animación de todos los modelos de la escena	5 SP	15 horas
Decoración del mundo	3 SP	9 horas
Implementación de audio y efectos sonoros del juego	3 SP	9 horas
TOTAL:	86 Story points	252 horas

Tabla 3.17; Estimación de tareas del sprint5

Estas tareas suman: 5 + 5 + 3 + 3 = 16 SP

3.1.5.3 Desarrollo del sprint

Para este sprint, las tareas no fueron complejas, fueron muy meticulosas.

Primeramente, con la tarea de interactuar con los objetos, al igual que con la tarea de interactuar con el entorno, hubo que ajustar bastante el raycast para evitar errores.

En cuanto a la animación de los modelos de la escena, se invirtió más tiempo en crear y buscar las animaciones que en el controlador de animaciones del modelado en sí.

Para la decoración del mundo, al igual que la animación de modelos de la escena, se perdió más tiempo buscando modelos y efectos para decorar más que en la decoración en sí.

Hablando del audio y efectos sonoros del juego, lo que lo hizo meticuloso fue ajustar los sonidos para que no se solapen entre ellos y que suenen bien.

Tras la realización de todas las tareas del sprint, se obtuvo la versión final del juego del proyecto.

Por último, a partir del estado del sprint:

INFORMACIÓN SOBRE EL SPRINT 5	
Campo	**Valor**
Fecha de comienzo	01/05/24
Fecha de fin de sprint	31/05/24
Longitud de sprint en días	23
Días de vacaciones	8
Tamaño del equipo	1
Ocupación máxima del equipo	100%
Jornada de trabajo diaria en horas	2,5
Horas de trabajo totales	57,50
Número de sprints realizados hasta la fecha	4
Puntos de historia totales en todo el desarrollo	88
Puntos de historia restantes en todo el desarrollo	18
Puntos de historia realizados en todo el desarrollo	70
Número de historias totales del sprint	**16**
Número de historias del sprint completadas	**16**

Tabla 3.18; Información sobre el sprint5

Se puede determinar el diagrama Burndown y el diagrama de velocidad del quinto sprint

3.1.5.4 Diagrama Burndown

Para realizar el diagrama Burndown, es necesario determinar los puntos de historia **restantes** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 5		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	16,00	16,00
1	15,30	16,00
2	14,61	15,00
3	13,91	14,00
4	13,22	13,00
5	12,52	13,00
6	11,83	13,00
7	11,13	12,00
8	10,43	12,00
9	9,74	10,00
10	9,04	10,00
11	8,35	10,00

12	7,65	9,00
13	6,96	9,00
14	6,26	7,00
15	5,56	7,00
16	4,87	7,00
17	4,17	5,00
18	3,48	5,00
19	2,78	3,00
20	2,09	3,00
21	1,39	3,00
22	0,70	1,00
23	0,00	0,00
Resultado (SP restantes)	**0,00**	**0,00**

Tabla 3.19; Detalles del diagrama Burndown del sprint5

Con estos datos, el diagrama Burndown queda:

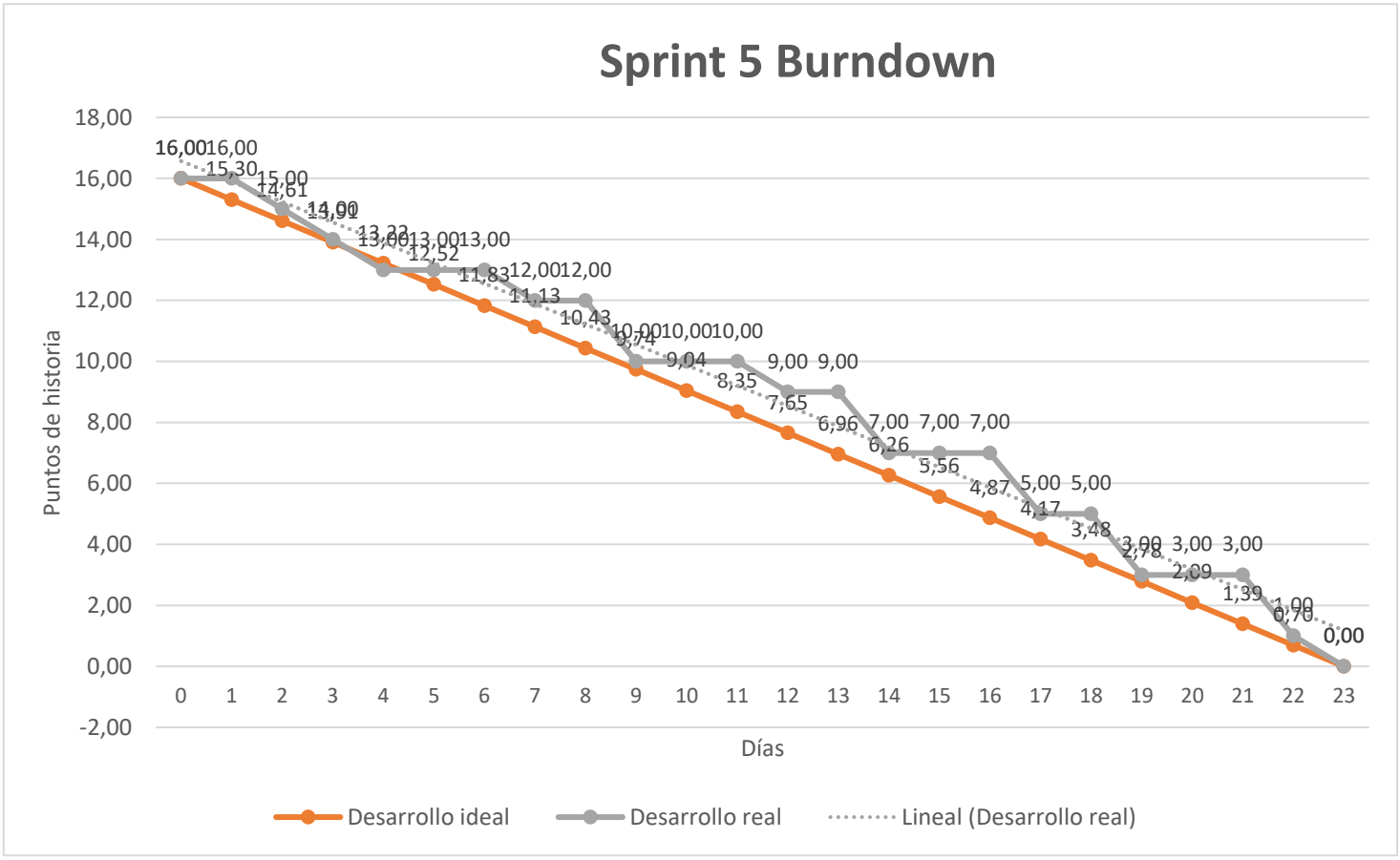


Ilustración 3.41; Diagrama Burndown del sprint5

3.1.5.5 Diagrama de velocidad

Para realizar el diagrama de velocidad, es necesario determinar los puntos de historia **realizados** en cada día del sprint. Por este motivo, se ha creado esta tabla de información del sprint:

DETALLE DEL SPRINT 5		
Día de sprint	**Desarrollo ideal**	**Desarrollo real**
0	0,00	0,00
1	0,70	0,00
2	1,39	1,00
3	2,09	2,00
4	2,78	3,00
5	3,48	3,00
6	4,17	3,00
7	4,87	4,00
8	5,56	4,00
9	6,26	6,00
10	6,96	6,00
11	7,65	6,00
12	8,35	7,00
13	9,04	7,00
14	9,74	9,00
15	10,43	9,00
16	11,13	9,00
17	11,83	11,00
18	12,52	11,00
19	13,22	13,00
20	13,91	13,00
21	14,61	13,00
22	15,30	15,00
23	16,00	16,00
Resultado (SP realizados)	**16,00**	**16,00**

Tabla 3.20; Detalles del diagrama de velocidad del sprint5

Con estos datos, el diagrama de velocidad queda:

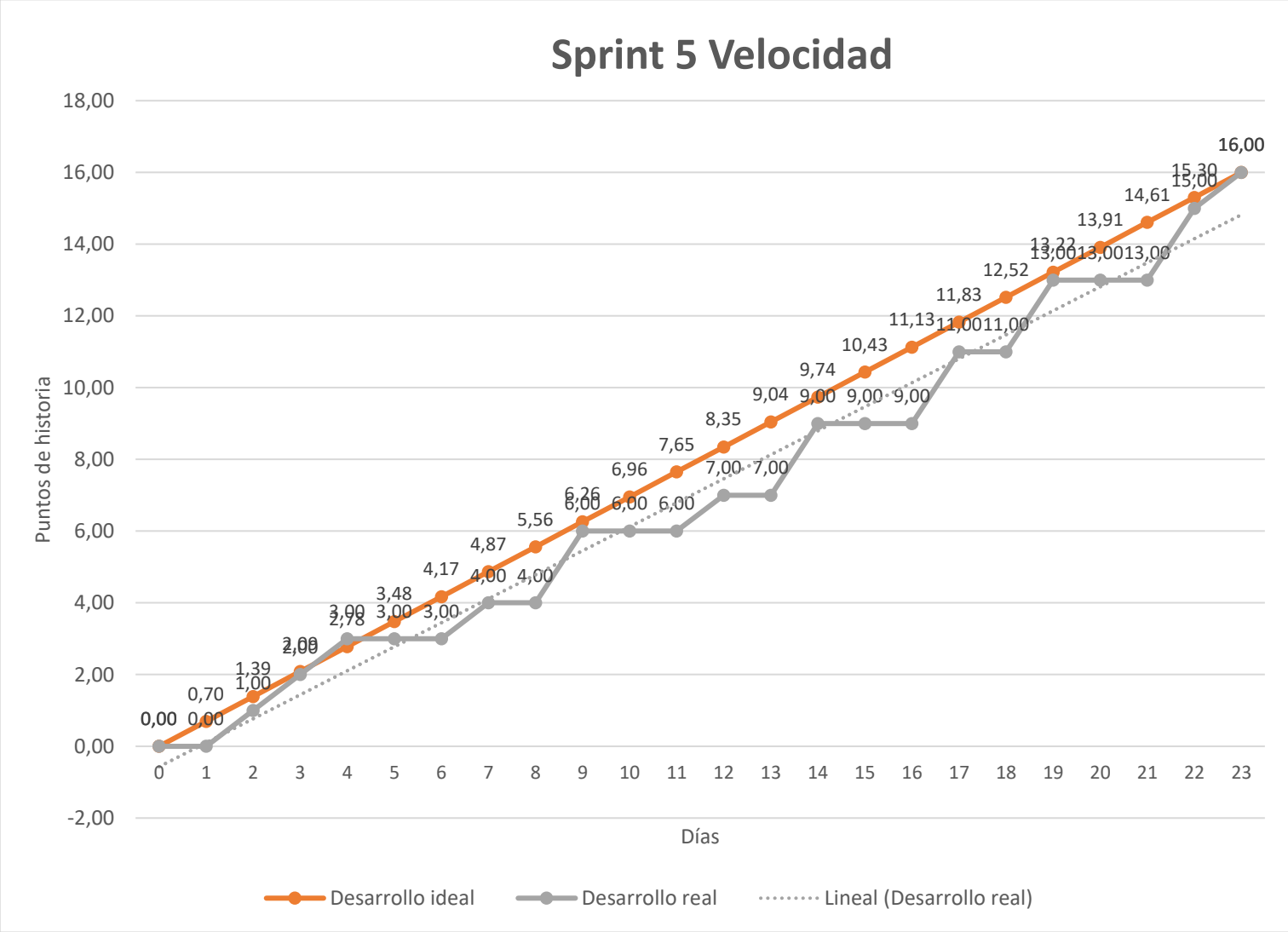


Ilustración 3.42; Diagrama de velocidad del sprint5

3.1.5.6 Comentarios finales

Para este sprint, como se puede ver en los diagramas de velocidad y Burndown mostrados en los subcapítulos anteriores, el ritmo ha disminuido con respecto a los demás sprints, aunque finalmente se han podido realizar todas las tareas. Esto es debido a que, el autor del TFG, tenía que estudiar para los exámenes de la ordinaria 2 de la universidad que tenía a finales de mayo.

A continuación, se van a mostrar imágenes para validar el trabajo realizado durante este sprint:

3.1.5.6.1 Interacción con objetos

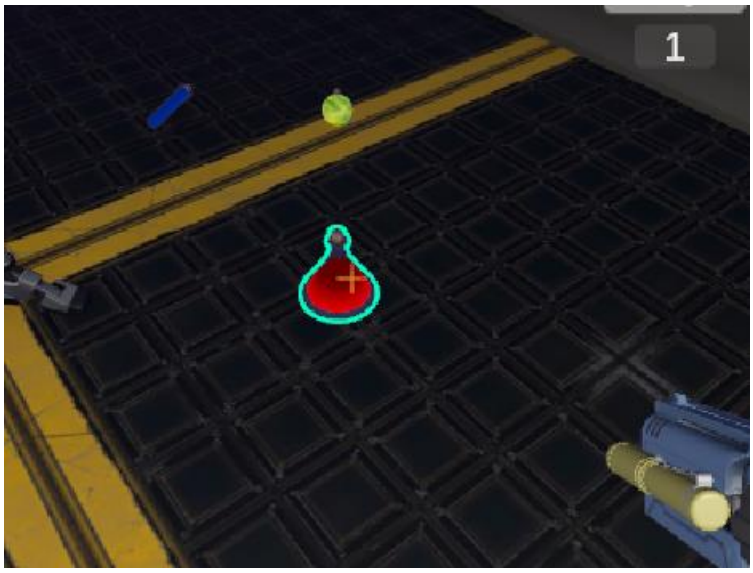


Ilustración 3.43; Interacción con poción de curación

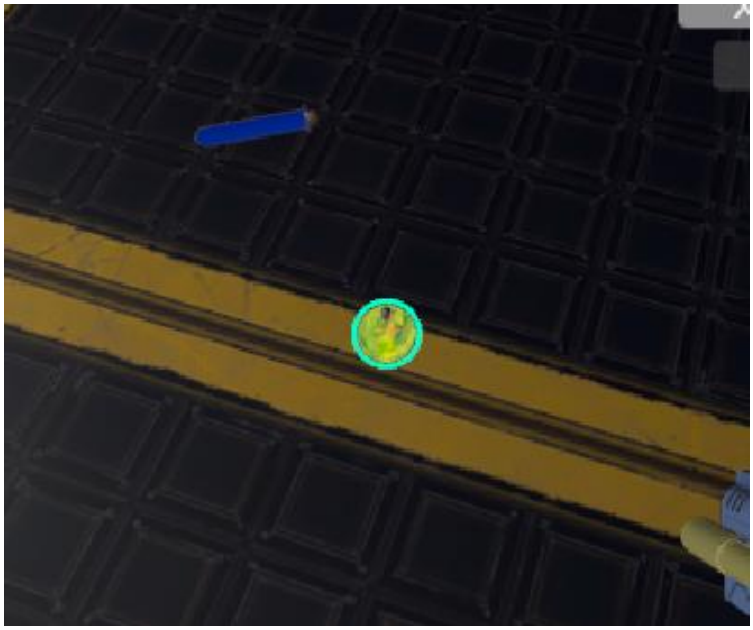


Ilustración 3.44; Interacción con poción de ácido

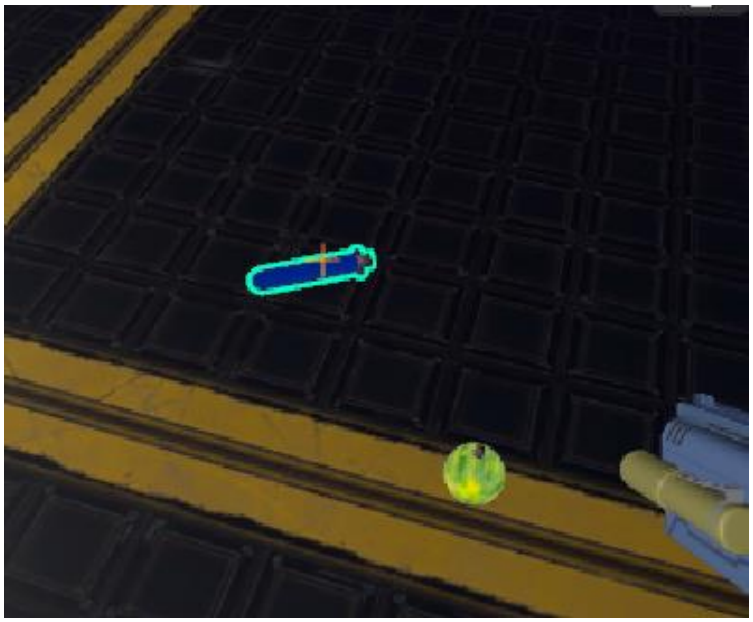


Ilustración 3.45; Interacción con poción de velocidad

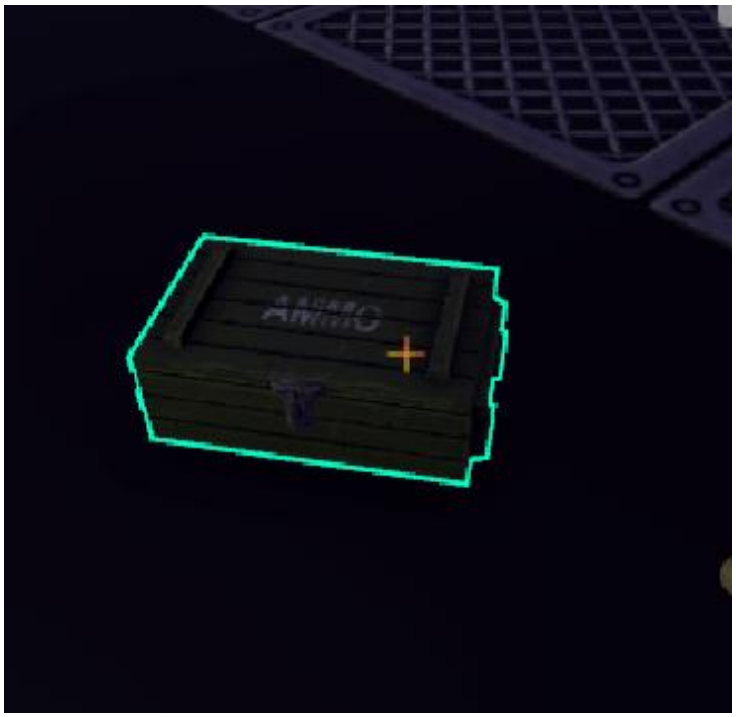


Ilustración 3.46; Interacción con caja de munición

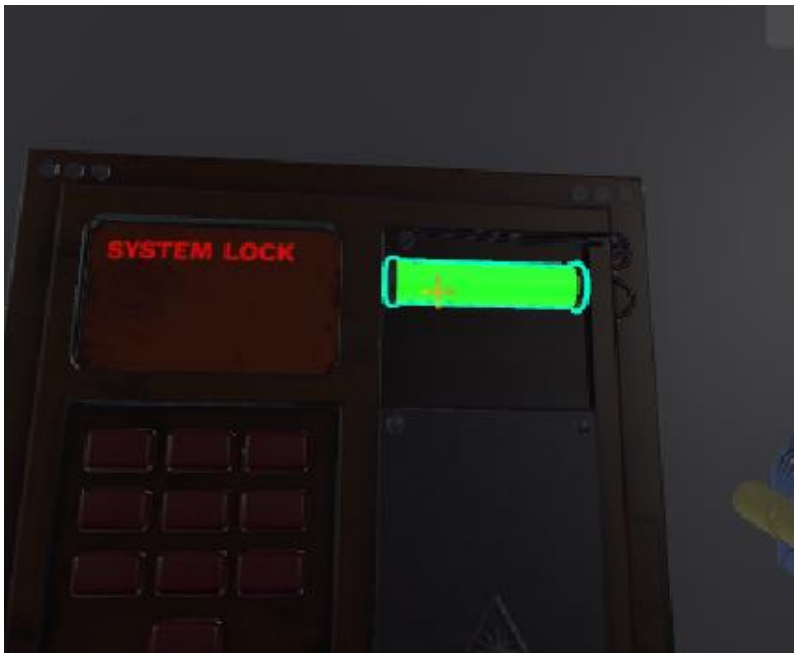


Ilustración 3.47; Interacción con batería

3.1.5.6.2 Animación de modelos

Ilustración 3.48; Animación de Darth Vader

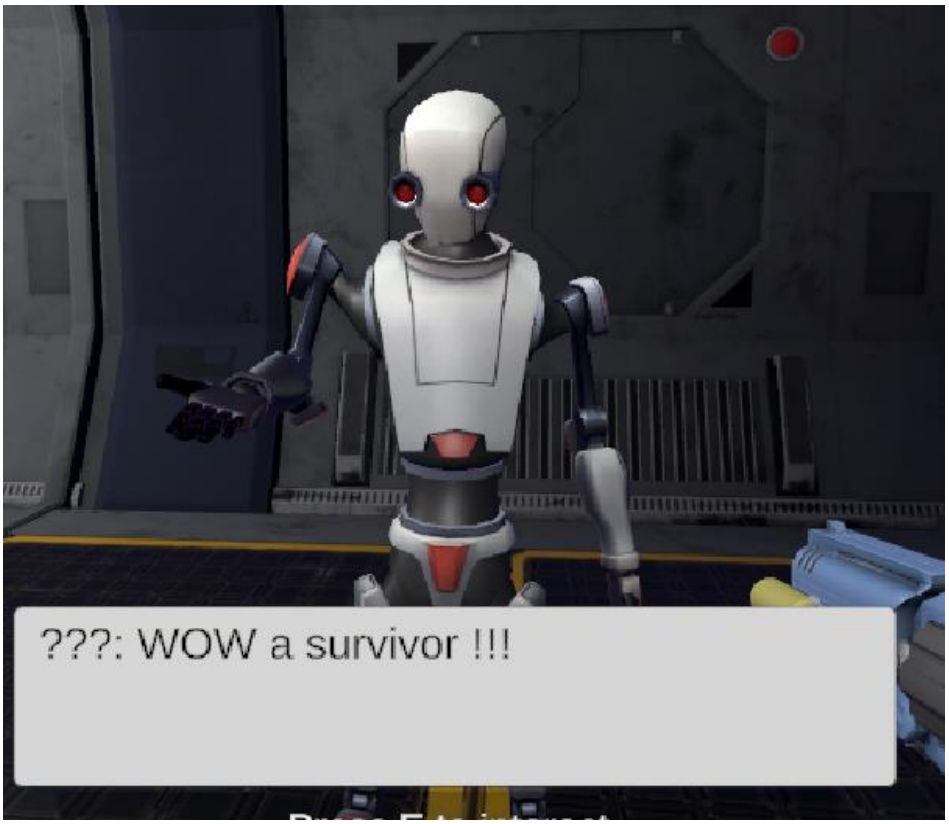


Ilustración 3.49; Dialogo con Kyle

Ilustración 3.50; Dialogo con Freddy

3.1.5.6.3 Decoración

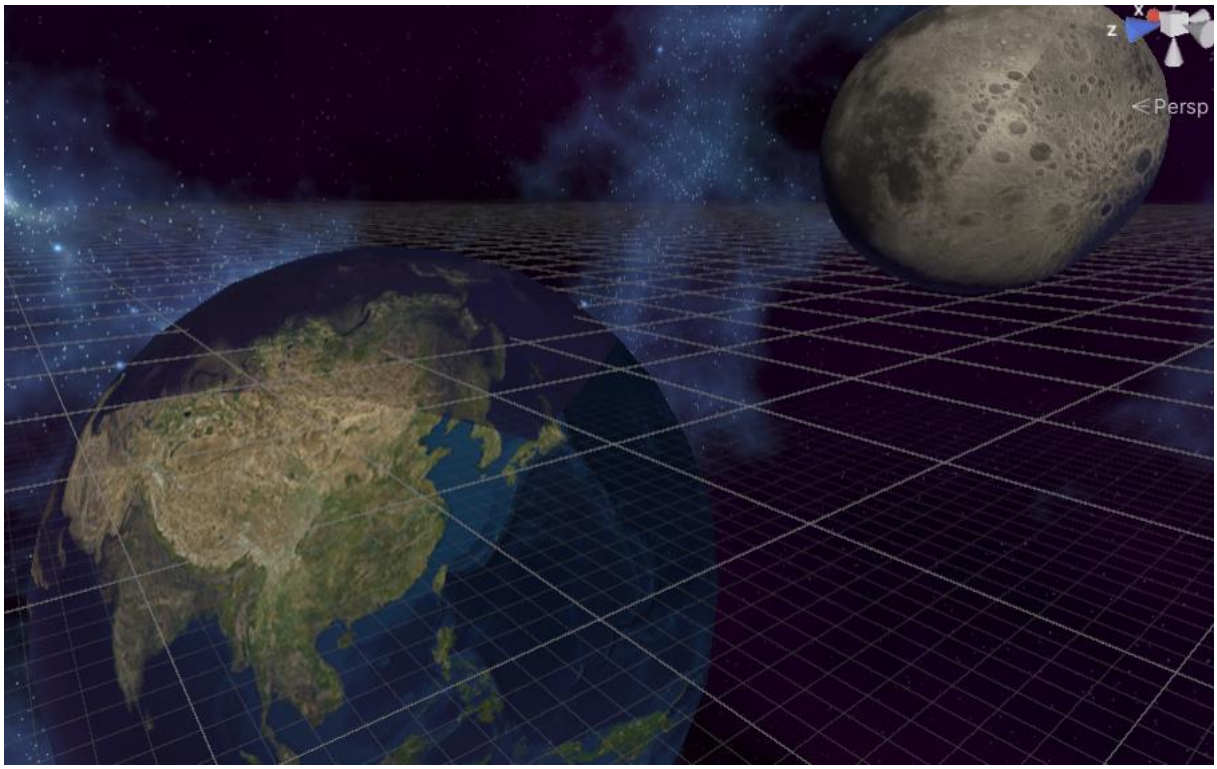


Ilustración 3.51; Tierra y Luna

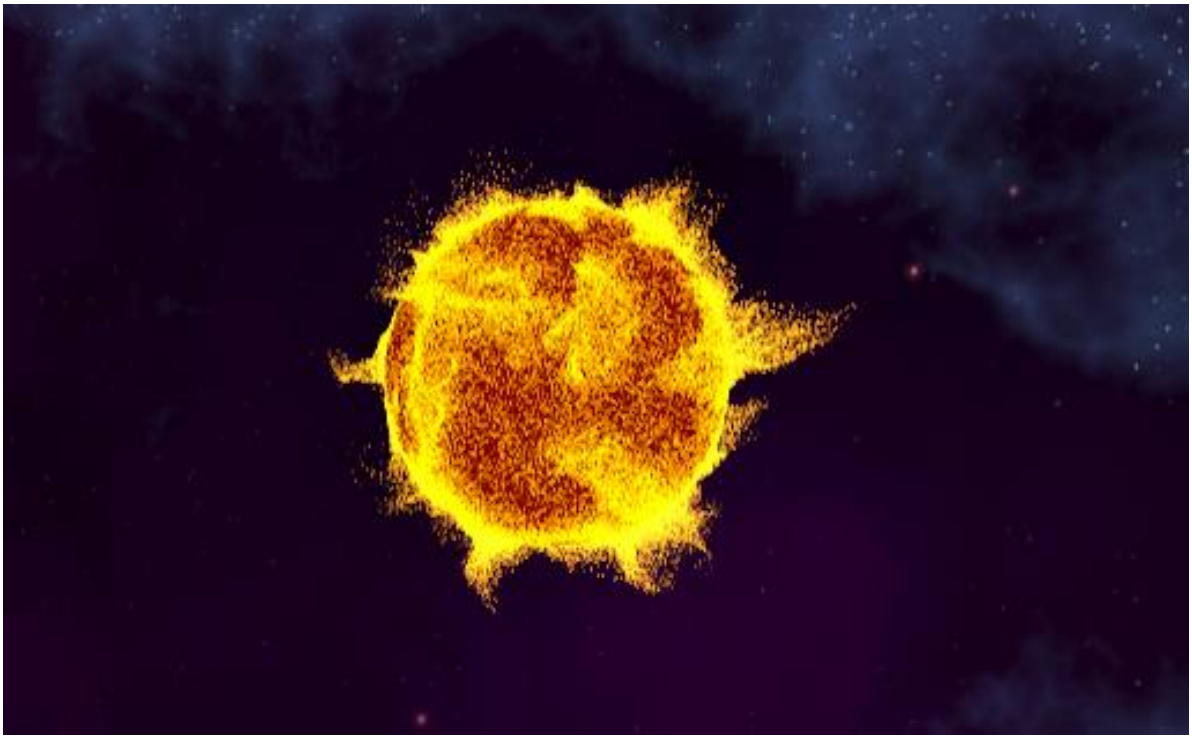


Ilustración 3.52; Sol

3.2 Funcionalidades desarrolladas

Como se ha comentado previamente, en este apartado se detallarán las funcionalidades desarrolladas en el juego del proyecto. Para ello, se harán uso de imágenes, diagramas, tablas, pseudocódigos, … con el objetivo de clarificar y explicar el razonamiento y funcionamiento del contenido del juego. [51] [52] [53].

Este apartado complementará el apartado anterior a cerca de las iteraciones del proyecto, explicando todas las funcionalidades desarrolladas en cada una de las tareas de las iteraciones.

Para cada funcionalidad, se detallarán las pruebas realizadas con una versión de la siguiente tabla:

Identificador del test o prueba *	
Nombre de la prueba o test	Nombre
Objetivo	Intención de la prueba
Resultado	Conclusión de la prueba
Comentarios	Información adicional y complementaria a la prueba

Tabla 3.21; Plantilla de la tabla de pruebas

* Identificador del test o prueba: este será un código único que facilitará la identificación del test o prueba a lo largo de la documentación. Este código estará formado por “Test”, concatenado con el número del apartado dentro de las funcionalidades (por ejemplo, para el apartado 3.2.5, el número del apartado es el 5), más la sucesión de test dentro del apartado

3.2.1 “GameManager“

El “GameManager“ es un script [54] que se encarga de centralizar la lógica y coordinar varios aspectos del juego.

Para este script, es común usar el patrón de diseño Singleton [55], el cual aporta a la aplicación una instancia de una clase, de forma que esta instancia sea accesible desde cualquier parte de la aplicación.

Este script controla:

1. Camara en primera persona del jugador y del menú principal
2. Al jugador y a su lógica de movimiento y vida.
3. Arma del jugador y su munición
4. Cambio entre armas y pociones
5. Dificultad del juego
6. Si el jugador ha completado el tutorial o no
7. Toda la información de la interfaz del jugador
8. Cantidad de cada una de las pociones que tiene el jugador.

Ahora, se van a comentar los distintos métodos del script, de los cuales algunos son aportados al resto de la aplicación para facilitar sus distintas funcionalidades.

Empezando por el método “Awake()”, el cual se ejecuta antes que incluso el método “Start()” [56]. Este método creará la instancia del “GameManager” y, además, actualizará el valor de la variable “tutorialCompleted”, la cual controla si el jugador se ha pasado el tutorial o no. Esta variable se actualizará en función de la variable “TutorialCompleted” de los “PlayerPrefs” [57] que son preferencias del jugador que se guardan entre escenas y entre ejecuciones del juego.

En el método “Start()”, primero se ocultarán las pociones y se mostrará el arma, para hacer que el jugador aparezca con el arma y no con pociones. También se desactivarán todos los paneles de la interfaz del jugador, ya que estos serán activados en el tutorial del juego a medida que el jugador vaya superando el tutorial. Esto se hace para que el jugador no se agobie con tanta información al principio del juego y

que esta información vaya apareciendo poco a poco a la par que la va a aprendiendo de la mano del tutorial.

En el método “Update()” [58], constantemente se estará controlando si la munición cargada del arma del jugador es menor que 5, momento en el cual activará el mensaje de información de recarga que le hará saber al jugador que le quedan pocas balas cargadas y que debería recargar su arma. Este mensaje tiene asociado un script llamado “ChangeColor”, el cual obtiene el color del panel del mensaje y, mediante una corrutina [59] llamada “ChangeColorLoop()”, junto con la función “Lerp” [60] para suavizar el cambio, pasa de tener transparencia a no tenerla, y viceversa. Dando la sensación de alerta o “peligro”, ya que es llamativo para el jugador que un mensaje “se apague” y “se encienda” constantemente.

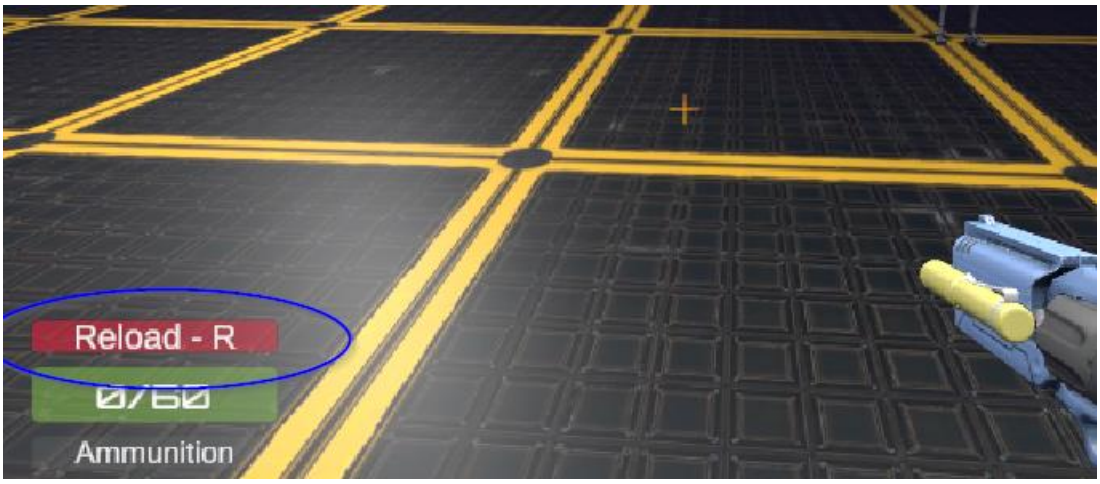


Ilustración 3.53; Mensaje de recarga de munición

Además, el método “Update()” controlará si el jugador pulsa la tecla ‘f’, momento en el cual la luz de la linterna se alternará entre apagado o encendido.

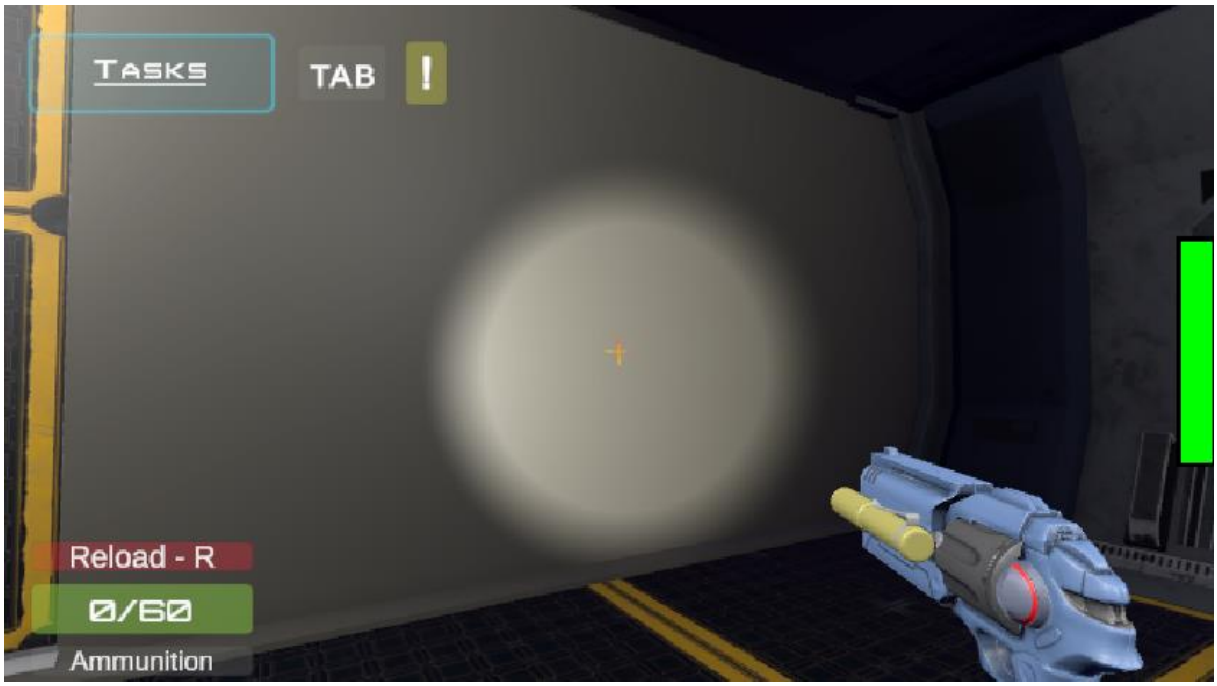


Ilustración 3.54; Luz de la lintera del arma

El script pondrá a disposición de la aplicación varios métodos. Estos son:

1. Actualización de cualquier panel de la interfaz del jugador y de su texto. En el caso del panel de tareas, si se añade un texto diferente al que previamente tenía el panel, se activará un icono de notificación amarillo al lado del panel de tareas si este está cerrado. Esto se hace para que el jugador, si tiene el panel de tareas cerrado, sepa que hay una tarea nueva disponible. En el momento en el que el jugador abra el panel de tareas, este icono desaparecerá. Tambien comentar que este icono, al igual que el mensaje de alerta por poca munición comentado antes, tiene asignado el script “ChangeColor”, por lo que este mensaje “palpitará” para llamar la atención del jugador.

Ilustración 3.55; Mensaje de nueva tarea

2. **"SetGunAmmo()":** este método recibe como parámetro una cantidad de munición cargada y otra cantidad de munición guardada. Con estos valores, el método actualiza el valor de ambas variables.
3. **"UpdateGunAmmo()":** este método, a diferencia del anterior, solo recibe una cantidad como parámetro. Esta cantidad, si es positiva actualizará la munición guardara sumándose a ella, y si es negativa, se sumará a la munición cargada, pero como es negativa, realmente restará.
4. **"Reload()":** este método se encargará de la lógica de recarga del arma. Para ello, recibe una cantidad como parámetro, la cual representa la capacidad del cargador del arma. Se usará esta cantidad para calcular la munición necesaria para recargar el arma. Por ejemplo, si el arma tiene cargadas 10 balas y el cargador tiene capacidad para 15 balas, la munición necesaria será de 15-10 = 5. Si la munición necesaria es mayor que 0, es decir, el arma no tiene cargadas tantas balas como capacidad tiene del cargador, si hay más balas guardadas que munición necesaria, se restará a la munición guardada la munición necesaria y se sumará está a la munición cargada, pero si hay menos balas guardadas que la munición necesaria, se cargará en el arma todas las balas restantes que queden en la munición guardada y esta se pondrá a 0.
5. **"UpdatePlayerLife()":** este método recibe como parámetro una cantidad que representará la cantidad de vida a actualizar del jugador. Se llamará al

método “UpdateLife()” del script “LifeBarLogic” asociado al jugador y se le pasará esta cantidad por parámetro.

6. **“TakeItem()”:** este método recibirá por parámetro un objeto el cual representa un ítem recogido por el jugador del entorno. En función del nombre del objeto recogido, se aumentará el número de ítems de ese tipo que el jugador tiene disponibles y se actualizará su cantidad en la interfaz del jugador. Además, para evitar que el jugador pueda recoger varias veces el ítem, este se destruirá.
7. **“ChangeSensibility()”:** Este método cambiará la sensibilidad del movimiento de la cámara. Para ello, recogerá el valor actual del componente “Slider” del menú, el cual puede ser un número entre el 1 y el 5, y lo multiplicará por 30 para obtener el valor de la sensibilidad. Cabe comentar que este método almacenará el valor de la sensibilidad en la variable “Sensibility” de las preferencias del jugador, de forma que, si el jugador reaparece o muere, se guardará este tipo de información.
8. “**UpdateItemShowed()”:** este método recibirá un objeto de tipo “ItemShowed”, el cual es una estructura enum que puede tener los valores “Potions”, haciendo referencia a las pociones, o “Weapons”, haciendo referencia al arma del jugador. En función del objeto de este tipo recibido, se llamará de una forma u otra a los métodos “SelectWeapon()” del script “Weapon_switching” y “SelectPotion()” del script “Potion_switching”. Si el objeto de tipo “ItemShowed” es “Weapons”, se desactivarán las pociones y se activará el arma. Si el objeto es “Potions”, se obtendrá, por medio del método “GetSelectedPotion()” del script “Potion_switching”, la poción que se quiere seleccionar (la cual será decida en función de la tecla que el jugador pulse, como se comentará más adelante). Una vez obtenida la poción que se quiere activar, si el jugador tiene cantidades de esta poción, se activará y se desactivará el arma. Si el jugador no tiene de esta poción, se desactivarán todas las pociones y se activará el arma.
9. **“UsedItem()”:** este método recibirá un objeto por parámetro que representará el ítem que el jugador ha usado. En función del nombre del objeto recogido, se disminuirá el número de ítems de ese tipo que el jugador

tiene disponibles y se actualizará su cantidad en la interfaz del jugador. Además, si al reducir la cantidad de ítems, esta cantidad se queda en cero, se llamará al método “UpdateItemShowed()” del “GameManager” pasándole por parámetro un objeto “Weapons” del enum “ItemShowed” para que se le active el arma al jugador instantáneamente.

10. **“IncreaseDifficulty()”:** este método incrementará en 1 la variable “difficultyCont”, la cual representa la dificultad del juego:

    1. Si esta es 1, la dificultad del juego será fácil
    2. Si esta es 2, la dificultad del juego será media
    3. Si esta es 3, la dificultad del juego será difícil

    Cabe destacar que se incrementará el valor de la variable utilizando el módulo de tres. Es decir, se pasará del valor 0 al 1, del 1 al 2 y del 2 al 0, sin salir de esa secuencia.

11. **“ResetPlayerSpeed()”:** este método iniciará una corrutina llamada “ResetSpeedRoutine()”, la cual, tras un tiempo determinado, devolverá a la normalidad la velocidad de movimiento del jugador. Para saber los valores normales de la velocidad del jugador, estos son recibidos como parámetros por el método y pasados a su vez por parámetros a la corrutina.
12. **“IncreaseBatteriesSetted()”:** este método aumentará el número de baterías colocadas en el panel de control de la puerta de StarLord. Para ello, si el número de baterías que tiene el jugador es distinto de cero, se reducirá esa cantidad en uno y se aumentará también en uno la cantidad de baterías colocadas en el panel de control. Después, al igual que se hacía con las pociones, si el número de baterías que tiene el jugador es cero, se llamará al método “UpdateItemShowed()” del “GameManager” pasándole por parámetro el objeto “Weapons” para desactivar las pociones y activar el arma del jugador.

13. **"GoToCredits()":** este método, simplemente lo que hará será cargar la escena [61] llamada "Credits". No obstante, antes de eso se guardará en las preferencias del jugador una variable "TutorialCompleted" que tendrá el valor 1 si el jugador ha completado el tutorial en el momento en el que se llama al método, o un 0 si el jugador no lo ha completado. Esto es útil para guardar esta información entre carga de escenas.
14. **"OnApplicationQuit()":** este método, aportado por Unity, se ejecuta cuando la aplicación se cierre. Lo que hará este método es eliminar todas las preferencias del jugador. De esta forma, si el jugador en una partida completa el tutorial, pero cierra el juego, cuando vuelva a jugarlo, deberá completar el tutorial otra vez. Lo mismo pasa con el volumen de los sonidos y de la música.

3.2.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el script "GameManager".

Test_01_01	
Nombre de la prueba o test	Aparición del mensaje de recarga
Objetivo	Debe aparecer en la interfaz del jugador un mensaje que le indique al jugador que tiene que recargar su arma.
Resultado	Cuando la munición cargada del arma del jugador es de 5 balas o menos, aparece un mensaje de recarga en la interfaz del jugador
Comentarios	Todo OK.

Tabla 3.22; Test_01_01: apertura de cápsula

Test_01_02	
Nombre de la prueba o test	Aparición del mensaje de nueva tarea
Objetivo	Debe aparecer en la interfaz del jugador un mensaje que le indique al jugador que tiene una nueva tarea que realizar

Resultado	Cuando se le asigne una nueva tarea al jugador, si este tiene el panel de tareas cerrado, aparecerá un mensaje que le indica al jugador que tiene una nueva tarea que tiene que realizar
Comentarios	Como el mensaje es simplemente un “!”, el jugador puede no saber qué significa ese símbolo.

Tabla 3.23; Test_01_02: aparición del mensaje de nueva tarea

Test_01_03	
Nombre de la prueba o test	Parpadeo del color de los mensajes
Objetivo	Los mensajes de alerta de la interfaz del jugador deben parpadear
Resultado	Cuando aparece algún mensaje de aviso, este parpadea
Comentarios	El jugador puede no saber qué hacer para dejar que parpadee el mensaje.

Tabla 3.24; Test_01_03: parpadeo del color de los mensajes

3.2.2 “GameLogic“

Este es un script que se encargará de controlar toda la lógica de menús y escenas del juego.

Al empezar el script, este activará el menú principal y la cámara del mismo, y desactivarán los demás menús.

En el método “Update()”, el cual se ejecuta en cada Frame [62], comprobará que la escena en la que se encuentra el juego no se la de los créditos. Una vez haga eso, si el jugador pulsa la tecla “ESC” y ni el menú de inicio ni la pantalla de muerte está activado, llamará al método “Resume()”. En otro caso, si el jugador pulsa la tecla “Tab” y ni el menú de inicio ni la pantalla de muerte está activado, alternará el panel de tareas entre abierto y cerrado, teniendo en cuenta que, si está cerrado y el icono de aviso de nueva tarea está activado, este se desactivará. Por último, este método controlará que el jugador muera, es decir, que su vida llegue a cero, momento en el

cual llamará al método “Die()”. Cabe destacar que el método en todo momento comprobará que el cursor del ratón esté activo si alguno de los menús está activado, y bloqueará el cursor del ratón si todos los menús están desactivados.

Ahora se va a proceder a especificar cada uno de los métodos del script y qué hace cada uno de ellos:

1. **“Die()”:** este método será llamado cuando el jugador muera, como se ha comentado antes, y activará la pantalla de muerte y desactivará los demás menús. Además, si el jugador ha completado el tutorial del juego antes de morir, se mostrará en la pantalla de muerte un texto que indicará al jugador que este no tiene que pasar por el tutorial si este reaparece. Por último, se parará el tiempo del juego para evitar que este siga corriendo y así poder evitar futuros problemas de cálculos y valores de variables.

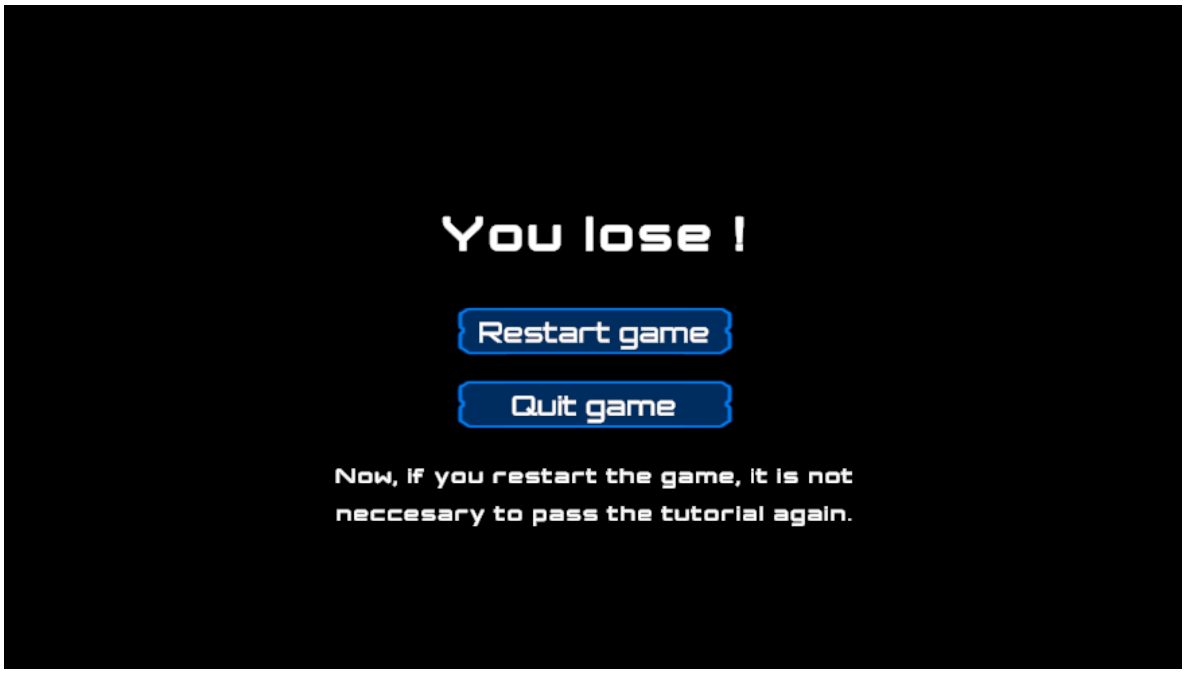


Ilustración 3.56; Pantalla de muerte tras morir

2. “**Resume()”:** este método, será llamado al jugador pulsar la tecla “ESC” y lo que hará será activar el menú de pausa y desactivará los demás o al contrario en función de si el jugador ha pulsado la tecla para pausar el juego o si la ha pulsado para quitar el menú de opciones y reanudar el juego. Con

la misma lógica, se pausará o reanudará el tiempo del juego cuando el jugador pulse la tecla “ESC”.

Ilustración 3.57; Menú de opciones tras pausar partida

3. **“StartGame()”:** este método será llamado cada vez que se quiera empezar el juego. Esto produce que se desactiven todos los menús, que se active el personaje y su interfaz, el mapeado del juego se prepare y que el tiempo de juego se reanude y no esté parado.
4. **“MenuPrincipal()”:** este método será ejecutado cada vez que se quiera empezar una partida, ya sea porque el jugador ha muerto y quiere reiniciar la partida o porque el jugador, voluntariamente, quiere reiniciar la partida desde el menú de opciones. Lo que hace este método es cargar de nuevo la escena principal del juego. No obstante, antes de eso se guardará en las preferencias del jugador una variable “TutorialCompleted” que tendrá el valor 1 si el jugador ha completado el tutorial en el momento en el que se llama al método, o un 0 si el jugador no lo ha completado. Esto es útil para guardar esta información entre carga de escenas. Después de que se cargue la nueva escena, se actualizará el valor de la variable “tutorialCompleted” del “GameManager” en función de la variable “TutorialCompleted” guardada en las preferencias del jugador antes de

cargar la nueva escena. Además, se activará el menú principal y la cámara del mismo, y se desactivarán los demás menús.

Ilustración 3.58; Menú de inicio

5. **"ToggleState()":** este método se usará para cambiar el estado de la dificultad del juego. Será llamado cada vez que el usuario pulse el botón de cambiar dificultad del juego en el menú principal y lo que hará será cambiar llamar al método "IncreaseDifficulty()" del "GameManager" y, en función del valor de la variable "difficultyCont" del "GameManager", se cambiará el color y el texto del botón de cambio de dificultad del menú de inicio.

Ilustración 3.59; Ciclo de dificultad del juego

6. **"ExitGame()":** este método controla si se está ejecutando el juego en el editor de Unity. En tal caso, detiene la ejecución del juego en el editor. Si se

está ejecutando en una ejecución, cierra la aplicación. Esto se hace para que, tanto los jugadores como el desarrollador mediante pruebas, puedan cerrar el juego independientemente del entorno en el que se encuentren.

3.2.2.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el script “GameLogic”.

Test_02_01	
Nombre de la prueba o test	Aparición de las interfaces
Objetivo	Las distintas interfaces deben aparecer cuando ocurra el evento que las inicia
Resultado	Las distintas interfaces aparecen cuando ocurre el evento que las inicia
Comentarios	El jugador puede no saber qué hacer para hacer aparecer una determinada interfaz

Tabla 3.25; Test_02_01: aparición de las interfaces

3.2.3 “SoundManager“

Este elemento es un script que será el encargado de controlar todos los sonidos del juego. Este script estará asociado a un elemento vacío de la escena llamado “SoundManager” el cual tiene asociado varios componentes “AudioSource” [63], los cuales serán los responsables de reproducir los clips de sonido del juego. El script se encargará de manejar, controlar y gestionar estos componentes para que estos reproduzcan el sonido deseado, de la forma deseada y en el momento deseado, en función de la situación en la que se encuentre el juego.

Ilustración 3.60; SoundManager

Primeramente, se van a comentar los distintos elementos que el script va a manejar. Existirán, como se ha comentado previamente, tres componentes "AudioSource". El primero de ellos, se encargará de los sonidos, tales como disparar, alarmas, golpes, pasos, … El segundo se encargará de la música, como la música del menú principal, música de combate, música del final del juego, … El tercero se usará como auxiliar a la hora de reproducir varios sonidos o música simultáneamente, es decir, por ejemplo, si hay que reproducir dos sonidos simultáneamente, el primer "AudioSource" reproducirá el primero y el tercer "AudioSource" reproducirá el segundo de forma simultánea. El mismo caso para la música.

Además de estos componentes, se tendrán los clips de audio de música y sonidos que se usarán en distintos momentos del juego. El script almacenará estos clips en una estructura de datos estática llamada “audioClips”. Habrá un total de 45 sonidos y música, los cuales son:

Sonido	Descripción y situación de uso
AcidPotionBreak	Momento en el que una poción de ácido lanzada por el jugador colisione con alguna superficie rígida y se rompa
AmmoBox	Cuando el jugador abra una caja de munición
DangerAlarm (1,2) *	Cuando alguna puerta de la nave espacial se abra
DeadRobot	Momento en el que StarLord muera
Elevator	Para el inicio y el transcurso del movimiento del elevador
ElevatorFinished	Para cuando el elevador llegue al final de su movimiento
EnemyShot	Momento en el que cualquier enemigo dispare
MeleeEnemyFalling	Cuando el enemigo Dron de combate, el único enemigo, aparte de StarLord, que es terrestre y no explota al morir, caiga derrotado al suelo.
MetalHit (1,2) *	Cuando alguno de los enemigos con ataques cuerpo a cuerpo golpee al jugador
NoAmmo	Para cuando el jugador intente disparar pero el arma no tiene munición cargada
PickUp (1,2,3,4) *	Momento en el que el jugador recoge algún ítem
PlaceBattery	Momento en el que el jugador coloca alguna batería en el panel de control de la puerta de la sala de StarLord
PlayerShip_movement	Para el movimiento de la nave espacial del jugador
PlayerShip_open	Para el momento en el que la cápsula de la nave del jugador se abra y para cuando el jugador interactúe con el entorno pulsando algún botón

PlayerShot	Para cuando el jugador dispare con su arma
PortalIdle	Momento en el que el portal del final del juego está y se mantiene activo
PotionThrow	Para cuando el jugador lance una poción de ácido.
QuitLasers	Momento en el que el jugador coge la batería de una de las salas y se quitan los láseres de la sección de recompensas
Reload	Cuando el jugador recargue su arma
RobotFury	Cuando la vida de StarLord llegue a un momento, StarLord se enfurecerá y se reproducirá este sonido.
RobotSpeech (1,2,3,4,5,6,7,8,) *	Para la interacción del jugador con algún NPC aliado del entorno.
SelectBattery	Para cuando el jugador seleccione una batería
SelectPotion	Para cuando el jugador seleccione una poción
SelectWeapon	Momento en el cual el jugador selecciona su arma
SpaceShipMovement	Para el movimiento de la nave espacial enemiga al ser llamada desde la cúpula
StartPortal	Para cuando se empiece a “cargar” el portal del final del juego
UsePotion	Al usar el jugador una poción

Tabla 3.26; Sonidos del juego

Música	Descripción y situación de uso
AmbientMusic (1,2) *	Música de ambiente que sonará cada cierto tiempo para añadir tensión al juego y para que no haya silencio durante un excesivo tiempo, lo que degradaría la experiencia de usuario.
BattleMusic (1,2) *	Música que se reproducirá cuando el jugador entre en combate.
IntroMusic	Esta música se reproducirá en el menú de inicio del juego
StarWarsMusic	Música para el final del juego, en la escena de los créditos.

ThirdFloorMusic	Para cuando el jugador derrote a StarLord y vaya hasta el hangar.

Tabla 3.27; Música del juego

* Comentar que existen ciertos sonidos y música del mismo tipo o que se reproducirán en la misma situación. Esto ocurrirá en situaciones muy recurrentes y lo que se hace es elegir uno de los sonidos o música del mismo tipo de forma aleatoria para evitar que siempre sea el mismo sonido en la misma situación.

A continuación, se van a comentar los distintos métodos que implementa el script:

1. **"Awake()":** este método se ejecutará antes incluso que el método "Start()" al iniciar la aplicación o programa. Lo que hará este método será crear la instancia del Singleton del script y creará una variable privada para cada componente "AudioSource".
2. **"Start()":** este método cogerá de las preferencias del jugador, las variables "SoundVolume" y "MusicVolume" para controlar el volumen de los componentes "AudioSource" correspondientes y se actualizará el componente "Slider" del menú de opciones del juego.
3. **"Update()":** este método, el cual se ejecutará en cada frame, comprobará el tiempo que ha pasado desde la última vez que se ha reproducido una música de combate. Si este tiempo es menor que una cantidad determinada de tiempo, se reproducirá una música de ambiente. Para que esto sea posible, el juego no deberá estar ni en ningún menú del juego, ni en la escena de créditos.
4. **"PlaySound()":** este método ejecutará la corrutina llamada "PlaySoundRoutine()". Esta corrutina se usará para reproducir un sonido. Para ello, antes de nada, comprobará que el componente "AudioSource" de los sonidos no sea nulo, que el sonido que se quiere reproducir no sea nulo y que el sonido que se quiere reproducir no es el que se está reproduciendo en ese momento. Si se cumple todo eso, entonces, el componente "AudioSource" de los sonidos reproducirá el sonido deseado. Para

conseguir el sonido deseado, el script cogerá el nombre del sonido, el cual será recibido por parámetro, y comprobará si existen varios sonidos que empiecen por el mismo nombre con el método “SeveralClipsStartWith()”. Si ese es el caso, llamará al método “GetRandomAudioClipFromSeveral()” para coger uno aleatorio de todos esos. Si no existen varios sonidos que empiecen por el mismo nombre, llamará al método “GetAudioClip()”.

5. “**StopSound()”:** este método, primero comprobará que el “AudioSource” de los sonidos no sea nulo y esté reproduciendo algún sonido. Si ese es el caso, detendrá la reproducción de ese clip.

6. **“PlayMusic()”:** este método ejecutará la corrutina llamada “PlayMusicRoutine()”. Esta corrutina se usará para reproducir una música. Para ello, antes de nada, comprobará que el componente “AudioSource” de la música no sea nulo, que la música que se quiere reproducir no sea nula y que la música que se quiere reproducir no es el que se está reproduciendo en ese momento. Si se cumple todo eso, entonces, el componente “AudioSource” de la música reproducirá la música deseada. Para conseguir el clip de música deseada, el script cogerá el nombre de la música, el cual será recibido por parámetro, y comprobará si existen varios clips que empiecen por el mismo nombre con el método “SeveralClipsStartWith()”. Si ese es el caso, llamará al método “GetRandomAudioClipFromSeveral()” para coger uno aleatorio de todos esos. Si no existen varios clips que empiecen por el mismo nombre, llamará al método “GetAudioClip()”. Además, si el clip reproducido se llama “BattleMusic”, se guardará el nombre del último de los tipos de música de batalla reproducido para que, cuando se vuelva a querer reproducir una música de batalla, se reproduzca la otra en vez de que se vuelva a reproducir la que se acaba de reproducir. Esto se hace para que no se repita siempre la misma música, cosa que cansaría al jugador, ya que la situación de combate es bastante frecuente en el juego desarrollado.

7. **“StopMusic()”:** este método, primero comprobará que el “AudioSource” de la música no sea nulo y esté reproduciendo una música. Si ese es el caso, detendrá la reproducción de ese clip.

8. **“PlayExtra()”:** este método ejecutará la corrutina llamada “PlayExtraRoutine()”. Esta corrutina se usará para reproducir un clip de audio, independientemente de que sea música o sonido. Para ello, antes de nada, comprobará que el componente “AudioSource” del audio extra no sea nulo, que el clip que se quiere reproducir no sea nulo y que el clip que se quiere reproducir no es el que se está reproduciendo en ese momento. Si se cumple todo eso, entonces, el componente “AudioSource” del audio extra reproducirá el clip deseado. Para conseguir el clip deseado, el script cogerá el nombre del clip, el cual será recibido por parámetro, y comprobará si existen varios clips que empiecen por el mismo nombre con el método “SeveralClipsStartWith()”. Si ese es el caso, llamará al método “GetRandomAudioClipFromSeveral()” para coger uno aleatorio de todos esos. Si no existen varios clips que empiecen por el mismo nombre, llamará al método “GetAudioClip()”.
9. **“StopExtra()”:** este método, primero comprobará que el “AudioSource” de los clips extras no sea nulo y esté reproduciendo un clip. Si ese es el caso, detendrá la reproducción de ese clip.
10. **“SeveralClipsStartWith()”:** este es un método que se usará para devolver un booleano que indicará si, pasado un nombre de un clip por parámetro, existen varios clips con el mismo comienzo de nombre, lo que indicará que hay varios clips para una misma situación. Para ello, se obtiene el comienzo o prefijo de un nombre con el método “GetPrefix()”. Con el prefijo del nombre, se comprueba si existe algún clip con el mismo prefijo. Si es así, el método devolverá true, si no, devolverá false.
11. **“GetPrefix()”:** este es un método auxiliar que, como se ha comentado antes, pasado un nombre por parámetro, obtendrá su prefijo y lo devolverá, el cual es la parte del nombre antes del primer dígito. Por ejemplo, teniendo el nombre “PickUp2”, el prefijo será “PickUp”.
12. **“GetAudioClip()”:** este método devolverá el clip de audio cuyo nombre sea el mismo al nombre recibido por parámetro.
13. **“GetRandomAudioClipFromSeveral()”:** este método devolverá un clip de audio aleatorio entre todos los clips que empiecen por el mismo prefijo que

tiene el nombre recibido por parámetro. Para ello, lo que se hace es coger el prefijo del nombre que se recibe por parámetro con el método "GetPrefix()" y se buscan todos los clips cuyos nombres empiecen por ese prefijo, los cuales se almacenarán en una estructura de datos dinámica, para más tarde, escoger uno de todos los clips almacenados en esa estructura de datos de forma aleatoria y devolverlo.

14. **"ChangeVolume()":** este método servirá para actualizar las variables "SoundVolume" y "MusicVolume" de las preferencias del jugador y el volumen de los componentes "AudioSource" en base al valor de los componentes "Slider" que el jugador podrá cambiar desde el menú de opciones.

Cabe mencionar que este script, al igual que el "GameManager", está implementado con el patrón de diseño "Singleton", por el mismo motivo por el que lo está el "GameManager". Gracias a esto, se puede acceder a cualquier método desde cualquier punto de la aplicación, lo que facilita la reproducción de los sonidos y música en diferentes situaciones del juego.

3.2.3.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el script "SoundManager"

Test_03_01	
Nombre de la prueba o test	Sonidos adecuados
Objetivo	Los sonidos deben estar relacionados con la acción que ocurra
Resultado	Los sonidos son acordes con la situación del juego al sonar
Comentarios	Todo OK.

Tabla 3.28; Test_03_01: sonidos adecuados

Test_03_02

Nombre de la prueba o test	Música adecuada
Objetivo	La música debe estar relacionada con el momento de la partida
Resultado	La música es acorde con la situación del juego al sonar
Comentarios	Todo OK.

Tabla 3.29; Test_03_02: música adecuada

Test_03_03	
Nombre de la prueba o test	Momento de los sonidos
Objetivo	Los sonidos deben sonar en el momento en el que sea realizan las acciones u ocurren las situaciones
Resultado	Los sonidos suenan cuando deben sonar, ni antes ni después
Comentarios	Todo OK.

Tabla 3.30; Test_03_03: momento de los sonidos

Test_03_04	
Nombre de la prueba o test	Momento de la música
Objetivo	La música debe sonar en el momento adecuado
Resultado	La música suena cuando deben sonar, ni antes ni después.
Comentarios	Todo OK.

Tabla 3.31; Test_03_04: momento de la música

Test_03_05	
Nombre de la prueba o test	Volumen de los sonidos
Objetivo	Los sonidos deben sonar con un volumen adecuado
Resultado	Los sonidos suenan con un volumen adecuado, sin que sea molesto para el jugador pero que se escuche bien
Comentarios	El jugador puede querer cambiar el volumen de los sonidos.

Tabla 3.32; Test_03_05: volumen de los sonidos

Test_03_06	
Nombre de la prueba o test	Volumen de la música
Objetivo	La música debe sonar con un volumen adecuado
Resultado	La música suena con un volumen adecuado, sin que sea molesto para el jugador pero que se escuche bien
Comentarios	El jugador puede querer cambiar el volumen de la música

Tabla 3.33; Test_03_06: volumen de la música

3.2.4 Entorno

El juego, como ya es sabido, se desarrollará en el espacio exterior en el que nos encontraremos una nave espacial y una cúpula. Aunque, antes de llegar a cualquiera de estas estructuras, se encontrará otra nave espacial más pequeña que será la que usará el jugador para empezar su aventura en el espacio exterior.

3.2.4.1 Nave del jugador

Esta será la nave que el jugador usará para llegar a la cúpula, estructura necesaria para llamar a la nave espacial.

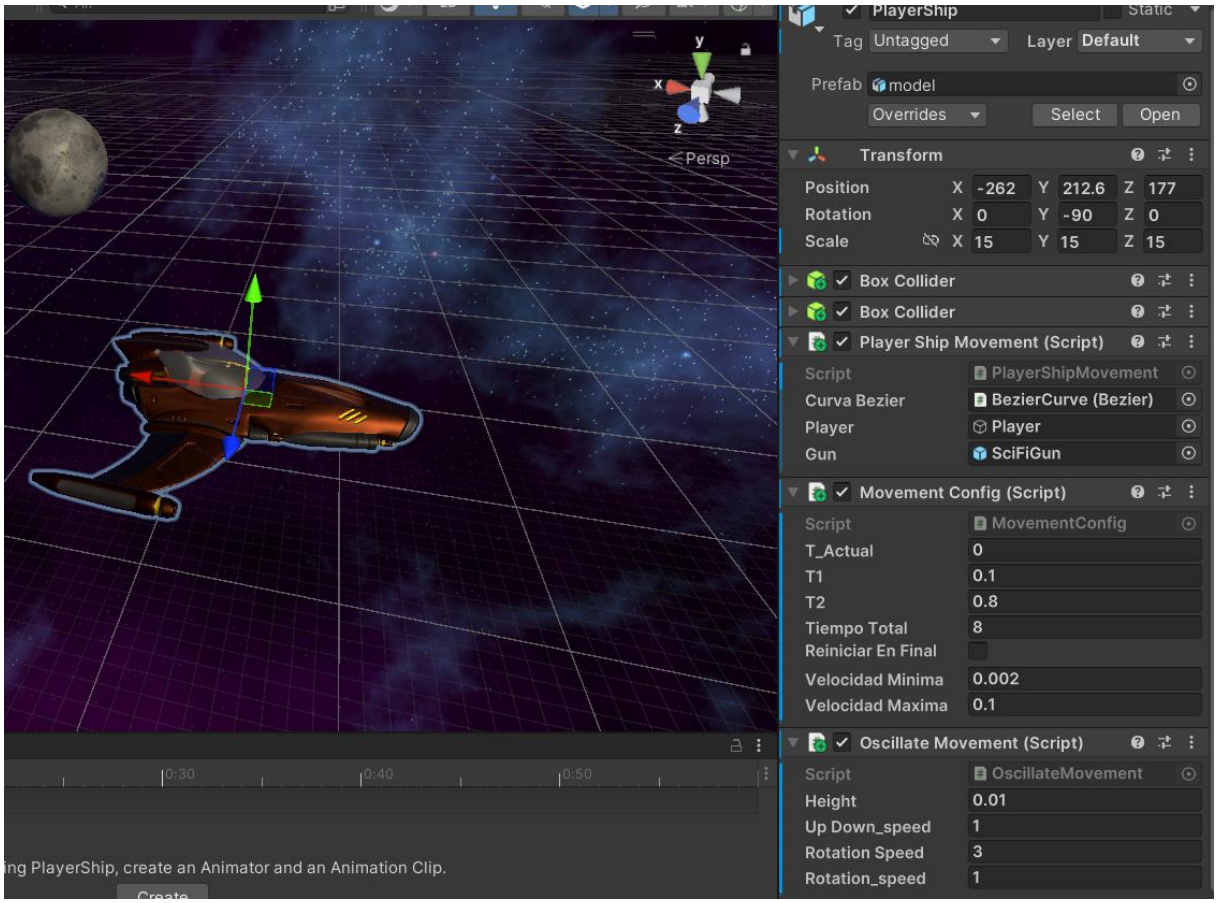


Ilustración 3.61; Nave del jugador, componentes

Esta nave espacial, está compuesta por los siguientes elementos:

Ilustración 3.62; Nave del jugador, elementos

De entre estos elementos, destacamos el objeto “PlayerPosition”, el cual es un simple objeto vacío que servirá para determinar la posicion del jugador dentro de la nave y que este no se pueda mover de ese punto, pero si mover la cámara para ver el entorno. Para conseguir esta funcionalidad, usaremos el script “PlayerShipMovement”, del cual se hablará más adelante.

Como componentes de la nave, tenemos el script “PlayerShipMovement”, el cual es el script principal de la nave y con el cual conseguimos mover la nave por el espacio exterior. Además, como se ha comentado antes, este script, al iniciar, desactiva el movimiento del jugador para que no pueda moverse dentro de la nave y se volverá a activar cuando la nave llegue al final de su movimiento. Mientras tanto, el script hace que la posicion del jugador sea constantemente, la posición del elemento “PlayerPosition” que, como se ha comentado antes, es un objeto hijo de la nave y que, por tanto, se moverá junto con ella.

Aparte de esta funcionalidad, este script también se encarga de mover la nave por una curva de Bézier [64] creada con cuatro puntos.

Esta curva de Bèzier, como se ha dicho, está definida por cuatro puntos en el espacio.

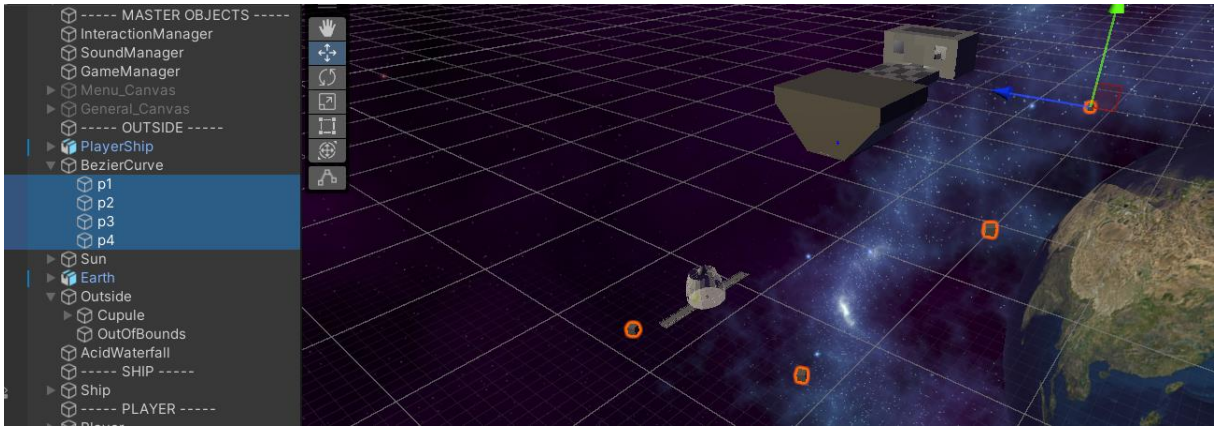


Ilustración 3.63; Nave del jugador, puntos de control de la curva de Bèzier

Estos cuatro puntos generarán una curva (marcada por las esferas rojas de la imagen) por la que pasará la nave a una velocidad constante. Cabe destacar que las esferas rojas no estarán presentes en la versión final del juego, son simplemente para mostrar cómo está definida la curva.

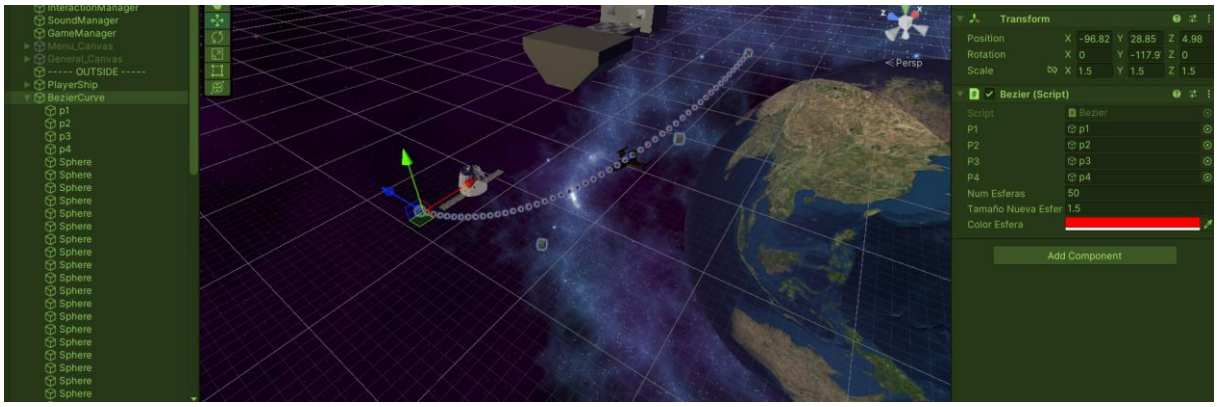


Ilustración 3.64; Nave del jugador, curva de Bèzier

Tendremos otro script llamado “MovementConfig”, con el cual podremos controlar los valores de la curva de Bèzier y, por tanto, por dónde va la nave. Podremos controlar valores como la velocidad de la nave, que recorrido de la curva lleva la nave, qué hacer y qué no hacer antes y después de que termine o empiece a moverse la nave por la curva, …

Gracias al script “MovementConfig”, cuando la nave llegue al final de la curva, se activarán dos scripts. El primero de ellos será el script “EntryRotation”, el cual está asociado a la cápsula de la nave del jugador y lo que hará será rotar la cápsula desde un punto de referencia.

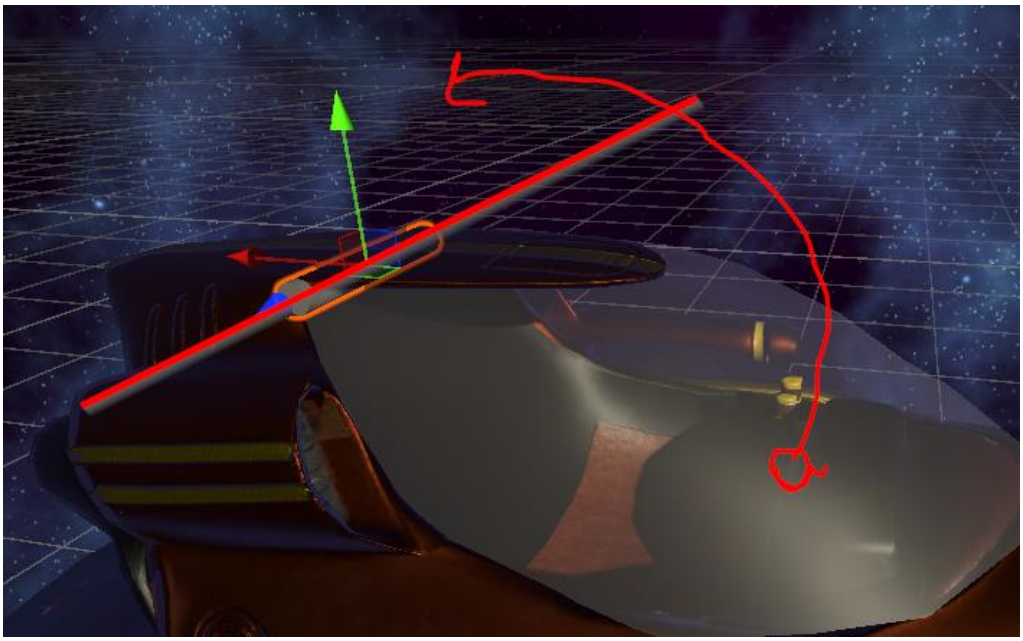


Ilustración 3.65; Nave del jugador, apertura de cápsula

Esta cápsula tiene asociado un material parecido al cristal, pero sin ser totalmente transparente, lo que facilita que el jugador pueda ver desde el interior sin dejar de verse que es un cristal. Para conseguir este efecto de transparencia, se ha ajustado el componente alfa [65] del color del material.

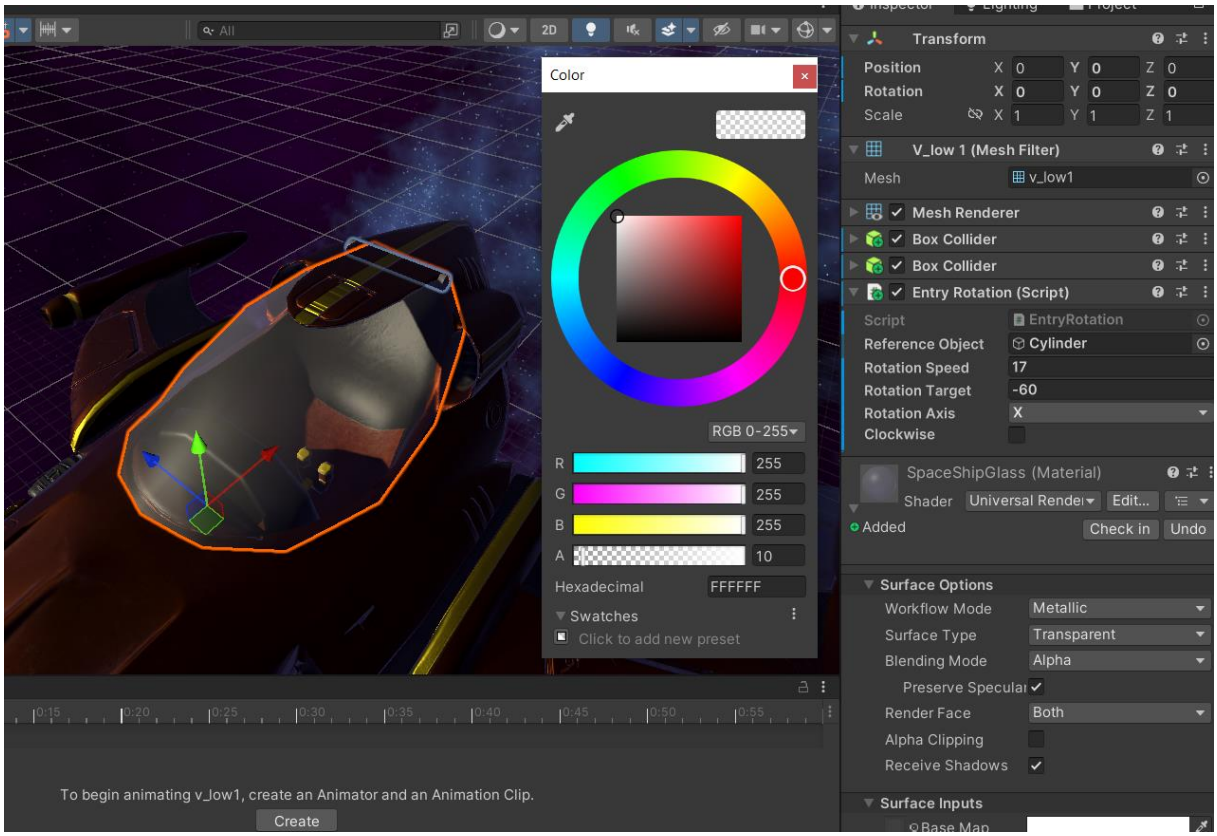


Ilustración 3.66; Nave del jugador, componentes de cápsula

Además del script “EntryRotation”, el script “MovementConfig”, al llegar la nave al final de la curva, activará también el script “OscillateMovement”, con el cual la nave pasará a hacer un movimiento de oscilación subiendo y bajando para dar la sensación de que la nave está flotando en el aire.

3.2.4.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la nave del jugador.

Test_04_01	
Nombre de la prueba o test	Apertura de cápsula
Objetivo	Abrir la cápsula de la nave de forma realista, teniendo en cuenta tiempo, velocidad y rotación.
Resultado	La cápsula se abre lentamente y de forma correcta al llegar la nave del jugador a su destino
Comentarios	Todo OK.

Tabla 3.34; Test_04_01: apertura de cápsula

Test_04_02	
Nombre de la prueba o test	Llegada de la nave
Objetivo	Hacer que la nave se detenga justo en la pasarela de la cúpula
Resultado	La nave se detiene justo en la pasarela.
Comentarios	No obstante, cuando empieza el movimiento oscilatorio de la nave, hay que tener cuidado de no caerse al vacío.

Tabla 3.35; Test_04_02: llegada de la nave

Test_04_03	
Nombre de la prueba o test	Movimiento de la nave
Objetivo	Obtener un movimiento de la nave del jugador realista
Resultado	La nave obtiene un movimiento a una velocidad constante, excepto al final de su movimiento, en el que se le aplica una técnica de frenado llamada “Ease-out”.
Comentarios	El movimiento de la nave, hace al personaje, el cual está dentro, moverse dentro de la nave, pero consigue hacerle llegar hasta su destino

Tabla 3.36; Test_04_03: movimiento de la nave

Test_04_04	
Nombre de la prueba o test	Ruta de movimiento de la nave
Objetivo	Hacer que la nave del jugador describa una curva de movimiento realista que le permita al jugador ver gran parte del entorno para que este sepa a lo que se enfrenta.
Resultado	La nave realiza un movimiento cuya ruta permite al jugador ver la mayor parte del entorno del juego.
Comentarios	Esta ruta puede hacer un poco pesado el movimiento de la nave, ya que este movimiento dura bastante tiempo

Tabla 3.37; Test_04_04: ruta de movimiento de la nave

Test_04_05	
Nombre de la prueba o test	Oscilación de la nave
Objetivo	Obtener un movimiento oscilatorio de la nave realista
Resultado	La nave obtiene un movimiento un movimiento oscilatorio bastante realista
Comentarios	El movimiento oscilatorio de la nave dificulta que el personaje pueda volver a entrar a la nave si el jugador quiere investigarla. Además, dificulta el movimiento del personaje dentro de la nave.

Tabla 3.38; Test_04_05: oscilación de la nave

3.2.4.2 La cúpula

A parte de la nave espacial del jugador, tendremos la cúpula, la cual es una estructura hecha con cubos rectangulares inclinados, dando la forma de cúpula. Además, estos cubos tendrán un material transparente asociado con un componente de color alfa bajo, pero no nulo, asemejando al cristal.

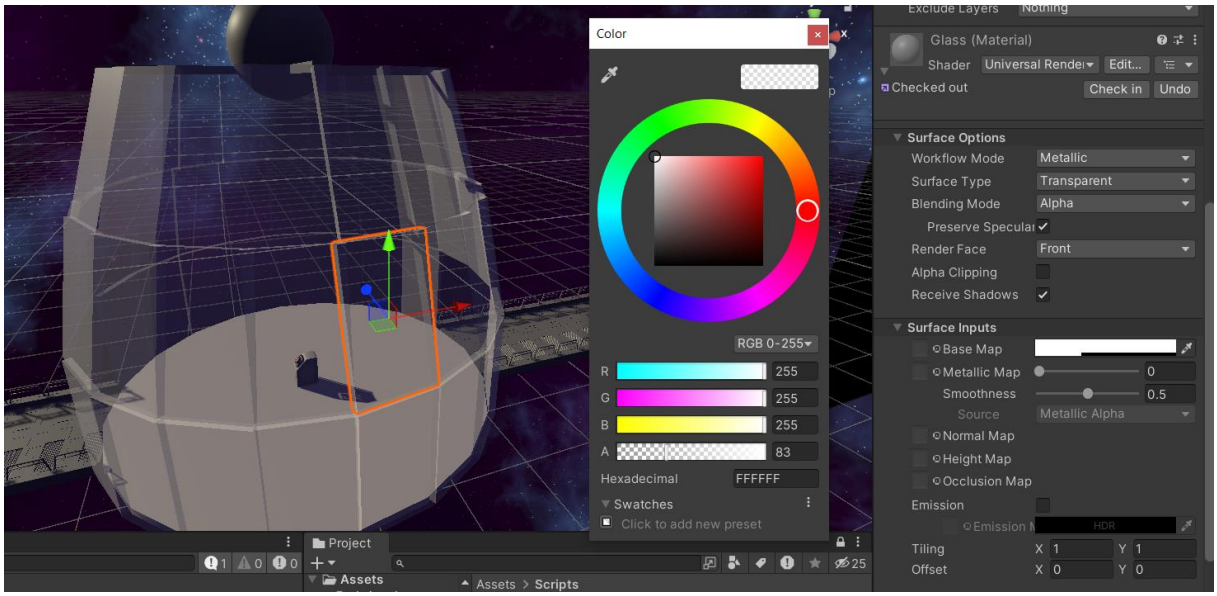


Ilustración 3.67; Cúpula, componentes del cristal

En el medio de esta cúpula, se encontrará una estructura, con el cual se podrá interactuar para llamar a la nave.

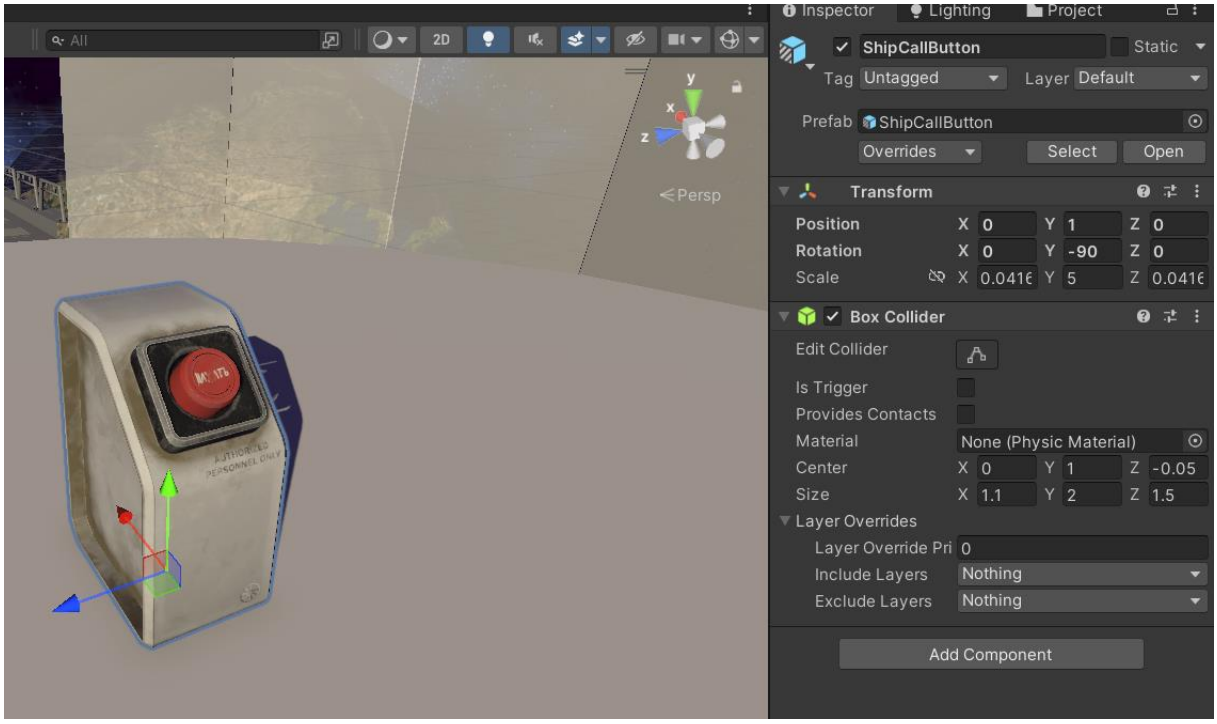


Ilustración 3.68; Cúpula, componentes de la base del botón central

Esta estructura está compuesta por varios componentes:

Ilustración 3.69; Cúpula, elementos de la base del botón central

De entre ellos, podemos destacar el elemento “Button”. Este objeto es el botón de la estructura, el cual es interactuable y con el cual podremos interactuar para llamar a la nave.

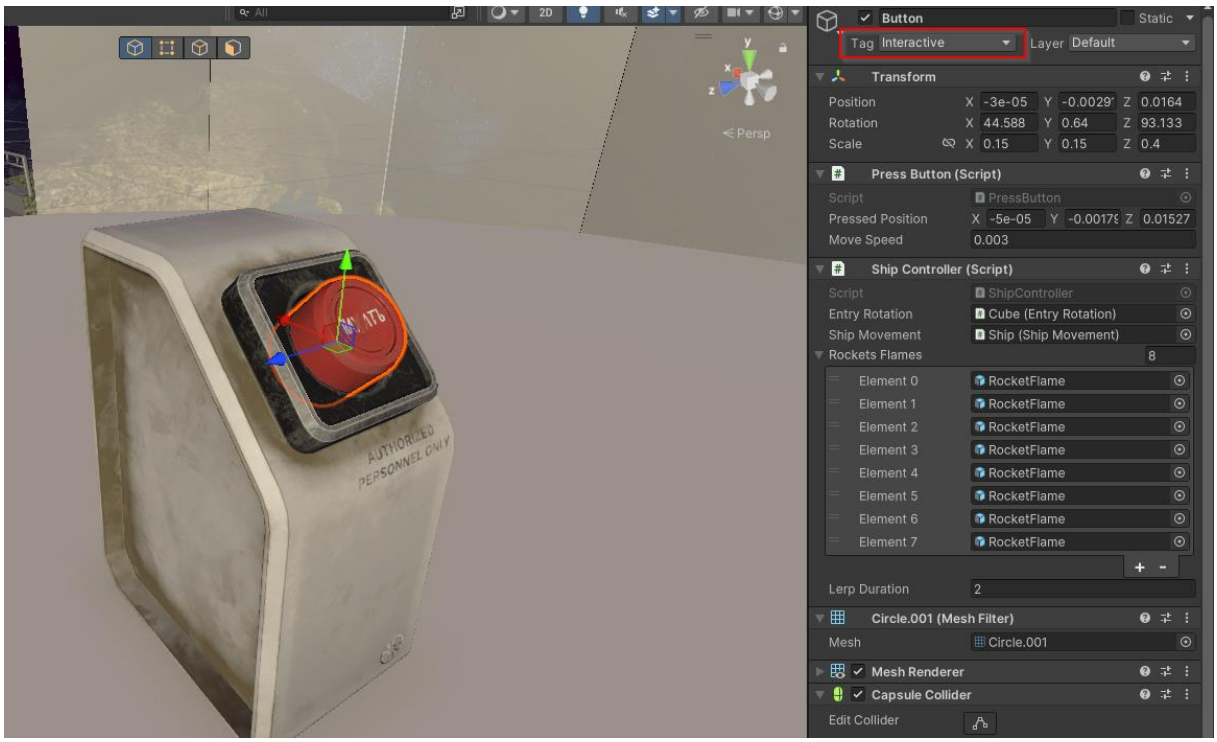


Ilustración 3.70; Cúpula, componentes del botón central

Notar que tiene la etiqueta (“tag”) [66] “Interactive”, la cual nos facilita identificar a los elementos interactuables del entorno. Entraremos más en detalle a cerca del funcionamiento de interactuar con elementos del entorno más adelante.

Este objeto tiene varios scripts asociados para conseguir la lógica de que la nave se acerque a la cúpula para poder avanzar en el juego.

El primero de ellos es el script “PressButton**”**. Este script simplemente es un script para hacer que el botón se hunda y del efecto visual de que se está presionando.

El otro script, llamado “ShipController”, es un script que, como su propio nombre indica, controla la nave. Entraremos en más detalle de este script más adelante, pero se ha comentar que, al ser pulsado el botón y, por tanto, activado el script, se produce un efecto en los “RocketFlames” que se le pasan al script. Este efecto está comentado en el **apartado 3.2.1.4.1.4 Cohetes de propulsión** de este documento.

Además de lo comentado, la cúpula también tiene unas pasarelas en parte trasera, para que el personaje pueda llegar a la cúpula desde su nave, y en la parte delantera, para que el personaje pueda llegar a la nave espacial después de haber activado el botón.

Por último, comentar que existe un plano que abarca toda el área a la que el jugador puede llegar tirándose al vacío, y que tiene un componente “MeshCollider” [67]con la opción “isTrigger” [68] marcada. A continuación, se muestra una imagen en la que se observa este plano. Cabe destacar que, en la imagen, el plano tiene un material asignado para poder visualizarlo en esta documentación. No obstante, este plano resulta invisible.

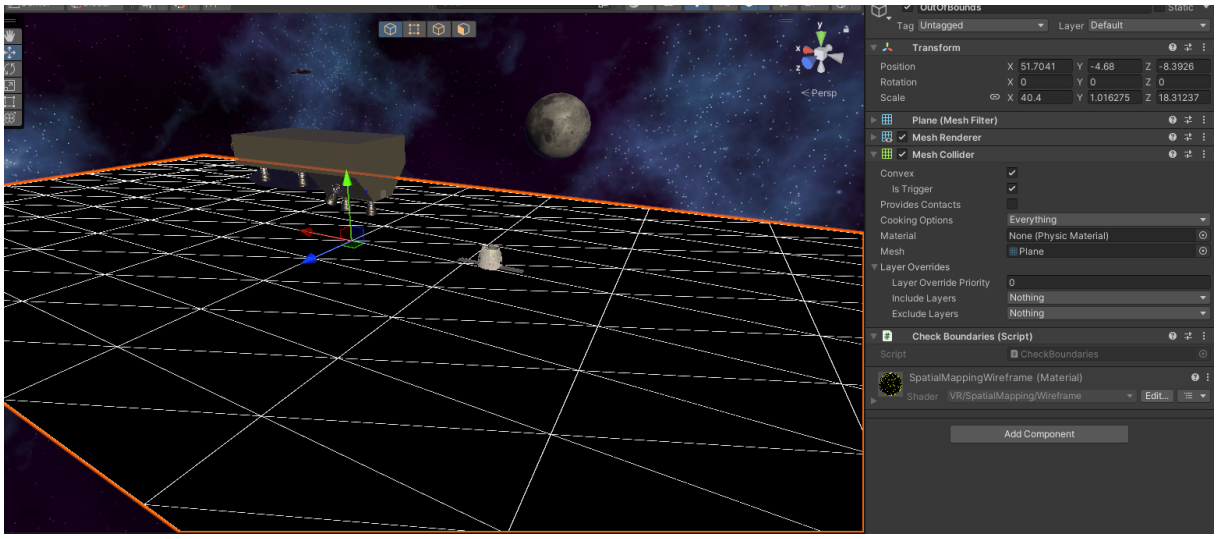


Ilustración 3.71; Plano de “out of bounds”

Como se puede apreciar en la imagen anterior, el plano tiene un script llamado "CheckBoundaries" asignado. Este script lo que hace es comprobar si alguna entidad ha entrado en contacto con el componente "Collider" [69] del plano. De ser así, si la entidad que ha entrado en contacto no es el personaje, esta entidad es eliminada. Si es el personaje, se obtiene el componente "CharacterController" [70] del personaje y se desactiva para poder mover al personaje hasta la posición inicial de la partida, la cúpula, y luego se vuelve a activar para que el jugador pueda seguir moviendo y controlando al personaje.

3.2.4.2.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la cúpula del exterior de la nave espacial.

Test_04_06	
Nombre de la prueba o test	Pulsación del botón central
Objetivo	El movimiento de pulsación del botón central para llamar la nave sea realista
Resultado	El movimiento del botón al ser pulsado es realista y es síncrono a su acción, llamar a la nave espacial enemiga.
Comentarios	El movimiento es realista, pero, depende del jugador, puede apreciarse un poco lento.

Tabla 3.39; Test_04_06: pulsación del botón central

Test_04_07	
Nombre de la prueba o test	Inicio de fuerza de los motores
Objetivo	El inicio de la fuerza de los cohetes de propulsión debe iniciarse al ser pulsado el botón central.
Resultado	El efecto de los cohetes de propulsión se inicia justo al pulsar el botón central.
Comentarios	No es un efecto muy realista puesto que la nave se empieza a mover antes de que el efecto de los cohetes inicie

Tabla 3.40; Test_04_07: inicio de fuerza de los motores

Test_04_08	
Nombre de la prueba o test	Regreso al caer
Objetivo	El jugador debe regresar a la cúpula al caer al vacío.
Resultado	El jugador, al caer al vacío desde cualquier parte, regresa a la cúpula.
Comentarios	El jugador puede no saber qué va a pasar si cae al vacío y puede llegar a ser desmotivador y decepcionante el hecho de que reaparezca en el principio al caer al vacío, en vez de reaparecer donde cayó.

Tabla 3.41; Test_04_08: regreso al caer

3.2.4.3 Nave espacial

Por último, hablando del entorno, se debe destacar la nave espacial, donde se desarrollará la mayor parte del juego.

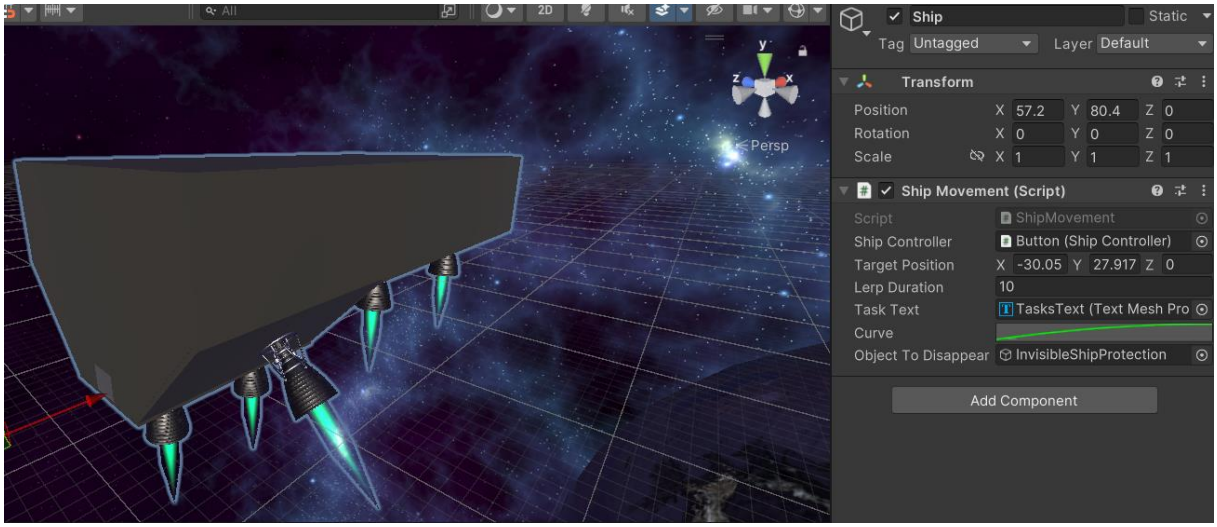


Ilustración 3.72; Nave espacial, componentes

La nave espacial como tal tiene un script asociado. Este script se llama “ShipMovement”.

Este script es llamado cuando se pulsa el botón de la cúpula y se ejecuta el script “ShipController”, visto en el apartado anterior. Antes de que “ShipController”

llame al script “ShipMovement”, se aumenta el tamaño y se desplazan los fuegos de los motores que tiene la nave usando la función “Lerp”, para dar sensación de movimiento de la nave. Con los fuegos de los motores aumentados, se llama al script “ShipMovement”, el cual hace que la nave se desplace con una curva de velocidad definida (la cual se puede ver en la imagen) de un punto A (punto inicial) a un punto B definido y pasado por parámetro. La curva de velocidad, como se puede ver, hace que la nave empiece a velocidad constante, pero a medida que se acerca al punto B, su velocidad decrezca, para dar sensación como de que está llegando a su destino. Además, se actualiza la lista de tareas, añadiendo la nueva tarea de “explorar la nave” para que el jugador sepa lo que tiene que hacer ahora.

Cuando la nave llegue al punto B fijado, el script “ShipController” llamará a un nuevo script: “EntryRotation”. Este script, al igual que se hacía con la cápsula de la nave espacial del jugador, rota un objeto con respecto a otro objeto de referencia. En este caso, este script estará asignado a la compuerta de entrada a la nave. Esta compuerta se abrirá rotando hasta unos grados con respecto a un cilindro que estará en la parte inferior.

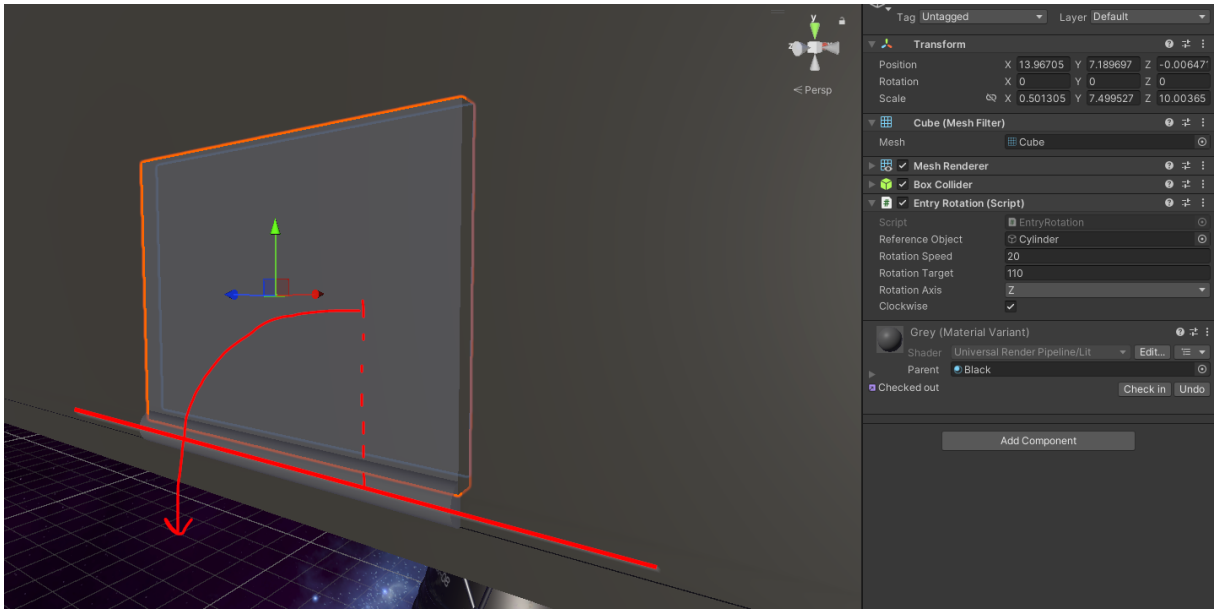


Ilustración 3.73; Nave espacial, apertura de puerta

Esta nave, por dentro, consta de tres plantas y un ascensor entre la primera y la segunda planta.

3.2.4.3.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la nave espacial.

Test_04_09	
Nombre de la prueba o test	Llegada de la nave espacial
Objetivo	La nave espacial debe detenerse en un sitio adecuado para que el jugador pueda entrar a la nave.
Resultado	La nave espacial se detiene justo al final de la pasarela de la cúpula, permitiendo al jugador poder entrar a la nave
Comentarios	Todo OK.

Tabla 3.42; Test_04_09: llegada de la nave espacial

Test_04_10	
Nombre de la prueba o test	Apertura de la puerta de la nave espacial
Objetivo	La puerta de la nave espacial debe abrirse justo cuando esta llega al final de su camino, a una velocidad adecuada y de la manera correcta
Resultado	La puerta de la nave espacial se abre correctamente en todas sus formas
Comentarios	Depende del jugador, esta rotación puede ser lenta

Tabla 3.43; Test_04_10: apertura de la puerta de la nave espacial

Test_04_11	
Nombre de la prueba o test	Fin de fuerza de los motores
Objetivo	El fin de la fuerza de los cohetes de propulsión debe iniciarse al llegar la nave espacial a su destino
Resultado	El efecto de los cohetes de propulsión se detiene justo al llegar la nave a su destino
Comentarios	No es un efecto muy realista puesto que la nave se detiene antes de que el efecto de los cohetes se detenga.

Tabla 3.44; Test_04_11: fin de la fuerza de los motores

3.2.4.3.2 Primera planta

En la primera planta de la nave espacial, se encontrará a Kyle, el primer NPC con el que se podrá hablar y mantener una conversación con él. Aunque más adelante se detallará su funcionamiento, su función en el juego es hacerle al jugador una especie de tutorial para que este aprenda a jugar.

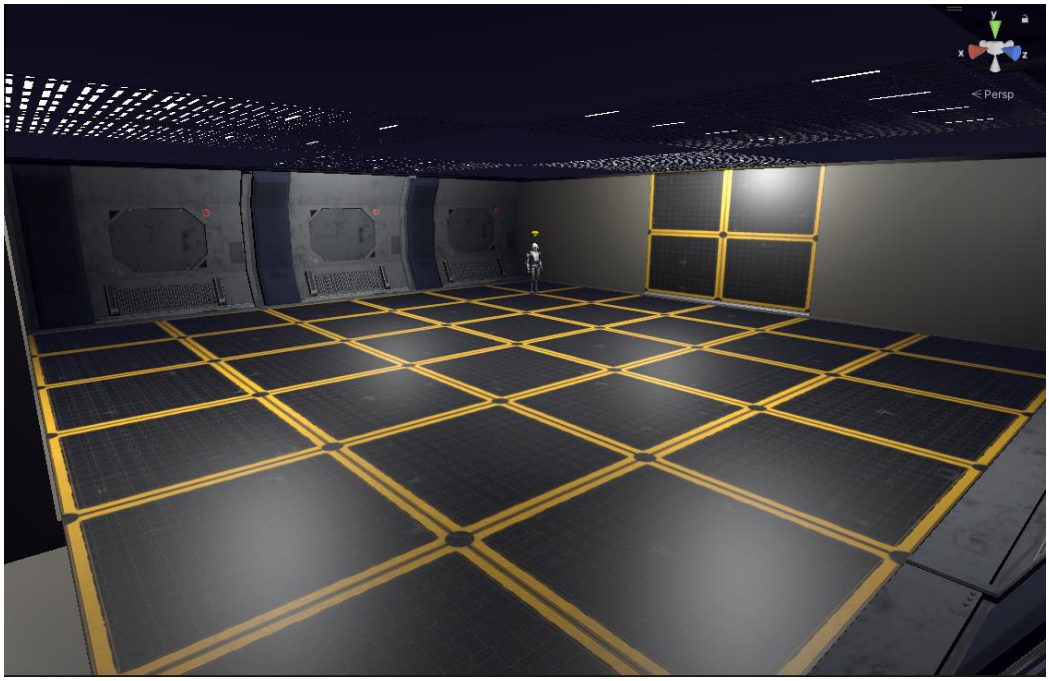


Ilustración 3.74; Primera planta, primera sala

3.2.4.3.2.1 Pared invisible

Para que el jugador no se pueda saltar el tutorial, se ha creado una pared invisible en la salida de la sala, la cual conecta con el ascensor. Esta pared será invisible a distancia, pero irá apareciendo en función de lo que el jugador se acerque a la pared.

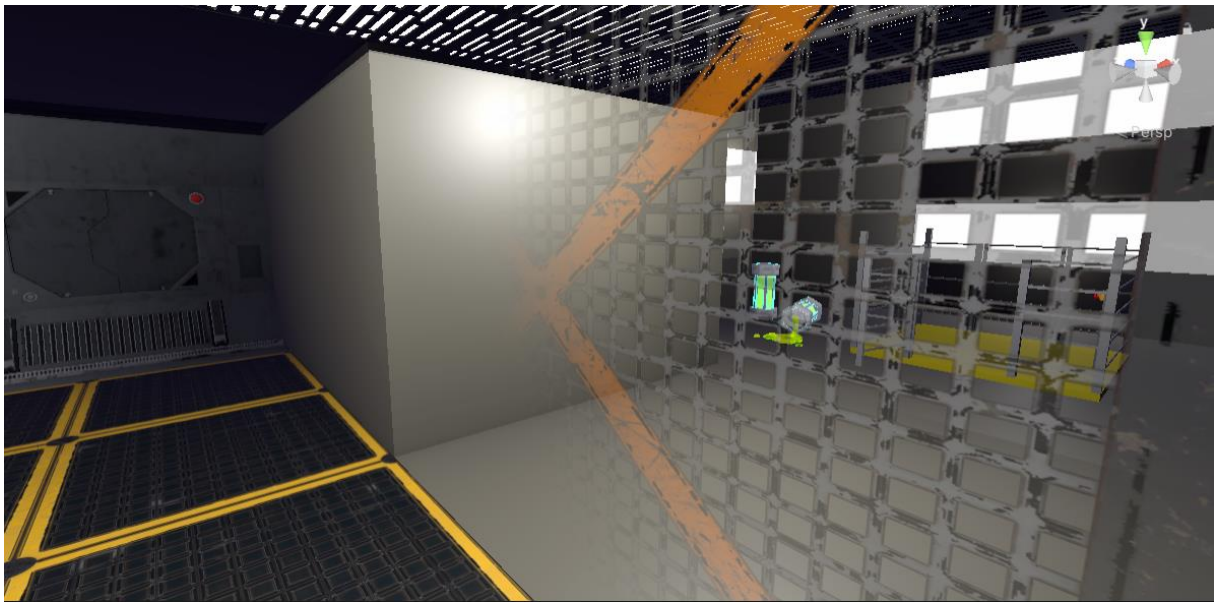


Ilustración 3.75; Primera planta, muro invisible

Esto se ha conseguido con un shader [71] llamado “InvisibleWall”, el cual está asociado al material aplicado a esta pared. Lo que se hace en el shader es aplicar un efecto de desvanecimiento basado en la distancia la pared y la cámara del jugador. El efecto que aporta el shader es que, a medida que la pared se aleja de la cámara, se vuelve más transparente, y a medida que la pared se acerca a la camara, se vuelve más opaco.

Este shader, llamado “InvisibleWallShader” ha sido desarrollado en lenguaje HLSL [72]. De forma detallada, el funcionamiento del shader es que, primeramente, se define una propiedad llamada “MainText”, la cual almacenará la textura de la pared invisible que aparecerá al estar pegado el jugador a la pared.

Luego, se define una sección “SubShader” [73], en la que se configura cómo el shader interactuará con el motor de renderizado de Unity. Aquí, se especifican unas etiquetas que permiten al objeto ser tratado como un objeto transparente, lo que resulta crucial para el efecto que se quiere lograr.

Ahora se procede a definir cómo actúa el Vertex Shader [74] y el Fragment Shader [74]. El Vertex Shader procesará cada vértice de la textura y se los pasará al Fragment Shader, el cual se encargará de procesar el píxel de cada vértice. Para ello,

lo que hará será aplicarle a cada uno de ellos la función “Frag()”, con la cual se le aplicará al pixel efecto de transparencia basado en la distancia con la cámara. No obstante, el valor de transparencia que se aplica en esta función se obtiene con otra función llamada “SmoothStep()”, el cual calcula el valor de transparencia en función de la distancia del objeto con la cámara.

3.3.4.3.2.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la pared invisible de la primera planta.

Test_04_12	
Nombre de la prueba o test	Transparencia del muro invisible
Objetivo	El muro debe permanecer invisible hasta que el jugador se acerque
Resultado	El muro no se aprecia de lejos y, a medida que el jugador se va acercando al muro, este se va volviendo visible.
Comentarios	Todo OK.

Tabla 3.45; Test_04_12: transparencia del muro invisible

3.3.4.3.2.2 Ascensor

Cuando se supere el tutorial impuesto por Kyle, se llegará al ascensor.

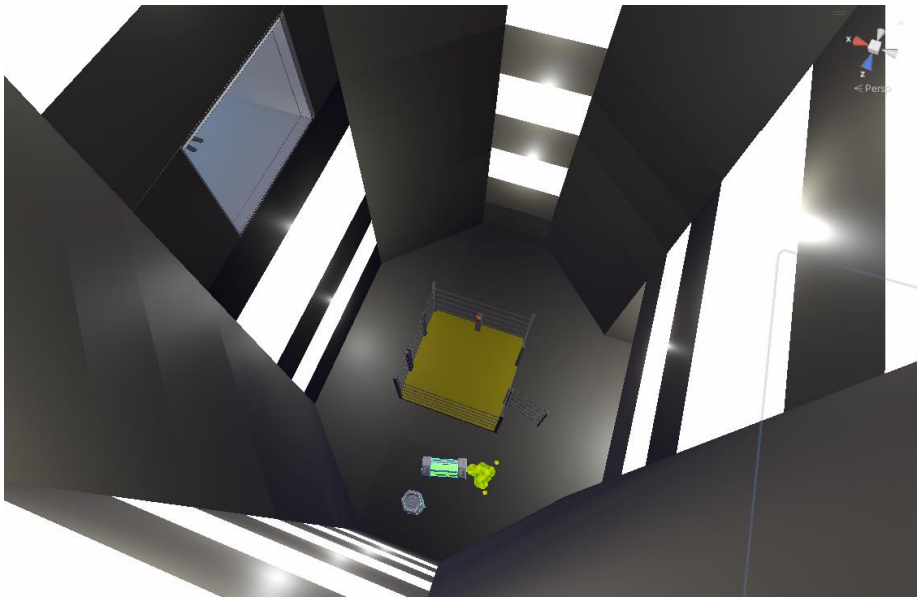


Ilustración 3.76; Primera planta, ascensor

En este ascensor, se encontrarán: un elevador, unos tanques de ácido y una habitación.

3.3.4.3.2.2.1 Elevador

El elevador servirá para llevar al jugador desde la primera planta hasta la segunda planta.

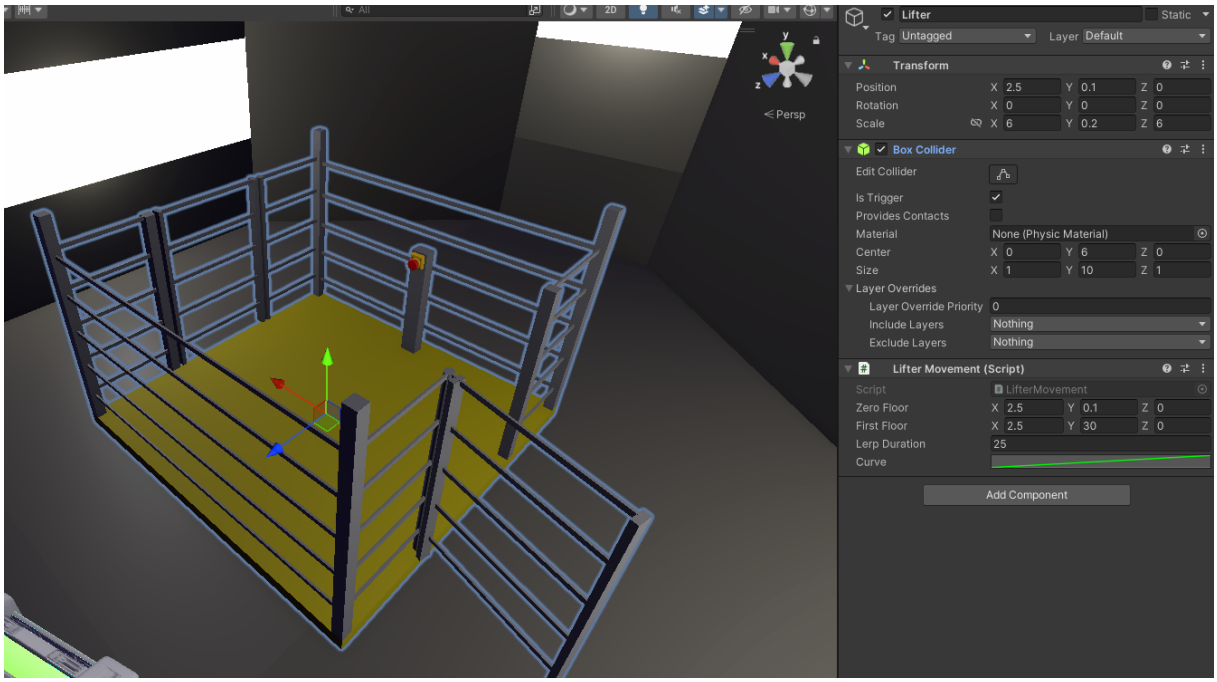


Ilustración 3.77; Primera planta, componentes del elevador

Esta estructura tendrá asignada un script llamado “LifterMovement”, el cual moverá al ascensor desde un punto A hasta un punto B usando la función “Lerp” acompañado de una curva de velocidad, la cual se puede ver en la imagen y se puede decir que el ascensor tendrá velocidad constante durante todo el trayecto. Este script, además de mover el elevador, cuando se detecta que el jugador entra en el elevador (por este motivo tenemos un componente “BoxCollider” [75] con opción “isTrigger”), este cogerá información del jugador para poder acceder a su componente “CharacterController” y así poderle aplicar un factor “displacement” positivo al eje Y de su posicion. Esto se hace así para elevar constantemente un poco al jugador y así evitar fallos a la hora de desplazar al jugador por el recorrido del elevador.

Tendremos también dos puertas que se abrirán y se cerrarán para prohibir y ceder el paso al jugador para que este pueda entrar al elevador en la primera planta y salir de él en la segunda planta.

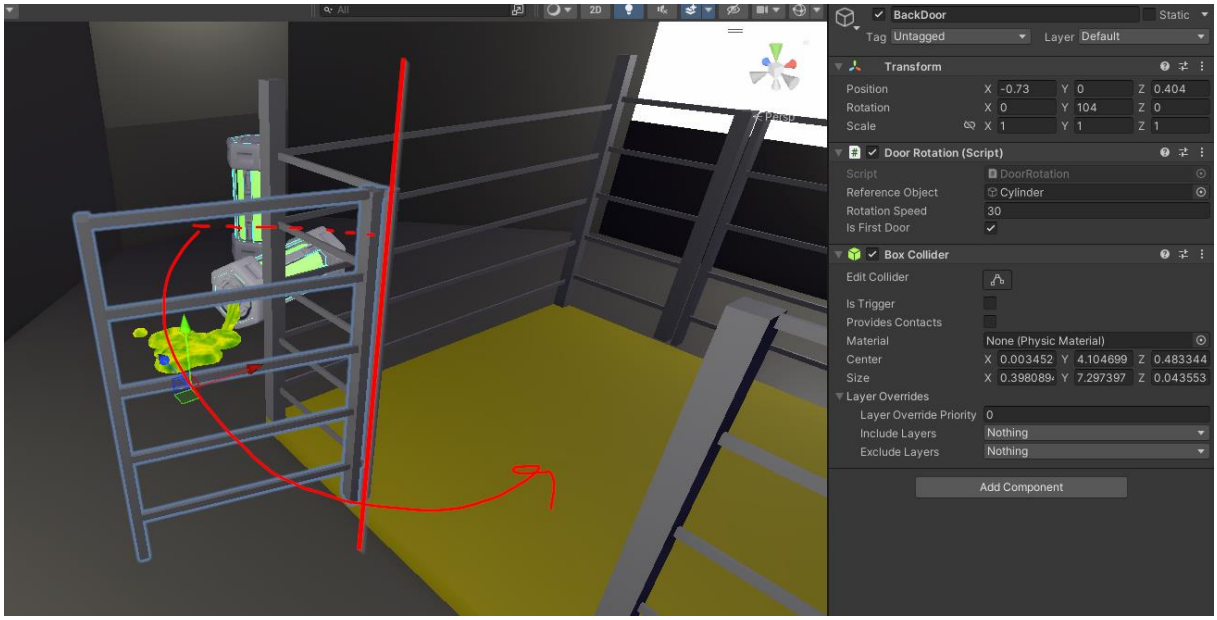


Ilustración 3.78; Primera planta, rotacion de puerta de elevador

Ambas puertas tienen el mismo funcionamiento, por lo que voy a comentar solo la puerta frontal. Estas puertas, tienen asignado un script llamado “DoorRotation”. Este script, al igual que el script “EntryRotation” antes visto, rota el objeto asociado con respecto a otro objeto, en este caso, un cilindro que junta la valla del elevador con la puerta. A diferencia del script “EntryRotation”, el script “DoorRotation” tiene un parámetro booleano “isFirstDoor”, con el cual se puede controlar si estamos haciendo referencia a la primera puerta (puerta delantera) o a la segunda puerta (puerta trasera). Esto es importante puesto que, como el mismo script está asociado a ambas puertas, hay que diferenciar la puerta a la que se hace referencia por motivos de claridad y evitar errores a la hora de invocar eventos de los scripts.

Además de rotar las puertas, este script, en el caso de la primera puerta, activará un elemento con un componente “BoxCollider” con el cual se conseguirá que el jugador no pueda salir del elevador una vez iniciada la rotación de la puerta de entrada. Este elemento se desactivará cuando la puerta llegue a su rotación final.

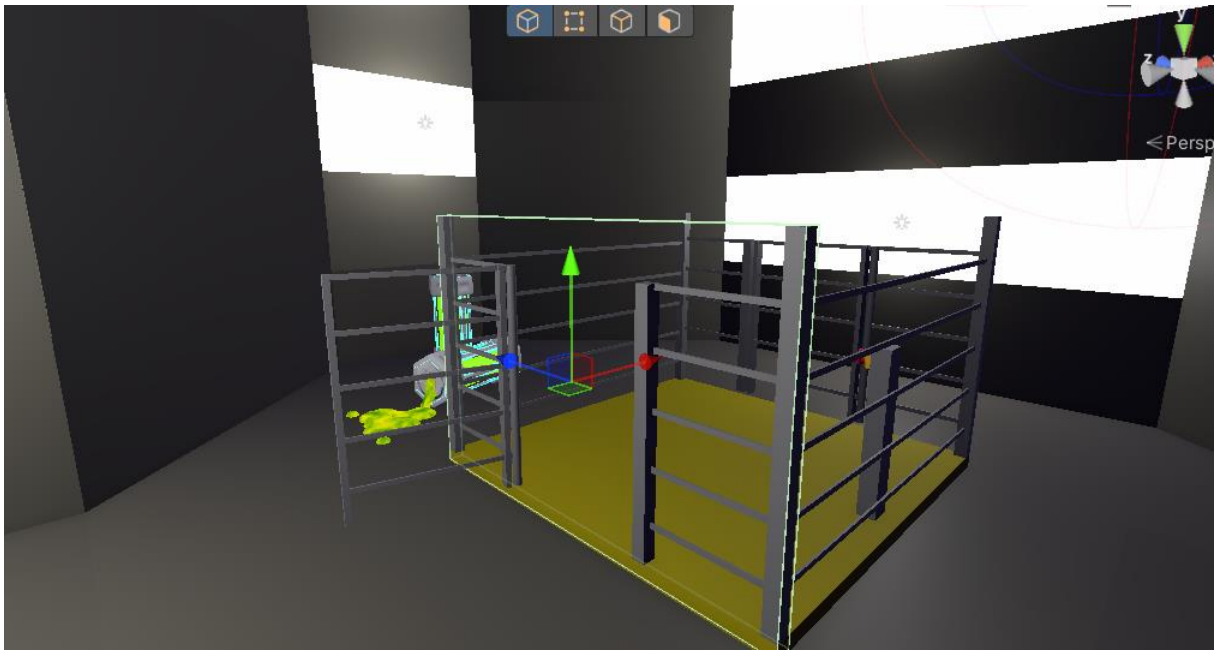


Ilustración 3.79; Primera planta, collider de la primera puerta del elevador

No obstante, para activar estos scripts, se tendrá que pulsar un botón que estará accesible a la entrada del elevador.

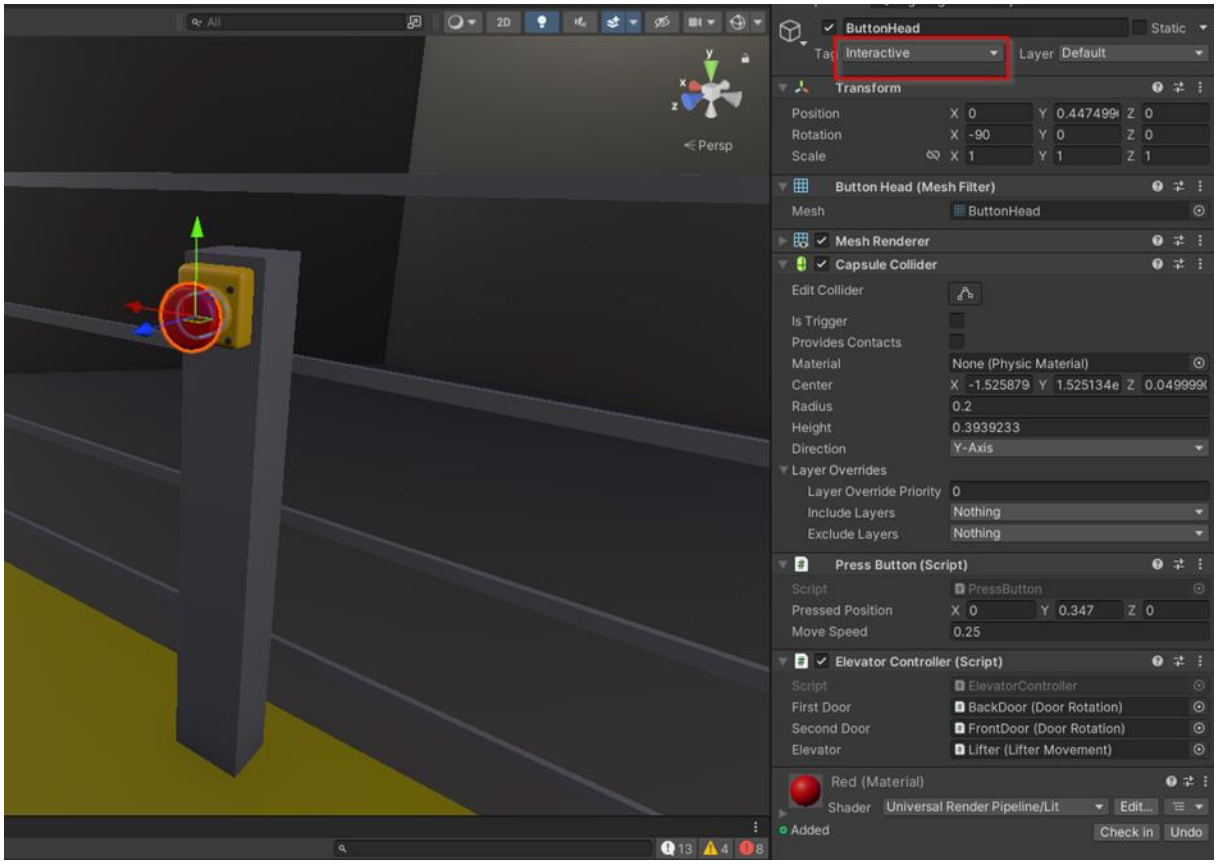


Ilustración 3.80; Primera planta, componentes del botón del elevador

Este objeto tiene una lógica y un funcionamiento parecido al botón de la cúpula. Tiene un script llamado “ElevatorController” asociado cuyo funcionamiento es que, cuando el botón se pulse, se activa el script “DoorRotation” de la puerta de adelante para cerrarla, luego de que se cierre la puerta frontal, se llama al script “LifterMovement” y cuando el elevador esté en el punto final fijado, se llama al script “DoorRotation” de la puerta de atrás para abrirla y que el jugador pueda salir a la segunda planta.

Además, el botón del elevador, al igual que el botón de la cúpula, tiene un script asociado llamado “PressButton”, con el que el botón se mete para adentro, dando la sensación de que se está pulsando.

También fijarnos que, al igual que el botón de la cúpula, este tiene la etiqueta “Interactive” asociado, lo cual facilita la interacción con él (como se ha comentado en el apartado de la cúpula, la interacción con el entorno se hablará más adelante).

3.3.4.3.2.2.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el elevador del hueco del ascensor.

Test_04_13	
Nombre de la prueba o test	Llegada del elevador
Objetivo	El elevador debe llegar a su destino
Resultado	El elevador llega a su destino sin pasarse ni quedarse corto
Comentarios	La velocidad de movimiento del elevador puede ser un poco lenta según la perspectiva del jugador.

Tabla 3.46; Test_04_13: llegada del elevador

Test_04_14	
Nombre de la prueba o test	Movimiento del personaje dentro del elevador
Objetivo	El personaje debe poder moverse dentro del elevador correctamente

Resultado	El personaje puede moverse libremente por el ascensor
Comentarios	El personaje “retumba” debido a la velocidad de movimiento excesiva del elevador.

Tabla 3.47; Test_04_14: movimiento del personaje dentro del elevador

Test_04_15	
Nombre de la prueba o test	Límites del elevador
Objetivo	El jugador no puede salirse del elevador una vez iniciado su movimiento
Resultado	El jugador, por mucho que salte, corra o se mueva, no puede escapar del elevador hasta que este llegue arriba y abra las puertas
Comentarios	Todo OK.

Tabla 3.48; Test_04_15: límites del elevador

Test_04_16	
Nombre de la prueba o test	Rotación de las puertas del elevador
Objetivo	Las puertas del elevador deben rotar a una velocidad adecuada y en el sentido correcto en función de si se abren o si se cierran
Resultado	Ambas puertas rotan correctamente en ambos casos
Comentarios	Las puertas a veces rotan en exceso si la velocidad de rotación es muy alta.

Tabla 3.49; Test_04_16: rotación de las puertas del elevador

Test_04_17	
Nombre de la prueba o test	Puerta de salida se abre al terminar movimiento elevador
Objetivo	La puerta del elevador debe abrirse cuando este llegue al final de su movimiento
Resultado	La puerta del elevador se abre justo cuando este llega al final de su movimiento
Comentarios	Velocidad de rotación puede ser un poco lenta

Tabla 3.50; Test_04_17: Puerta de salida se abre al terminar movimiento del elevador

Test_04_18	
Nombre de la prueba o test	Movimiento del elevador
Objetivo	El elevador debe iniciar su movimiento al cerrarse la puerta de entrada
Resultado	El elevador se empieza a mover justo al cerrarse la puerta de la entrada
Comentarios	Todo OK.

Tabla 3.51; Test_04_18: movimiento del elevador

Test_04_19	
Nombre de la prueba o test	Puerta de la entrada al elevador
Objetivo	El jugador no debe poder salir del elevador una vez que la puerta de entrada a este empiece a rotar
Resultado	El jugador no puede salir
Comentarios	Todo OK.

Tabla 3.52; Test_04_19: puerta de la entrada al elevador

Test_04_20	
Nombre de la prueba o test	Movimiento de botón al ser pulsado
Objetivo	El botón del movimiento del elevador debe “hundirse” al ser pulsado por el jugador
Resultado	El botón se hunde al ser pulsado por el jugador
Comentarios	El movimiento del botón puede llegar a ser lento dependiendo de la perspectiva del jugador.

Tabla 3.53; Test_04_20: movimiento del botón al ser pulsado

Test_04_21	
Nombre de la prueba o test	Cierre de la puerta del elevador al pulsar el botón

Objetivo	La puerta de la entrada del elevador debe cerrarse al pulsar el botón
Resultado	La puerta de la entrada del elevador comienza a cerrarse justo al pulsar el botón.
Comentarios	Todo OK.

Tabla 3.54; Test_04_21: cierre de la puerta del elevador al pulsar botón

3.3.4.3.2.2.2 Habitación

A parte del elevador en sí, podemos encontrar una habitación en mitad del recorrido del elevador.

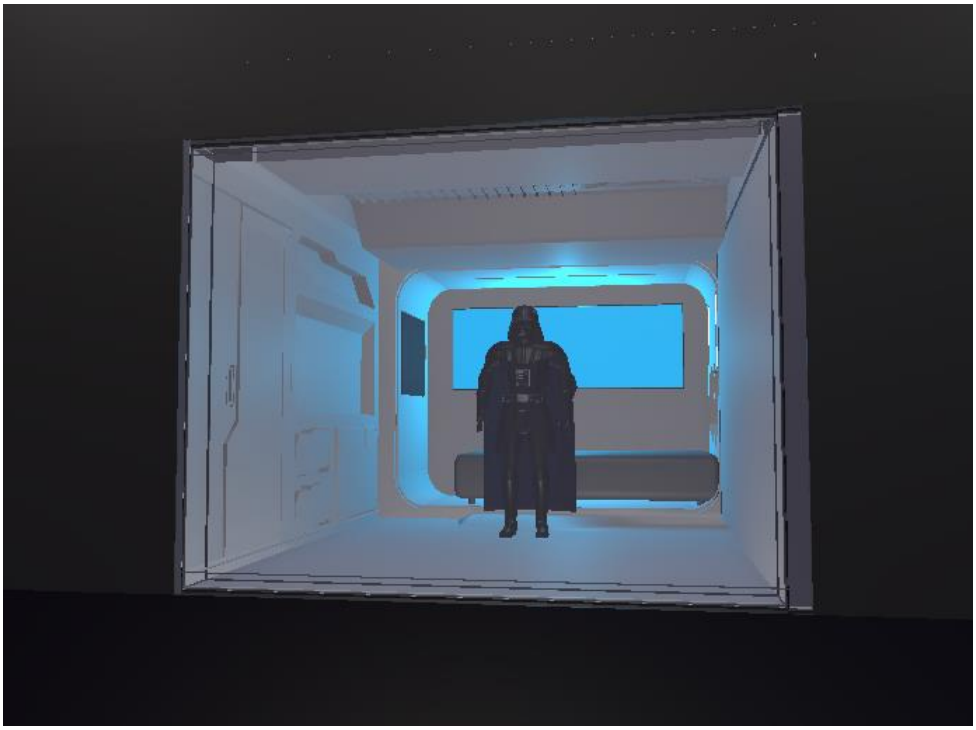


Ilustración 3.81; Primera planta, habitación de Darth Vader

En esta habitación estará encerrado Darth Vader, el cual no afecta en nada en el estado del juego, pero sí que aporta a su historia, indicando que él intentó enfrentarse a StarLord, pero no pudo salir victorioso.

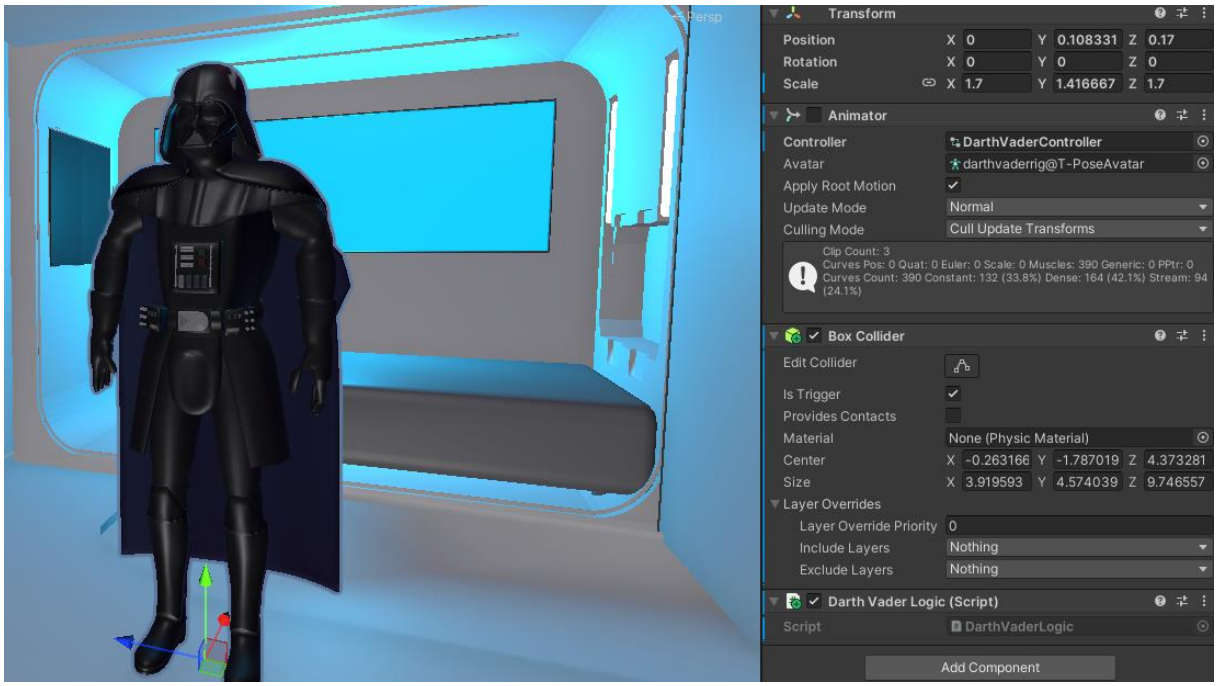


Ilustración 3.82; Primera planta, componentes de Darth Vader

Este NPC tendrá un componente “AnimatorController” [76] el cual gestionará todas las animaciones propias del modelo. El flujo de animaciones del objeto es: primero se levanta (Standing Up), luego saluda (Salute) y, por último, se queda en estado idle (Idle)

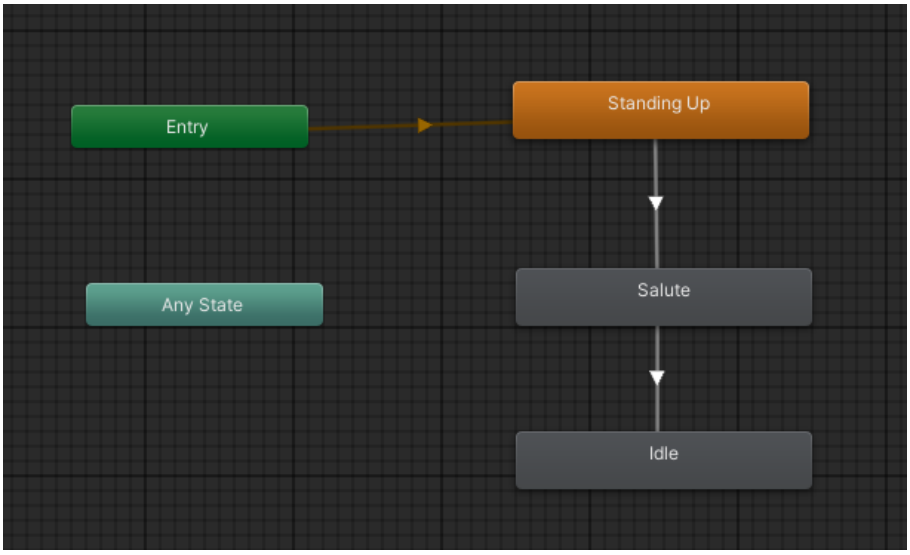


Ilustración 3.83; Primera planta, AnimatorController de DarthVader

Además, Darth Vader tendrá un script llamado “DarthVaderLogic” el cual activará el componente “AnimatorController” y, por tanto, empezará con sus

animaciones en el momento en el que el jugador entre en el componente “BoxCollider” asociado al objeto.

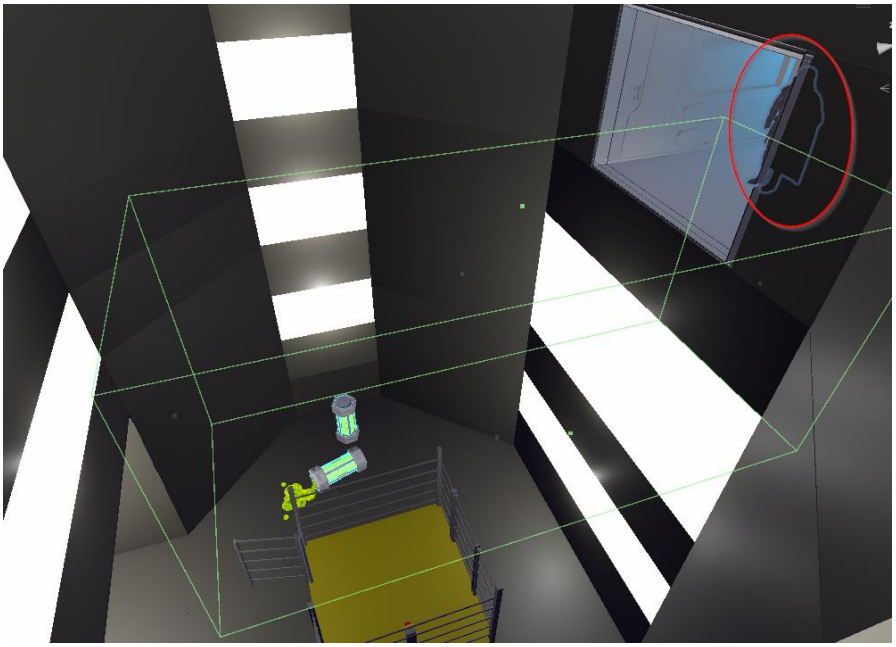


Ilustración 3.84; Primera planta, collider de Darth Vader

Como se puede ver, el componente “BoxCollider” está en la trayectoria del elevador, en el que estará el jugador. Por lo que, en el momento en el que el elevador, junto con el jugador, suba una determinada altura, se activará la animación de Darth Vader.

3.3.4.3.2.2.2.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el NPC de la habitación.

Test_04_22	
Nombre de la prueba o test	Inicio de animación
Objetivo	La animación del NPC debe iniciarse al pasar el jugador
Resultado	La animación del NPC inicia en el momento preciso
Comentarios	Si el jugador salta, puede ver el estado del NPC antes de iniciar la animación

Tabla 3.55; Test_04_22: inicio de animcación

Test_04_23	
Nombre de la prueba o test	Flujo de animación
Objetivo	El NPC debe realizar un flujo de animación con sentido
Resultado	La animación realizada por el NPC se adapta al contexto de la situación del juego.
Comentarios	Si el jugador no sabe la historia o “lore” del juego, puede ser desconcertante y no entender la situación.

Tabla 3.56; Test_04_23: flujo de animación

3.3.4.3.2.2.3 Tanques de ácido

En cuanto a los tanques de ácido, simplemente están ahí para decorar.

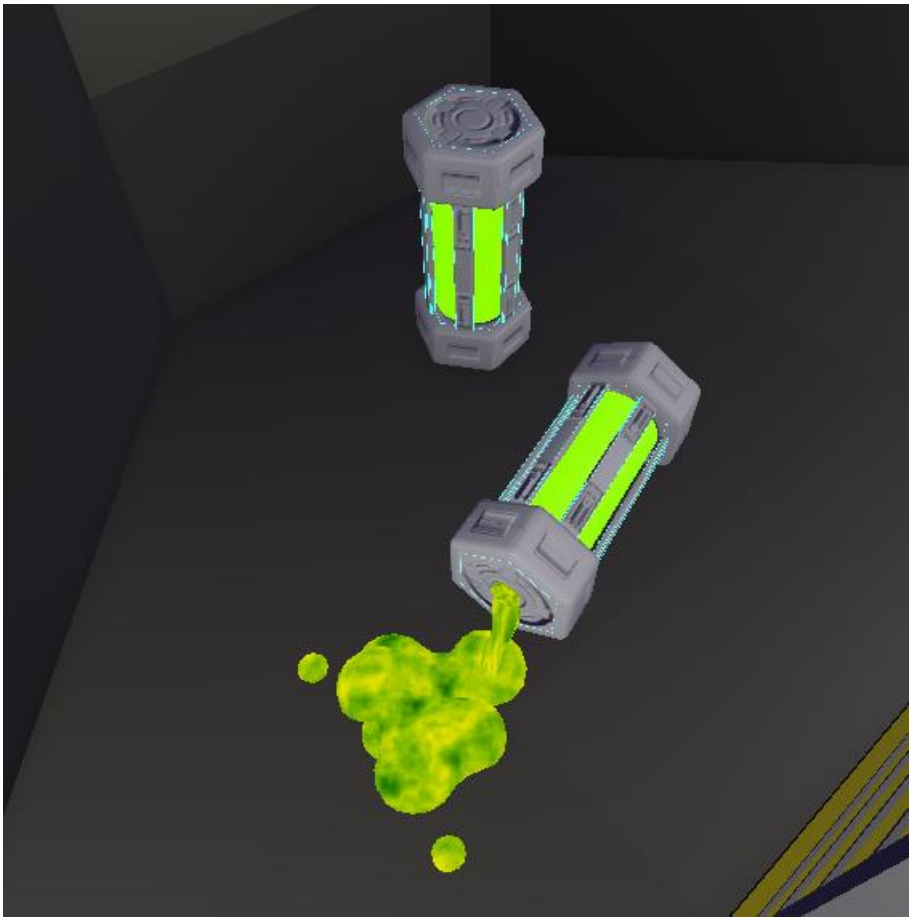


Ilustración 3.85; Primera planta, tanques de ácido

No obstante, el líquido que sale del tanque es ácido, por lo que tiene un script asociado llamado “AcidEffect” el cual aplicará daño al jugador si este entra en contacto con el líquido.

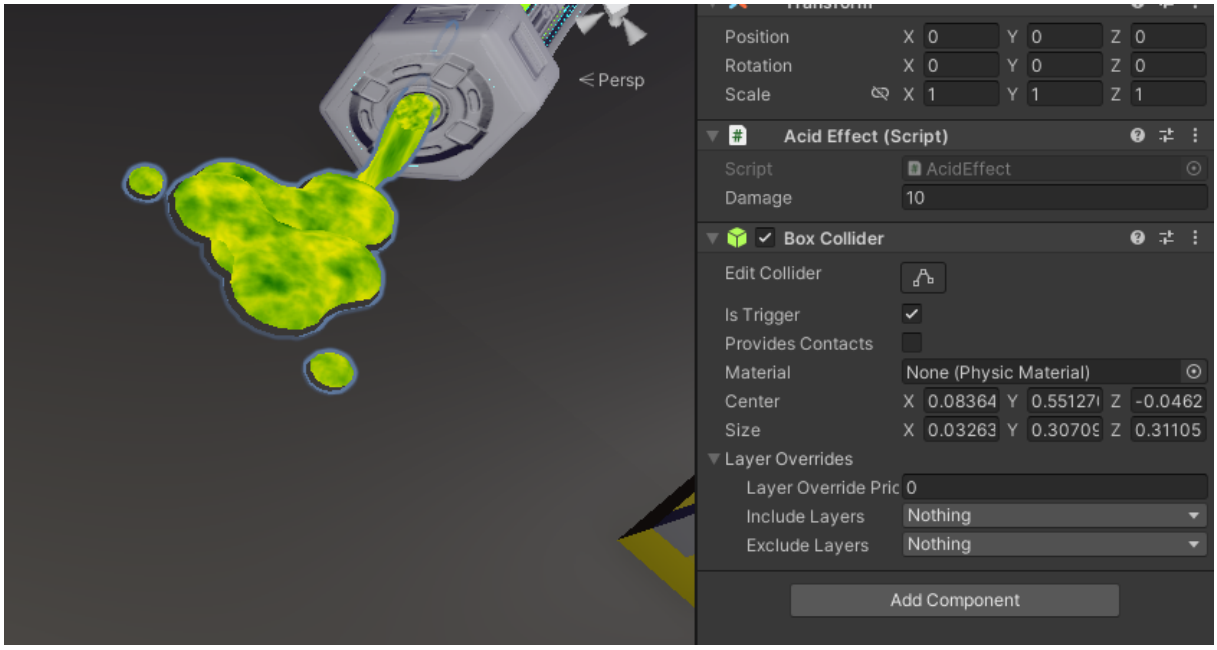


Ilustración 3.86; Primera planta, componentes del ácido de los tanques de ácido

3.3.4.3.2.2.3.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el tanque de ácido.

Test_04_24	
Nombre de la prueba o test	Daño producido al jugador
Objetivo	El ácido del tanque debe infligir daño al jugador
Resultado	El ácido infringe daño al jugador cuando este contacta con el ácido.
Comentarios	El jugador puede no percibir que entrar en contacto con el ácido le producirá daño.

Tabla 3.57; Test_04_24: daño producido al jugador

Test_04_25

Nombre de la prueba o test	Ruta para llegar al ácido
Objetivo	Debe existir al menos una forma de que el jugador llegue al ácido
Resultado	El jugador puede tocar sin problema el ácido.
Comentarios	Todo OK.

Tabla 3.58; Test_04_25: ruta para llegar al ácido

3.2.4.3.3 Segunda planta

En esta segunda planta de la nave espacial, se encontrará a Freddy, otro NPC con el que se podrá hablar y mantener una conversación con él. Aunque más adelante se detallará su funcionamiento, su función en el juego es ayudar al jugador a avanzar en el juego.

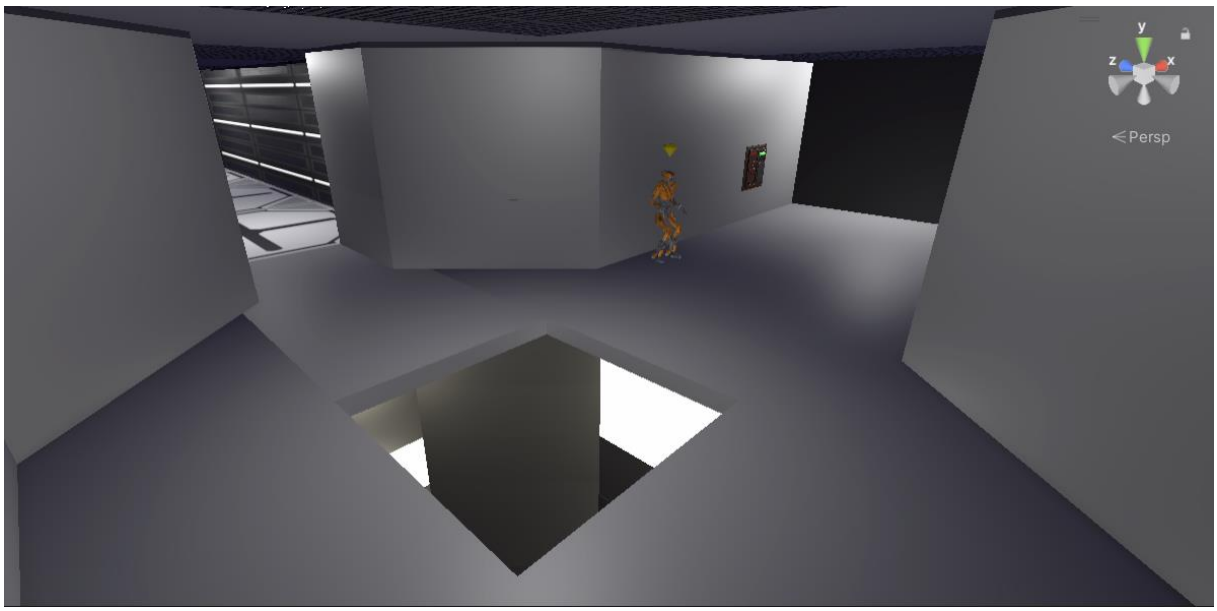


Ilustración 3.87; Segunda planta, sala intermedia

En esta planta, nos podremos encontrar varias salas: sala izquierda, sala derecha, sala lateral, sala de StarLord y sala intermedia.

3.2.4.3.3.1 Sala izquierda

Empezando por la sala izquierda, esta es una de las salas de enemigos que se encuentran en el juego.

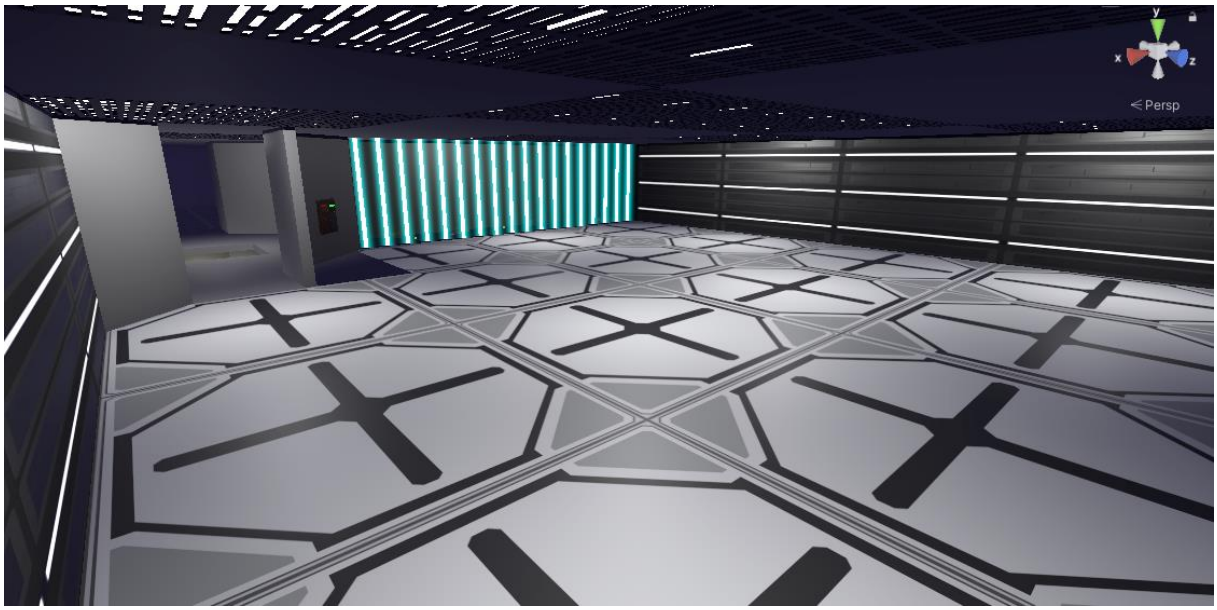


Ilustración 3.88; Segunda planta, sala izquierda

Esta sala tendrá un panel de control, el cual tendrá una de las baterías necesarias para entrar a la sala de StarLord.

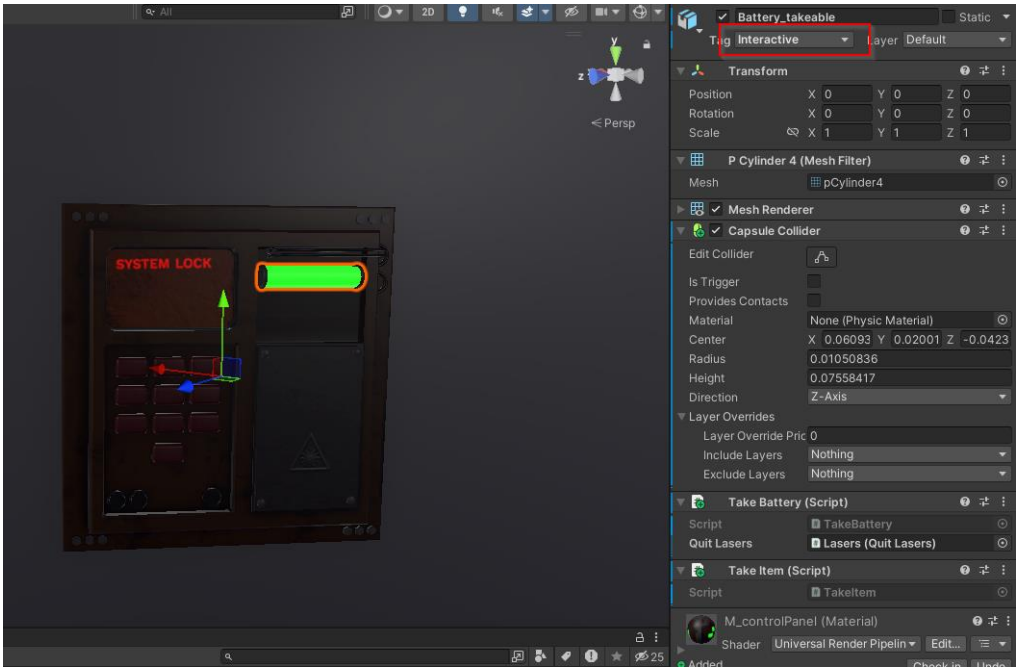


Ilustración 3.89; Segunda planta, sala izquierda, batería recogible

Esta batería, como se puede ver, tiene la etiqueta “Interactive”, por lo que es interactuable. Al interactuar con la batería, se activarán tanto el script “TakeItem”, el cual llamará a la función “TakeItem()” del GameManager, como el script “TakeBattery”, el cual activa el script “QuitLasers” asociado a los láseres que protegen la recompensa o el alijo de la habitación.

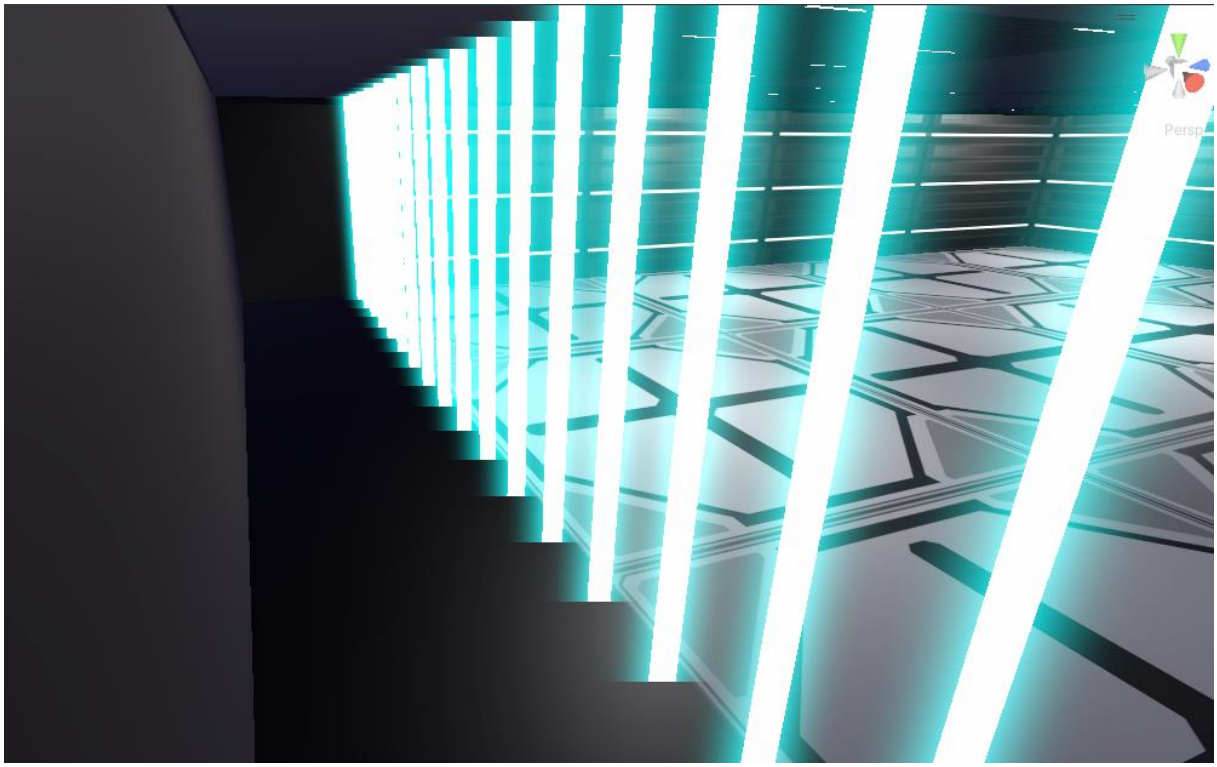


Ilustración 3.90; Segunda planta, sala izquierda, sección de recompensas

En esta sección de la habitación donde se encuentra el alijo, se podrán encontrar unos objetos en forma de láseres. Estos láseres son unos objetos aportados por un asset llamado “VolumetricLines” [77], sacado de la “AssetStore” de Unity [78], los cuales tienen un script asociado llamado “VolumetricLineBehaviour”, el cual controla la forma y color del láser.

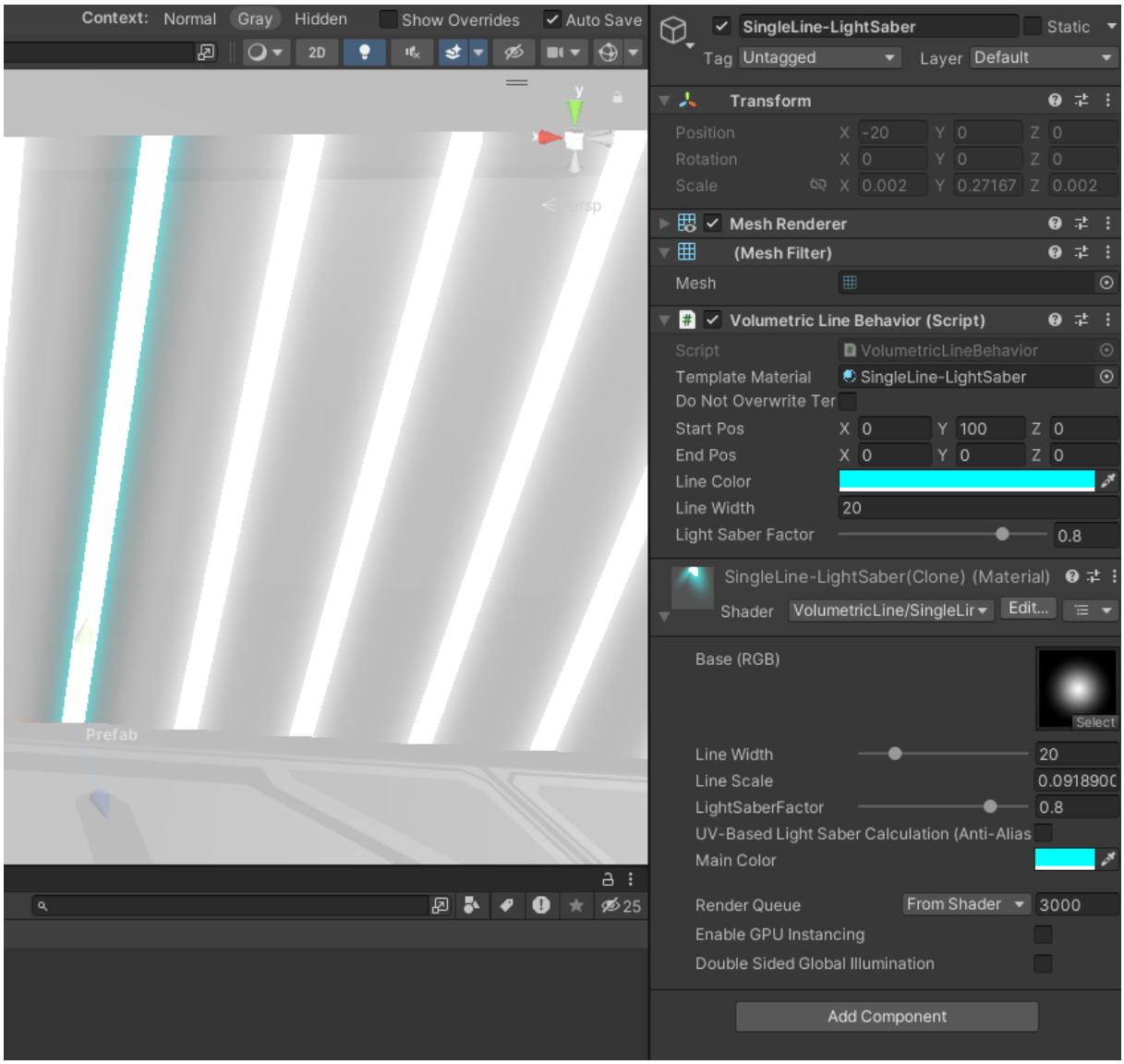


Ilustración 3.91; Segunda planta, láser de la sección de recompensa

La forma en la que funciona el script “QuitLasers” es que, mediante una corrutina y la función “Lerp” para cada láser, se va aumentando poco a poco el parámetro “Light saber factor”, el cual controla “lo blanco” del láser. Al aumentar ese parámetro, “lo blanco” va desapareciendo, dando la sensación de que se está apagando el láser y así poder indicarle al jugador que puede coger el alijo.

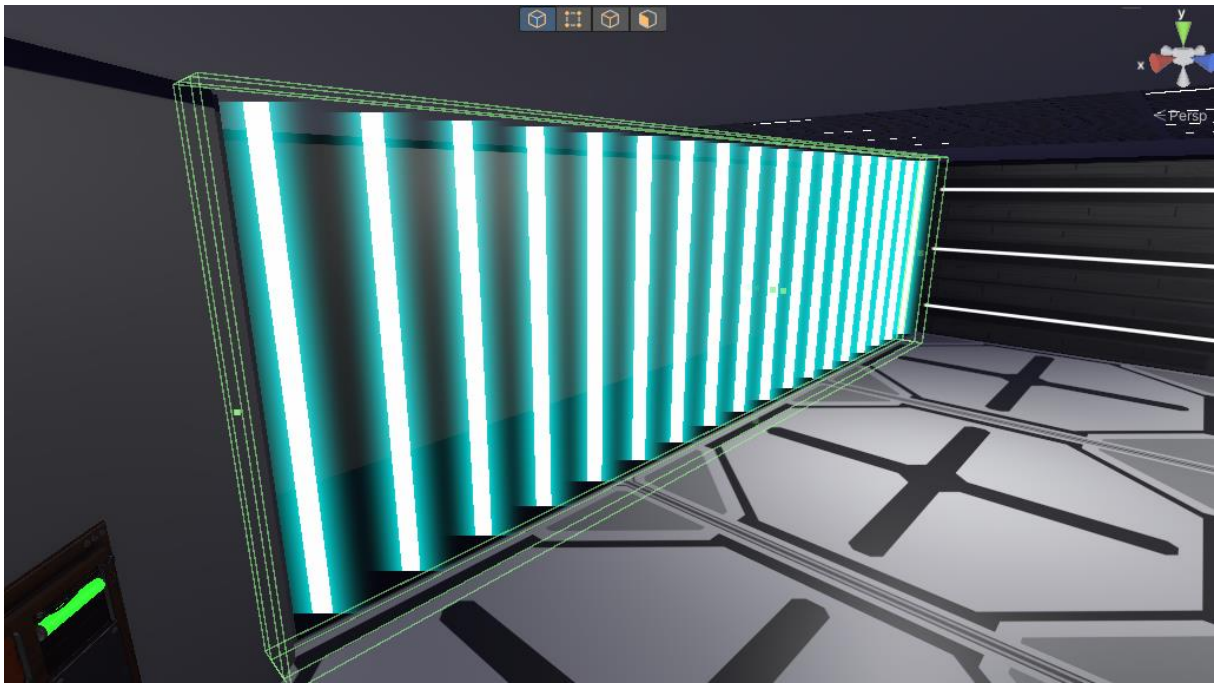


Ilustración 3.92; Segunda planta, collider de la sala de recompensa

También comentar que existe un “BoxCollider” que inhabilita al jugador poder entrar a esta sección ni coger ítems desde fuera, el cual desaparece al terminar todas las corrutinas de todos los láseres, es decir, al “apagarse” todos los láseres.

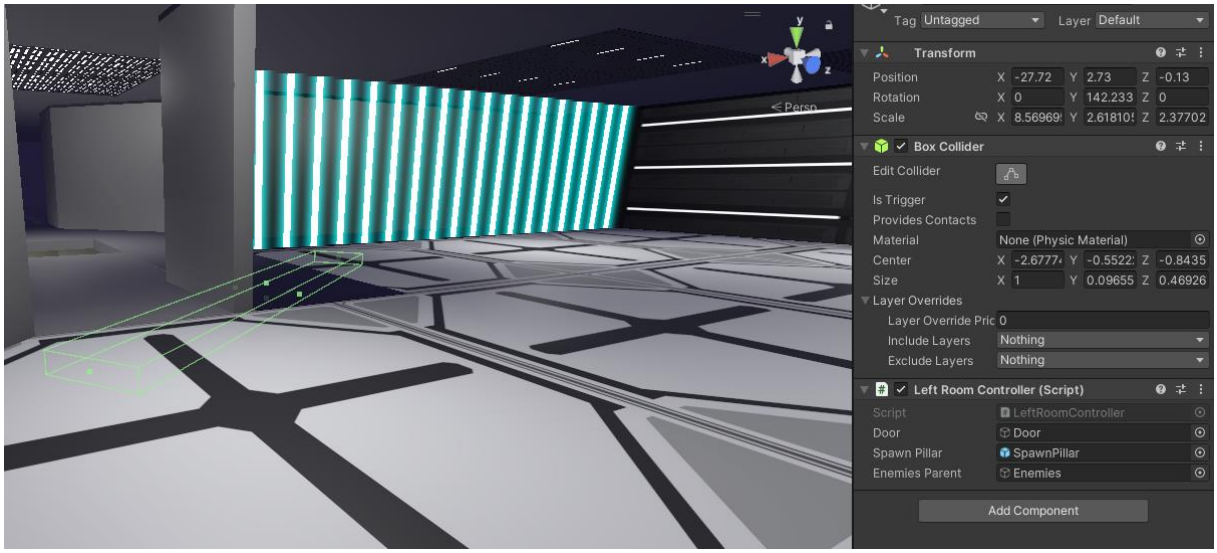


Ilustración 3.93; Segunda planta, collider de la sala izquierda

Por último, destacar que esta habitación tiene un componente “BoxCollider” con el cual se puede controlar si el jugador entra en la sala. Esta sala, además tiene asociado un script llamado “LeftRoomController”, el cual al entrar el contacto el

jugador con el “BoxCollider” y, por tanto, entrar en la sala, activa un pilar “spawn” de enemigos y controla los enemigos activos en la sala. Además de activar el pilar, activa otra pared invisible igual que la de la primera planta para que el jugador no pueda salir de la habitación hasta que no haya acabado con todos los enemigos de la sala, que será cuando el mismo script desactive esa pared.

3.6.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la sala izquierda de la segunda planta

Test_04_26	
Nombre de la prueba o test	Límite de recogida de recompensas
Objetivo	El jugador no debe poder recoger recompensas de la sección de recompensas desde fuera de la misma si están activos los láseres
Resultado	El jugador no puede recoger recompensas de la sección de recompensas desde fuera de la misma si están activos los láseres, necesita desactivar los láseres para poder recogerlas
Comentarios	Todo OK.

Tabla 3.59; Test_04_26: límite de recogida de recompensas

Test_04_27	
Nombre de la prueba o test	Momento de desactivación de los láseres
Objetivo	Los láseres deben empezar a desactivarse justo cuando el jugador coge la batería
Resultado	Justo cuando el jugador coge la batería del panel de control, se empiezan a desactivar los láseres.
Comentarios	El jugador podría no darse cuenta de que se están desactivando los láseres

Tabla 3.60; Test_04_27: momento de desactivación de los láseres

Test_04_28	
Nombre de la prueba o test	Velocidad de desactivación de los láseres
Objetivo	Los láseres de la sala deben desactivarse a una velocidad adecuada
Resultado	Los láseres se desactivan gradualmente y a una velocidad adecuada
Comentarios	Dependiendo de la perspectiva del jugador, la velocidad puede ser lenta.

Tabla 3.61; Test_04_28: velocidad de desactivación de los láseres

Test_04_29	
Nombre de la prueba o test	Cumplir tarea antes de salir de la sala
Objetivo	El muro invisible de la sala debe permanecer activo hasta que el jugador complete su tarea
Resultado	El muro invisible de la sala no desaparece hasta que el jugador no elimina a todos los enemigos de la sala
Comentarios	El jugador puede no saber qué tiene que hacer en ese momento para hacer desaparecer el muro invisible

Tabla 3.62; Test_04_29: cumplir tarea antes de salir de la sala

Test_04_30	
Nombre de la prueba o test	Recoger batería del panel de control
Objetivo	El jugador debe poder recoger la batería del panel de control
Resultado	El jugador puede recoger la batería del panel de control sin ningún problema
Comentarios	Todo OK.

Tabla 3.63; Test_04_30: recoger batería del panel de control

Test_04_31	
Nombre de la prueba o test	Recogida de las recompensas

Objetivo	El jugador debe poder recoger las compensas
Resultado	El jugador puede recoger las recompensas de la sección al coger la batería del panel
Comentarios	Todo OK.

Tabla 3.64; Test_04_31: recogida de las recompensas

3.2.4.3.3.2 Sala derecha

La sala de la derecha es prácticamente idéntica a la sala de la izquierda, por lo que se obvia su explicación. La única diferencia es que esta sala carece de iluminación, por lo que será recomendable que el jugador encienda la linterna de su arma para poder ver la sala con mayor facilidad.

3.2.4.3.3.3 Sala lateral

En cuanto a la sala lateral es una sala en la que podremos encontrar el alijo secreto de StarLord.

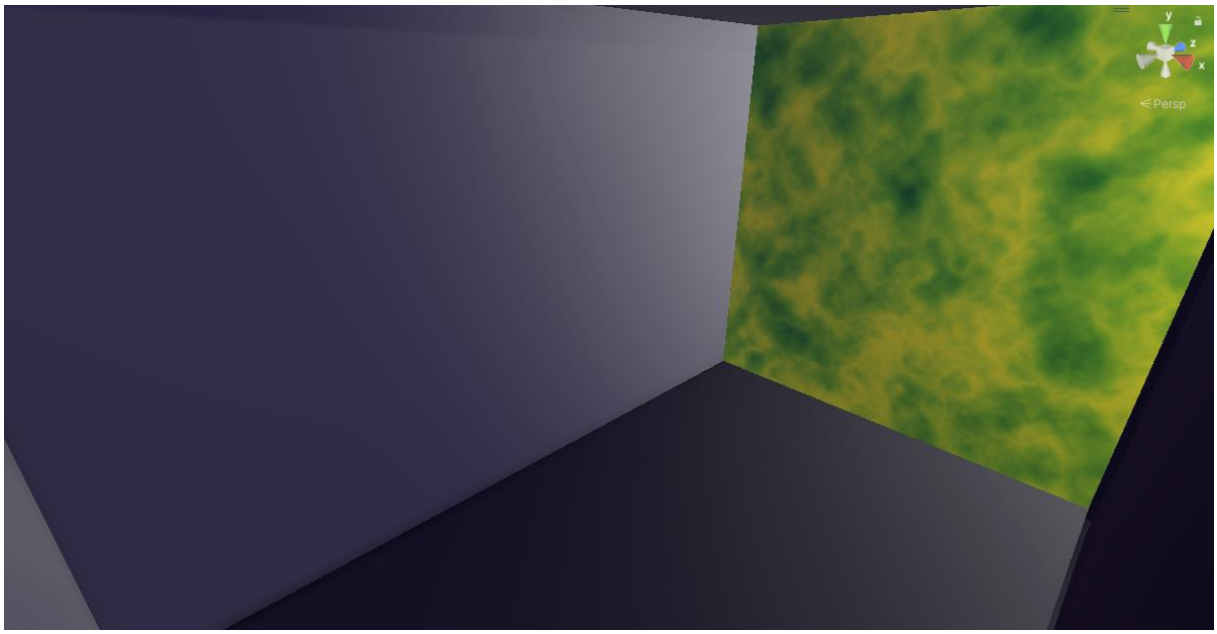


Ilustración 3.94; Segunda planta, sala lateral

Lo más relevante por comentar de esta sala es su cascada de ácido.

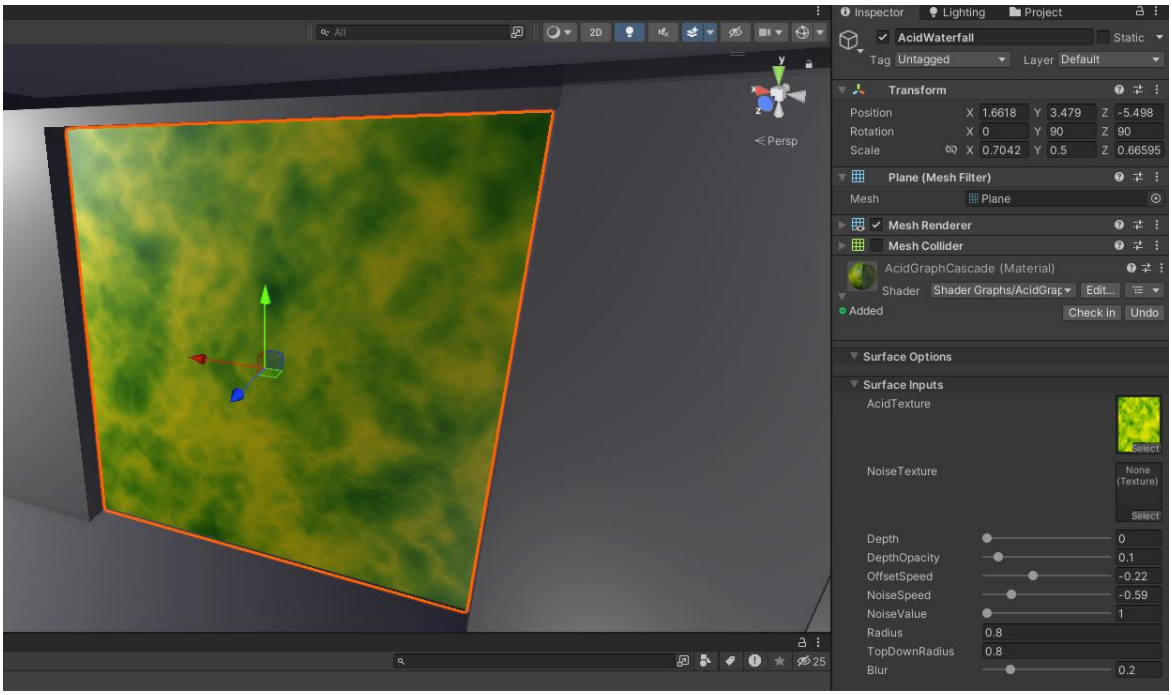


Ilustración 3.95; Segunda planta, cascada de ácido

Esta es una cascada de ácido especial puesto que tiene un material asociado el cual está basado en un shader especial. Este shader especial lo que hace es que, en función de la proximidad del jugador con la cascada, esta se va abriendo para dar paso al jugador.

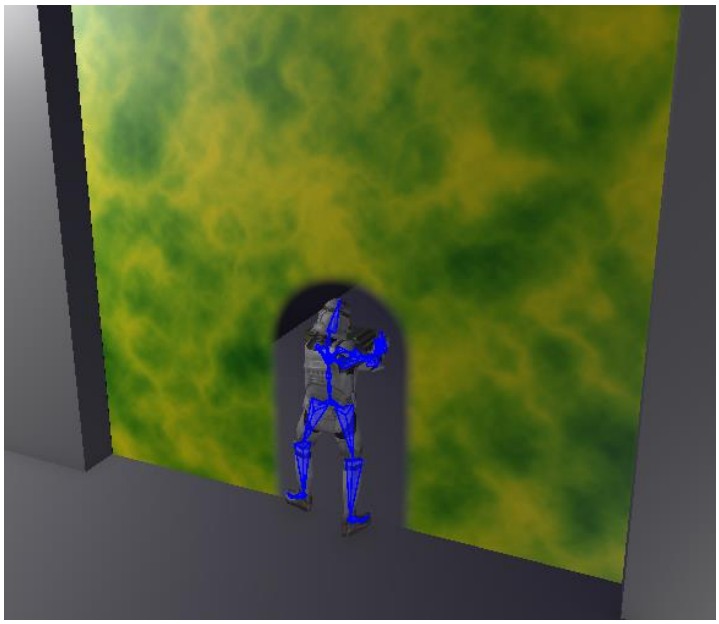


Ilustración 3.96; Segunda planta, efecto de la cascada de ácido

Para conseguir este efecto, se ha usado la potente herramienta que nos aporta Unity y que sirve para crear shaders de forma más simple que con lenguaje HLSL. Esta herramienta es Shader Graph, la cual está detallada en el **apartado 3.3.2 Shader Graph.**

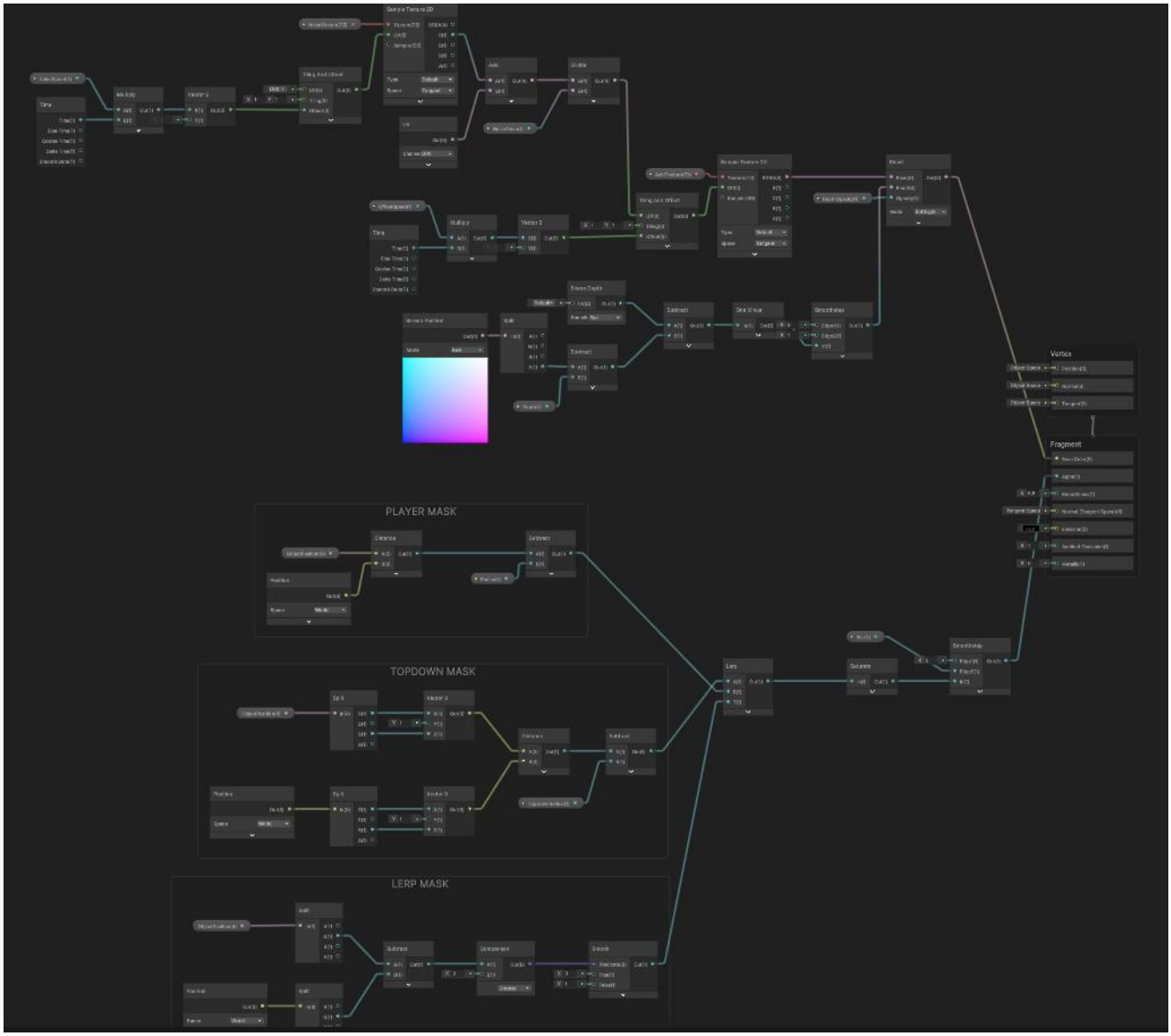


Ilustración 3.97; Segunda planta, shader graph completo de la cascada de ácido

Para explicar este shader, vamos a desglosarlo puesto que es bastante complejo. Primero, veamos la textura:

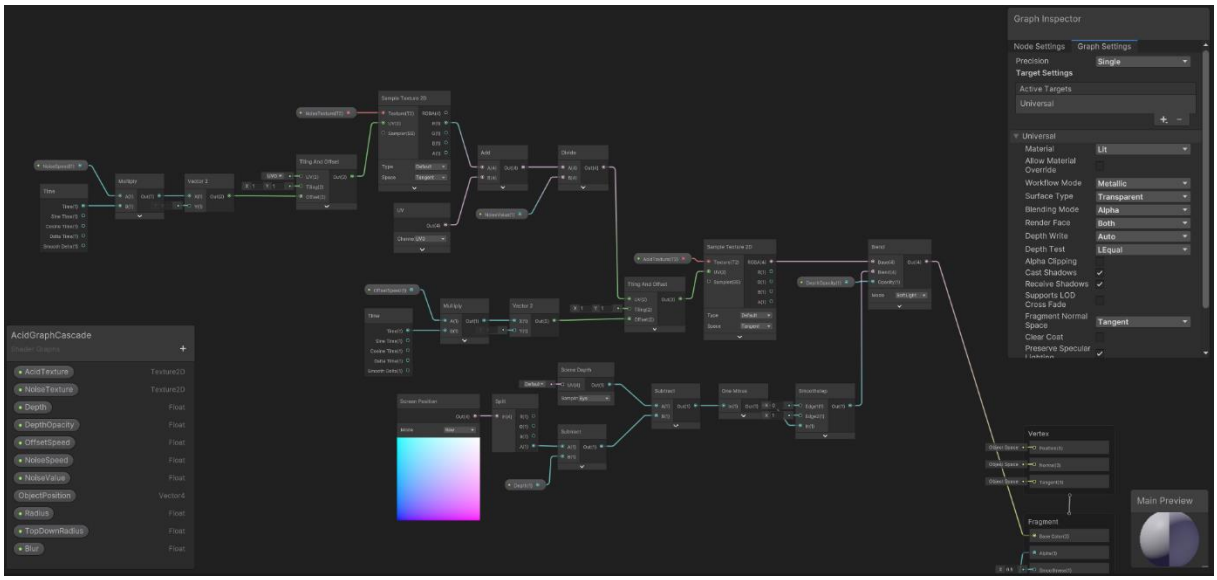


Ilustración 3.98; Segunda planta, primera parte del shader graph de la cascada de ácido

Para conseguir la textura, se ha usado el nodo “SampleTexture2D” [79], con el que se ha cargado la textura. Al componente “UV” del nodo “SampleTexture2D” se le ha incluido unos valores transformados y regidos por el parámetro “OffsetSpeed” para variar el movimiento de la textura. Textura a la cual se le aplica un ruido basado en una textura pasada por el parámetro “NoiseTexture” y determinada por el parámetro “NoiseSpeed” el cual varía la velocidad de movimiento de la textura del ruido.

Además, tenemos las máscaras aplicadas al shader. Estas, tienen un efecto en cómo se ve la pared en función de algunas ”condiciones”, como la distancia entre la cámara y la pared.

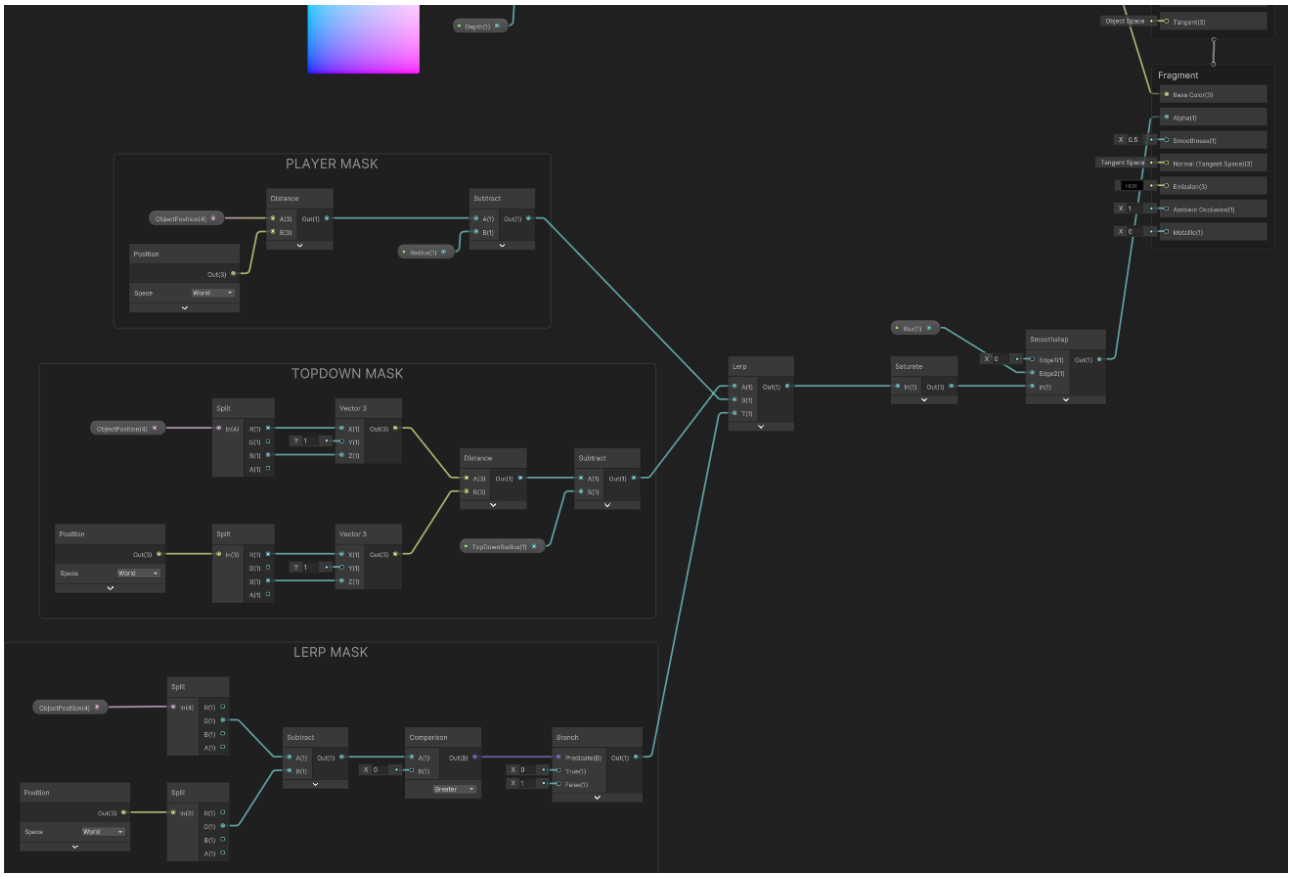


Ilustración 3.99; Segunda planta, segunda parte del shader graph de la cascada de ácido

La primera de las máscaras [80] es la “PlayerMask”, la cual se utiliza para generar una máscara basada en la distancia entre la posición del objeto y la posición del jugador. Esta máscara crea un área basada en lo cerca que esté el jugador de la pared, permitiendo que los efectos que se apliquen, se intensifiquen o se atenúen dependiendo de esta distancia.

Otra de las máscaras del shader es la “TopDownMask”. Esta máscara genera una máscara desde una vista superior, la cual crea un área circular en la que se permite aplicar efectos dentro del radio de esa área. Los efectos que se aplican son los producidos por la “PlayerMask”.

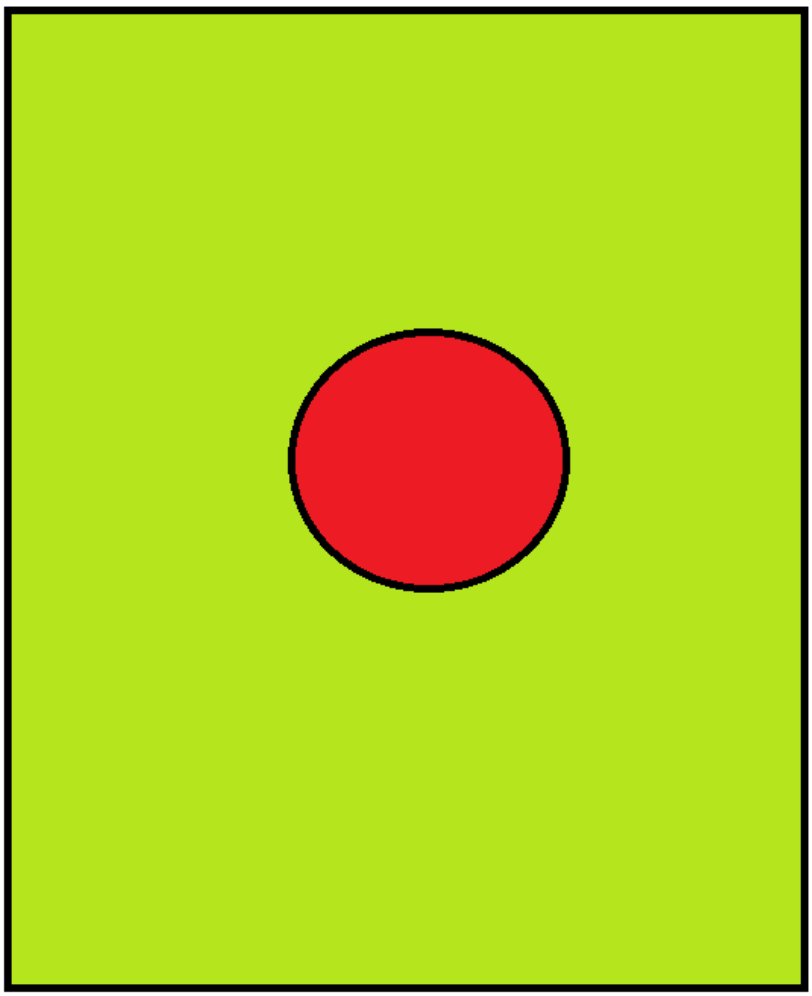


Ilustración 3.100; Segunda planta, primera parte explicación del efecto de la cascada de ácido

La última máscara es la “LerpMask”, con la cual se interpola entre el área circular creada por la máscara “TopDownMask” y un camino rectangular desde el centro de esa área circular hasta el suelo.

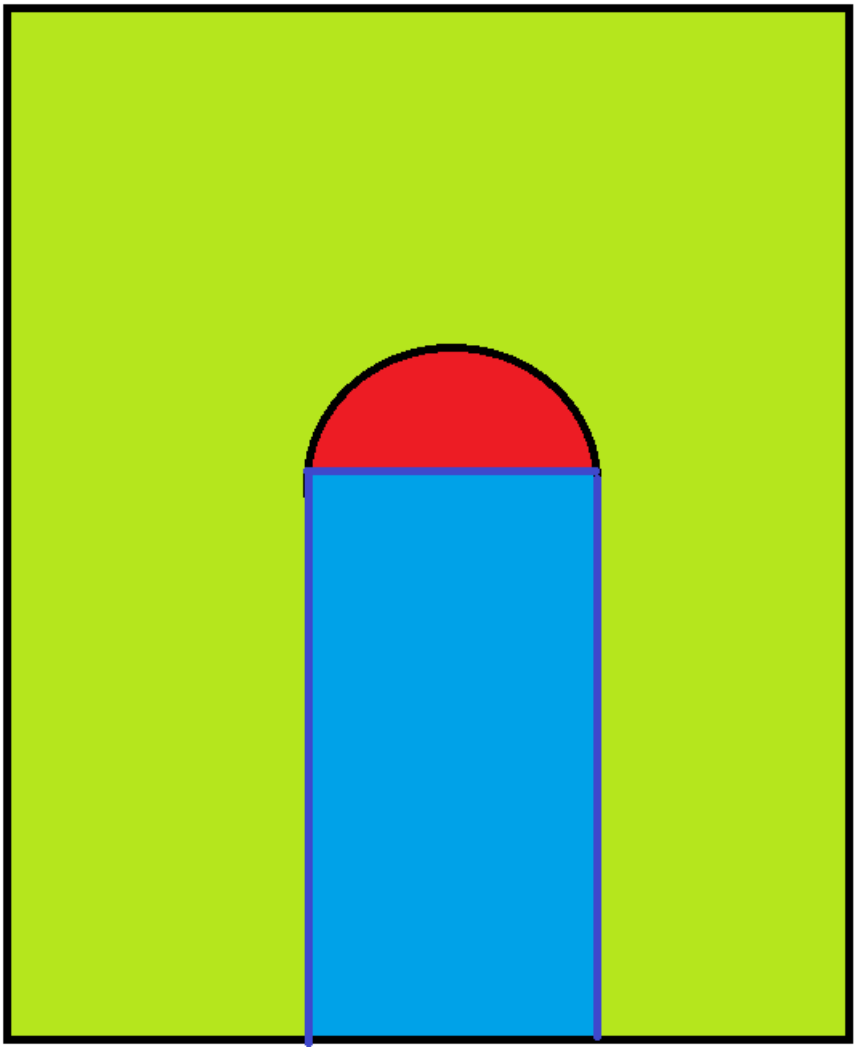


Ilustración 3.101; Segunda planta, segunda parte explicación del efecto de la cascada de ácido

Con estas máscaras, se da la sensación de “cascada”. Es decir, el jugador entra y todo lo que esté debajo del punto de contacto más arriba con la cascada, la cabeza del personaje, se abre como simulando que la cabeza impide que el ácido de la cascada siga bajando.

3.8.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la sala lateral de la segunda planta

Test_04_32	
Nombre de la prueba o test	Apertura de cascada de ácido

Objetivo	La cascada de ácido debe poder abrirse conforme el jugador entra en ella.
Resultado	La cascada se abre poco a poco a medida que el jugador entra en ella sin dañarse.
Comentarios	Todo OK.

Tabla 3.65; Test_04_32: apertura de cascada de ácido

3.2.4.3.3.4 Sala de StarLord

Antes de entrar a la sala de StarLord, en el pasillo se encontrará la puerta de la sala, la cual tendremos que abrir colocando, en el panel de control próximo a la puerta, las baterías encontradas en las habitaciones anteriores.

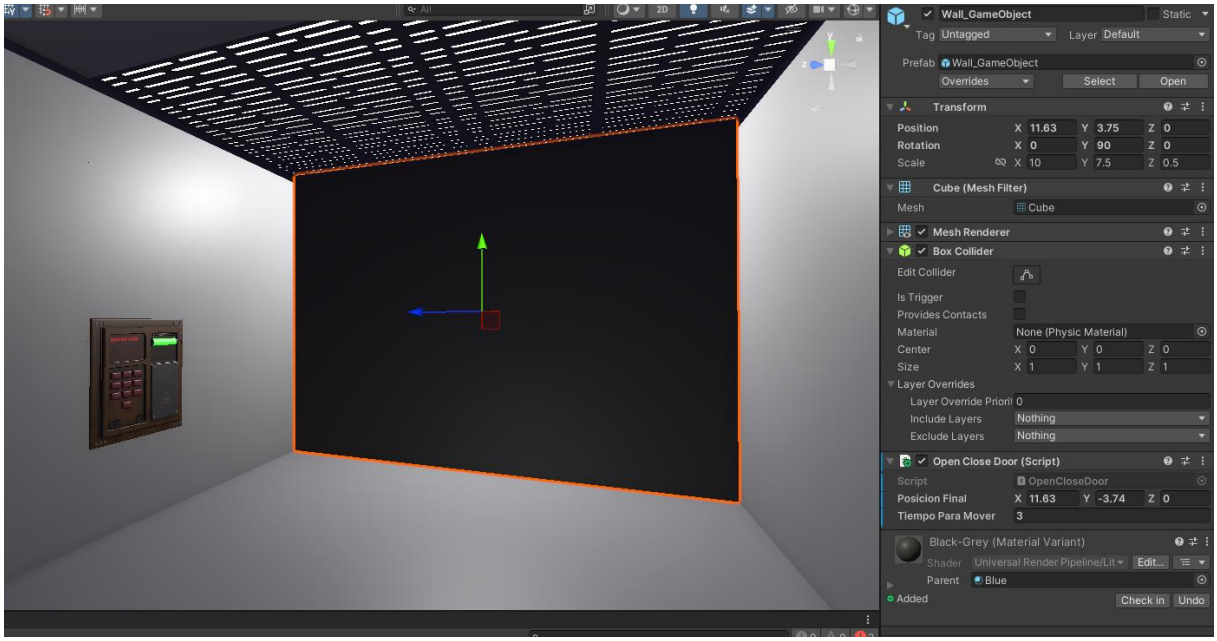


Ilustración 3.102; Segunda planta, puerta de la sala de StarLord

Esta puerta tiene un script asociado llamado “OpenCloseDoor”, el cual desplazará la puerta desde un punto A (posición inicial) hasta un punto B (posición “abierta”) usando la función “Lerp”. Durante el movimiento de la puerta, se llamará a la corrutina “Shake()” del script “CameraShake” de la cámara principal, con la que se conseguirá el efecto de temblor en la cámara al mover la puerta.

Las baterías, como se ha comentado, se deben colocar en el panel de control para abrir la puerta.

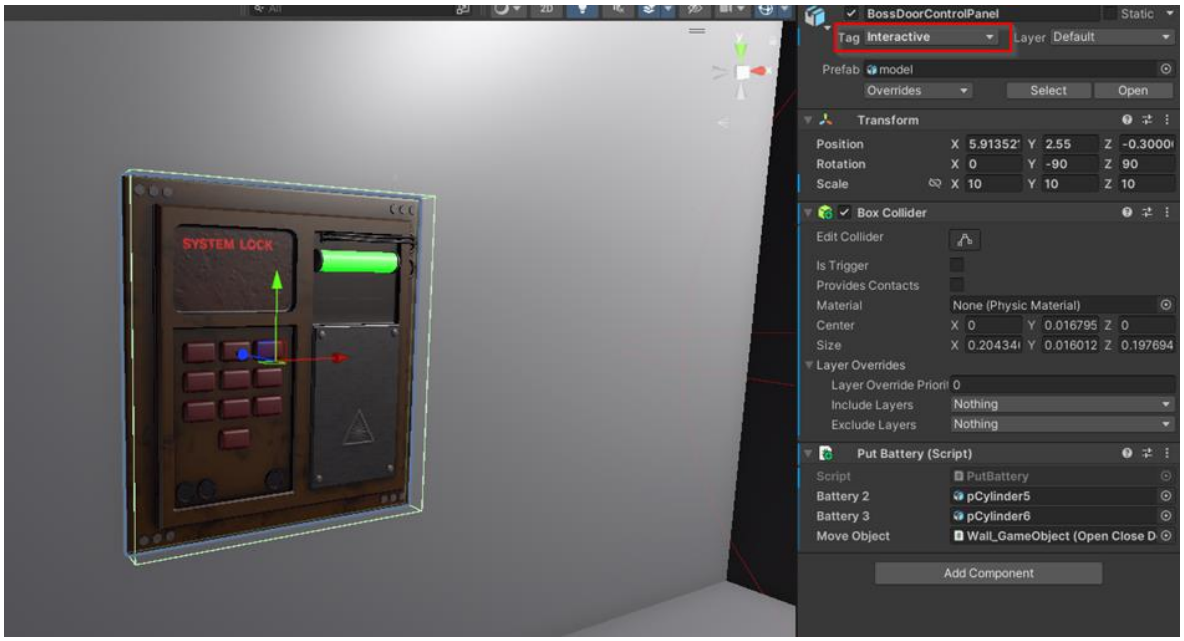


Ilustración 3.103; Segunda planta, panel de control de la puerta de la sala de StarLord

Este panel de control, como se puede ver, tiene la etiqueta “Interactive”, por lo que se puede interactuar con él. Cabe destacar que, para interactuar con el panel de control, es indispensable tener seleccionada una batería, si no, no se podrá interactuar con el panel de control. Esto es conseguido validando el nombre del objeto seleccionado al intentar interactuar con el panel de control. El resultado de interactuar con él es la activación del script asociado al objeto, el cual es el llamado “PutBattery”.

Ilustración 3.104; Segunda planta, no se puede interactuar con panel si jugador no tiene batería seleccionada

Ilustración 3.105; Segunda planta, se puede interactuar con panel si jugador tiene batería seleccionada

El panel de control, inicialmente, tiene una batería activa y dos desactivadas, como se puede ver en la siguiente imagen.

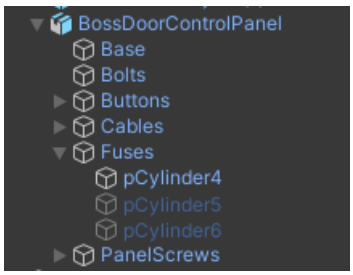


Ilustración 3.106; Segunda planta, elementos del panel de control de la puerta de la sala de StarLord

El script “PutBattery”, lo que hace es que, en función de cuántas veces se haya activado el script, se activa la segunda batería o la tercera, para así dar la sensación de que se está colocando la batería recogida. Cuando se active el script dos veces y se coloque la tercera batería, este script llamará al script “OpenCloseDoor” de la puerta de StarLord.

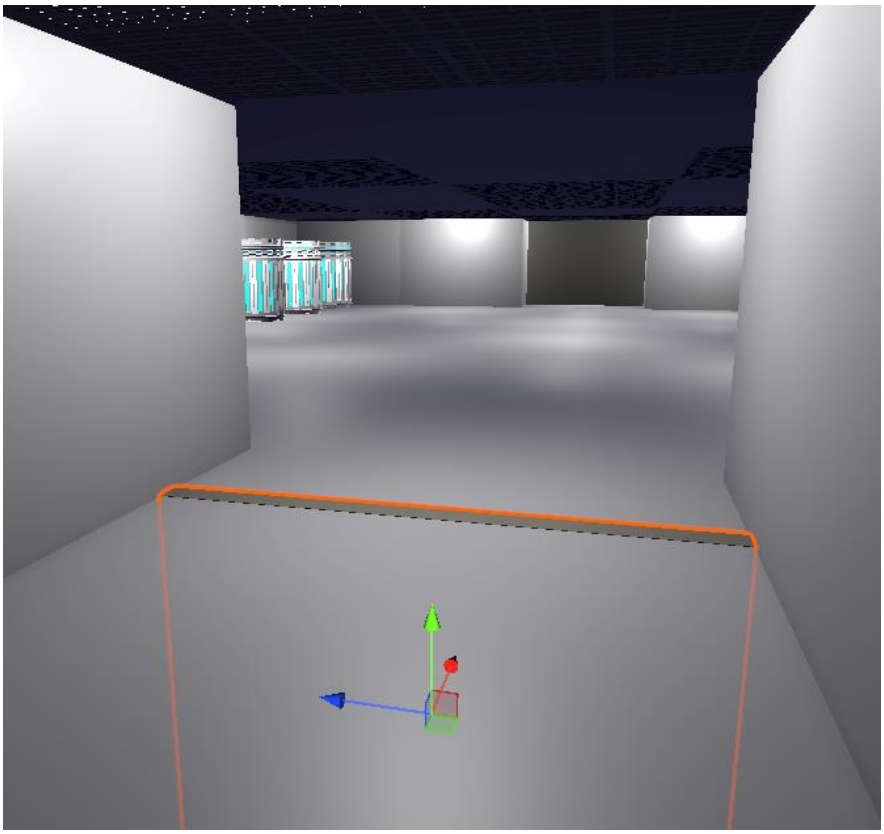


Ilustración 3.107; Segunda planta, puerta de la entrada de la sala de StarLord abierta

Una vez se abra la puerta de la habitación de StarLord, podremos acceder a la sala.

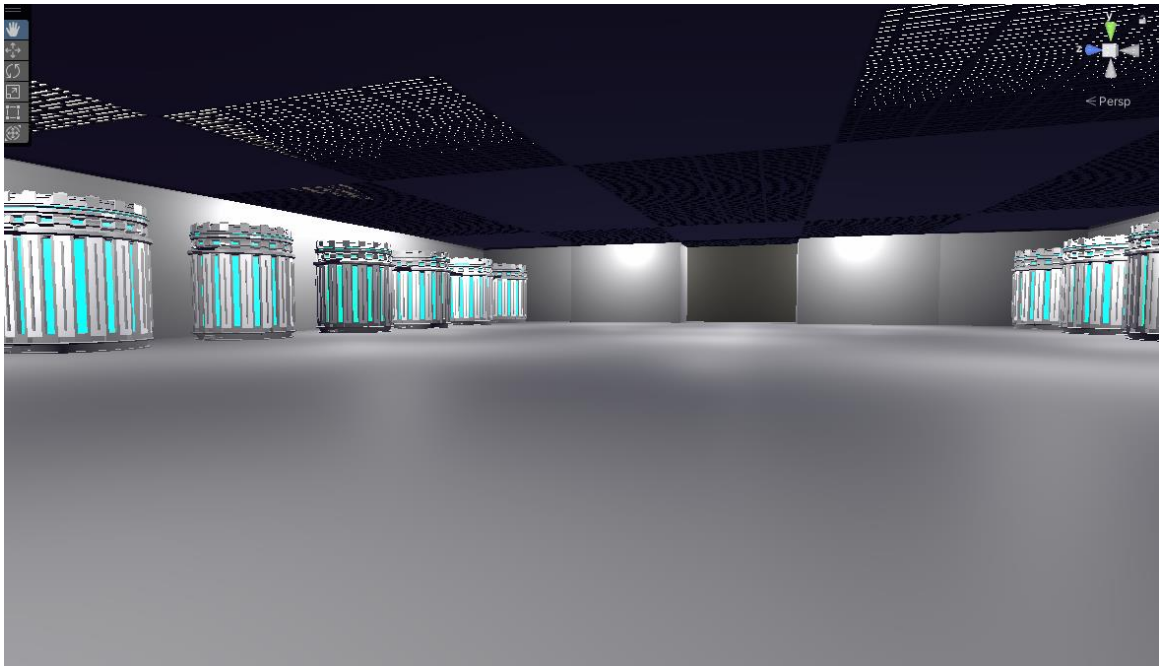


Ilustración 3.108; Segunda planta, sala de StarLord

Esta sala tendrá asignado un componente “BoxCollider” con la opción “isTrigger” marcada para detectar si algún objeto entra en contacto con el componente.

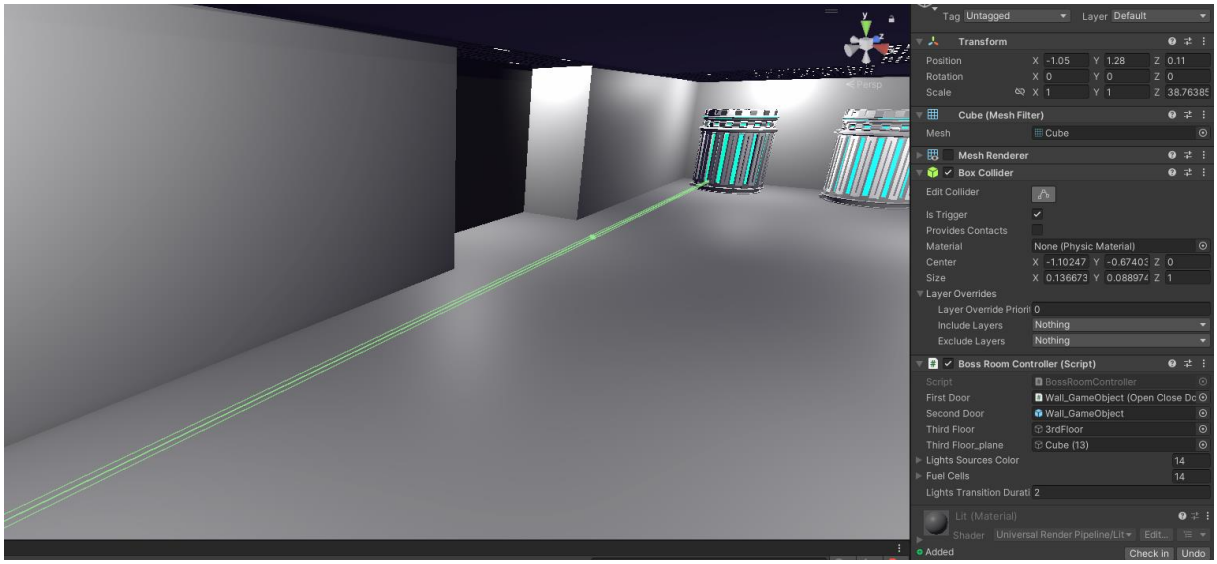


Ilustración 3.109; Segunda planta, collider de la sala de StarLord

Además, la sala tendrá asociado el script “BossRoomController”, con el que se controlará todo lo que ocurra en la habitación del jefe final.

Una de las cosas que ocurren en la sala del jefe final, es que, al entrar en contacto el jugador con el componente “BoxCollider” de la sala, se volverá activar el script “OpenCloseDoor” de la puerta principal de la sala. Esto lo que causará es que la puerta principal vuelva de la posición B a la posición A usando la función “Lerp”.

Cuando la puerta haya alcanzado el punto inicial, y por tanto, esté cerrada la puerta, se iniciarán dos corrutinas. La primera, llamada “ChangeLightsColor()”, disminuirá la intensidad de todas las luces de la habitación y las cambiará a color rojo. Con esto, se dará un ambiente de peligro y estado de emergencia, lo que pondrá al usuario en estado de alarma. La segunda corrutina será la llamada “ChangeFuelCellsMaterial()”, con la que se cambiarán las luces internas de las células de energía que se encontrarán a los laterales de la sala.

Ilustración 3.110; Segunda planta, sala de StarLord después de que jugador pase por el collider de la sala.

Cuando se termine el encuentro con el jefe final, el script “BossRoomController” detectará que no hay enemigos restantes y, tras dos segundos, llamará al script “OpenCloseDoor”, asociado a la puerta del fondo, el cual abrirá la puerta del fondo.

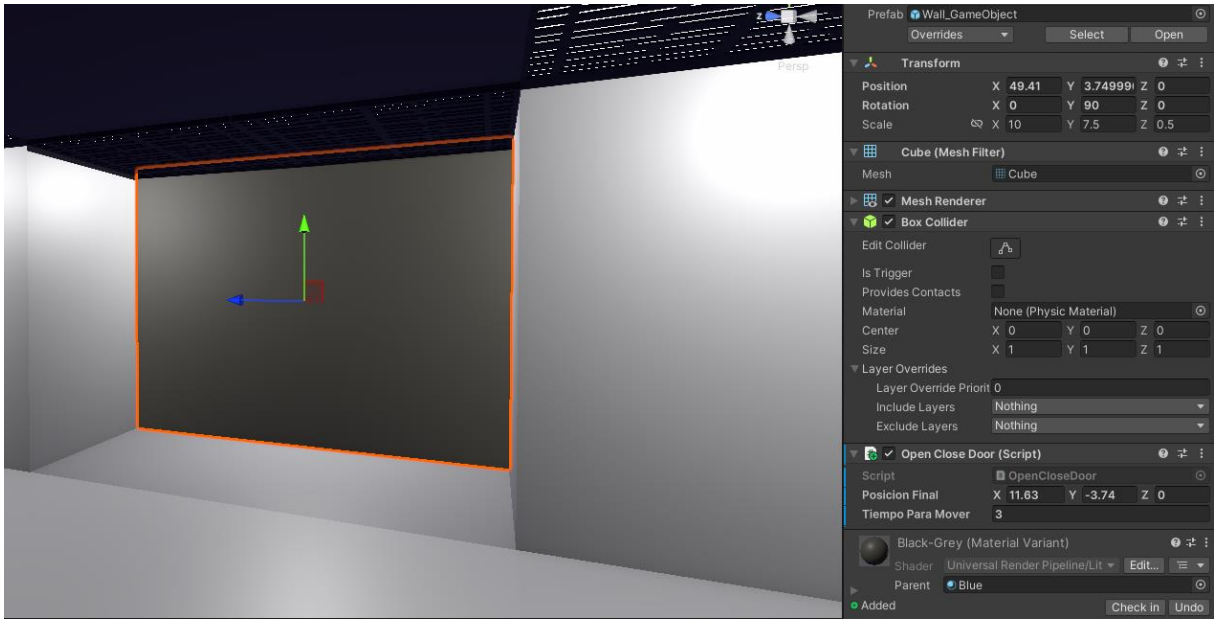


Ilustración 3.111; Segunda planta, puerta de salida de la sala de StarLord

Cuando se abra la puerta del fondo, se activará la tercera planta, inicialmente desactivada. El motivo por el cual está desactivada inicialmente, no es otro que el de reducir los recursos que el sistema debe cargar y que el juego vaya más fluido, ya que hacemos que el sistema se ahorre cargar los recursos de la tercera planta durante todo el juego y solo los cargue cuando estos sean necesarios, es decir, cuando se derrote al jefe final y el jugador se dirija al final del juego.

3.10.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la sala de StarLord de la segunda planta

Test_04_33	
Nombre de la prueba o test	Colocación de la batería
Objetivo	La batería solo puede ser colocada por el jugador si se cumplen las condiciones
Resultado	La batería solo se coloca si el jugador tiene seleccionada la batería y tiene baterías suficientes
Comentarios	El jugador puede no saber qué tiene que hacer para colocar las baterías

Tabla 3.66; Test_04_33: colocación de la batería

Test_04_34	
Nombre de la prueba o test	Aparición de las baterías colocadas
Objetivo	Debe aparecer la batería que se ha colocado
Resultado	La batería colocada por el jugador aparece al ser está colocada
Comentarios	Todo OK.

Tabla 3.67; Test_04_34: aparición de las baterías colocadas

Test_04_35	
Nombre de la prueba o test	Cantidad de vibración de la cámara principal

Objetivo	La cámara debe vibrar una cantidad adecuada
Resultado	La cámara del jugador vibra una cantidad adecuada, sin llegar a molestar al jugador y dando a entender que algo pesado se está moviendo
Comentarios	Todo OK.

Tabla 3.68; Test_04_35: cantidad de vibración de la cámara principal

Test_04_36	
Nombre de la prueba o test	Momento de vibración de la cámara principal
Objetivo	La cámara del jugador debe ponerse a vibrar en el momento adecuado
Resultado	La cámara del jugador empieza a vibrar al colocar todas las baterías en el panel de control.
Comentarios	El jugador puede interpretar que hay algo peligroso acechándole.

Tabla 3.69; Test_04_36: momento de vibración de la cámara principal

Test_04_37	
Nombre de la prueba o test	Tiempo de vibración de la cámara principal
Objetivo	La cámara del jugador debe vibrar una cantidad de tiempo adecuada
Resultado	La cámara del jugador vibra el tiempo que la puerta está moviéndose.
Comentarios	Todo OK.

Tabla 3.70; Test_04_37: tiempo de vibración de la cámara principal

Test_04_38	
Nombre de la prueba o test	Momento de apertura de la puerta de la sala de StarLord
Objetivo	La puerta de la sala de StarLord debe empezar a moverse en el momento indicado
Resultado	La puerta de la sala de StarLord se mueve justo al colocar la tercera batería

Comentarios	Todo OK.

Tabla 3.71; Test_04_38: momento de apertura de la puerta de StarLord

Test_04_39	
Nombre de la prueba o test	Velocidad de apertura de la puerta de la sala de StarLord
Objetivo	La puerta de la sala de StarLord debe moverse a una velocidad adecuada
Resultado	La puerta de la sala de StarLord se mueve a una velocidad con la que el jugador pueda darse cuenta de qué está pasando, pero sin llegar a ser pesada.
Comentarios	El jugador puede no centrarse en el movimiento de la puerta y mirar hacia la puerta una vez que esta esté abierta, sin llegar a saber qué ha pasado.

Tabla 3.72; Test_04_39: velocidad de apertura de la puerta de la sala de StarLord

Test_04_40	
Nombre de la prueba o test	Momento de cierre de la puerta de la sala de StarLord
Objetivo	La puerta de la sala de StarLord debe empezar a cerrarse en el momento indicado
Resultado	La puerta de la sala de StarLord se cierra justo al pasar el jugador por el componente “Collider” del controlador de la sala
Comentarios	Al cerrarse la puerta a la espalda del jugador, este puede no saber lo que está pasando.

Tabla 3.73; Test_04_40: momento de cierre de la puerta de la sala de StarLord

Test_04_41	
Nombre de la prueba o test	Velocidad de cierre de la puerta
Objetivo	La puerta de la sala de StarLord debe cerrarse a una velocidad adecuada
Resultado	La puerta de la sala de StarLord se cierra a una velocidad con la que el jugador pueda darse cuenta de qué está pasando, pero sin llegar a ser pesada.

Comentarios	El jugador puede no centrarse en el movimiento de la puerta y mirar hacia la puerta una vez que esta esté cerrada, sin llegar a saber qué ha pasado.

Tabla 3.74; Test_04_41: velocidad de cierre de la puerta de la sala de StarLord

Test_04_42	
Nombre de la prueba o test	Momento de cambio de las luces de la sala de StarLord
Objetivo	Las luces de la sala de StarLord deben cambiar en el momento adecuado
Resultado	Las luces de la sala de StarLord cambian justo cuando la puerta de la sala se cierra
Comentarios	El jugador, si no ha visto cerrar la puerta, puede no saber por qué han cambiado las luces

Tabla 3.75; Test_04_42: momento de cambio de las luces de la sala de StarLord

Test_04_43	
Nombre de la prueba o test	Iluminación de la sala de StarLord tras el cambio de luces
Objetivo	La sala de StarLord debe estar correctamente iluminada tras haber cambiado las luces de color
Resultado	La sala tiene una iluminación visualmente correcta tras el cambio de luces.
Comentarios	Todo OK.

Tabla 3.76; Test_04_43: iluminación de la sala de StarLord tras el cambio de luces

3.2.4.3.3.5 Sala intermedia

Por último, tendremos la sala que conecta la segunda planta con la tercera. Esta sala no tendrá ninguna funcionalidad, por lo que su único propósito es conectar ambas plantas.

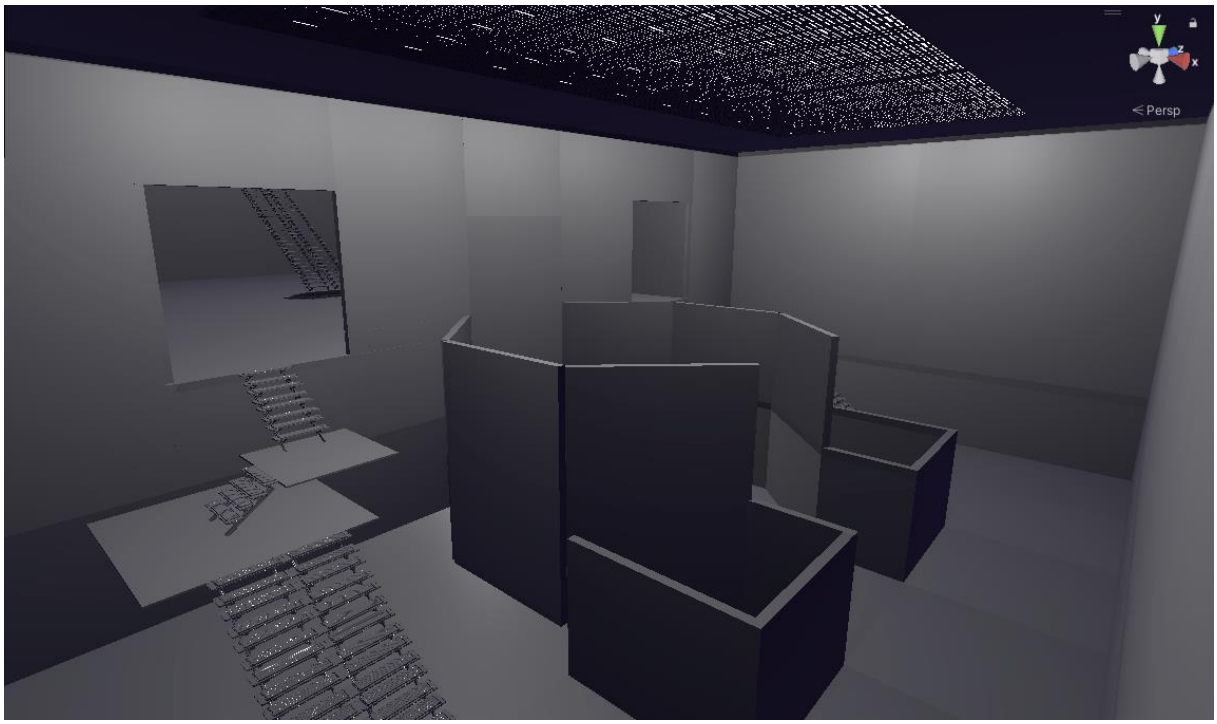


Ilustración 3.112; Segunda planta, sala intermedia a la tercera planta

3.2.4.3.4 Tercera planta

En la tercera planta, se tendrán dos grandes salas.

3.2.4.3.4.1 Primera sala

La primera habitación será la que conecta el hangar con la sala intermedia de la segunda planta.

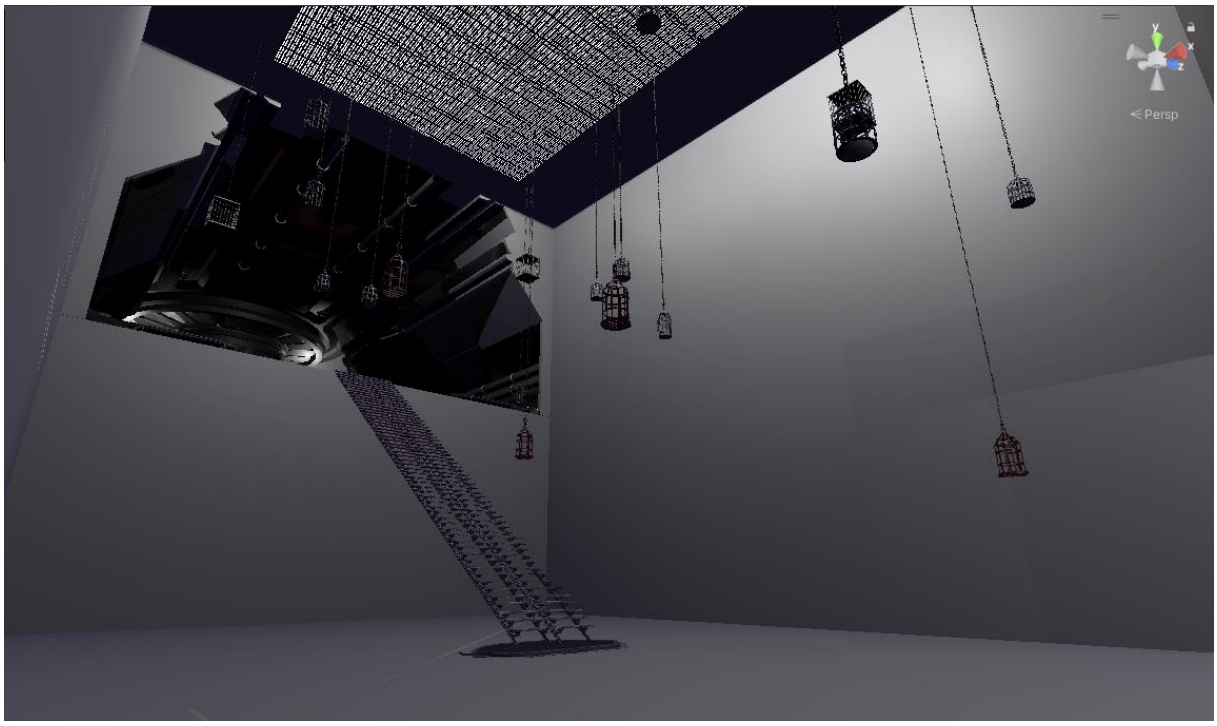


Ilustración 3.113; Tercera planta, primera sala

Esta habitación tampoco tiene ninguna funcionalidad, solo da paso al jugador a entrar al hangar.

3.2.4.3.4.2 Hangar

Esta será una sección de la nave en la que el jugador se podrá encontrar un portal que le llevará directo al final del juego.

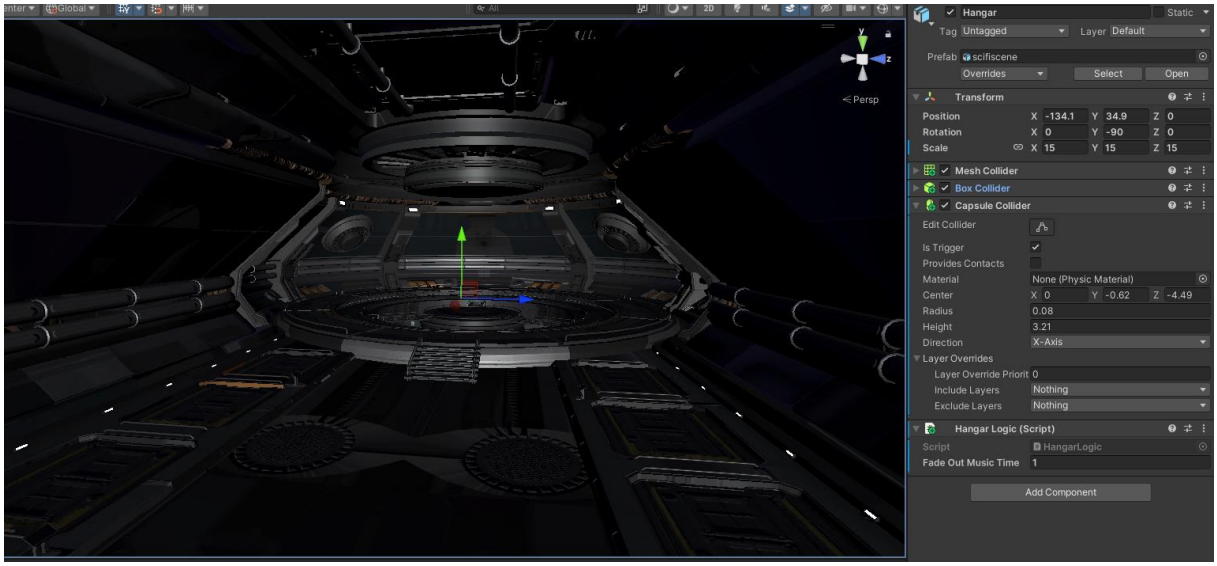


Ilustración 3.114; Tercera planta, hangar

Esta sala tiene asociado un componente “CapsuleCollider” [81] con la opción “isTrigger” marcada.

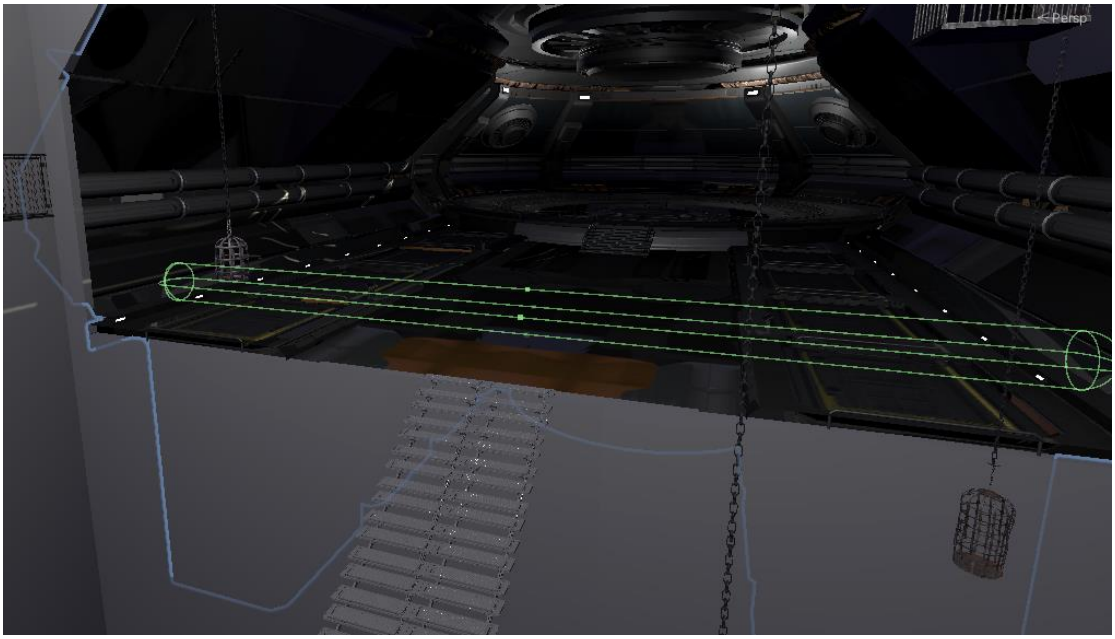


Ilustración 3.115; Tercera planta, collider del hangar

El hangar tiene el script llamado “HangarLogic”, el cual controla si el jugador entra en contacto con el componente “CapsuleCollider”, lo que activaría el portal que llevará al jugador al final del juego.

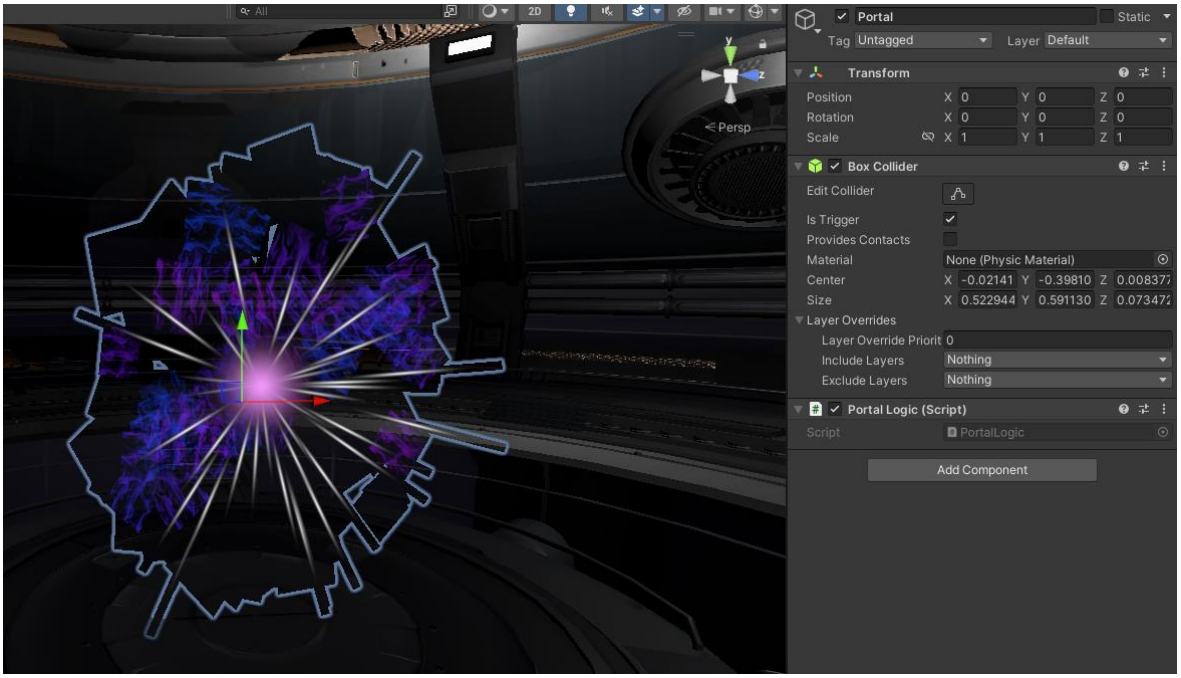


Ilustración 3.116; Tercera planta, portal del hangar

Este portal tiene asociado un componente “BoxCollider” con la opción “isTrigger” marcada, además de un script llamado “PortalLogic”, el cual detectará si el jugador entra en contacto con el componente “BoxCollider”. En ese momento, el script llamará al método “GoToCredits()” del GameManager.

3.12.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el hangar de la tercera planta

Test_04_44	
Nombre de la prueba o test	Activación del portal
Objetivo	El portal debe activarse en el momento adecuado
Resultado	El portal del hangar se activa justo cuando el jugador colisiona con el componente “BoxCollider” del hangar.
Comentarios	El jugador puede pensar que va a aparecer un nuevo enemigo

Tabla 3.77; Test_04_44: activación del portal

Test_04_45	
Nombre de la prueba o test	Movimiento del jugador por el hangar
Objetivo	El jugador debe poder moverse por todo el hangar
Resultado	El jugador puede moverse por todo el hangar sin tener problemas de caídas y sin quedarse pillado en ningún sitio
Comentarios	Todo OK.

Tabla 3.78; Test_04_45: movimiento del jugador por el hangar

Test_04_46	
Nombre de la prueba o test	Paso a créditos
Objetivo	El jugador podrá interactuar con el portal para ir a los créditos
Resultado	El jugador puede ir a los créditos finales del juego interactuando con el portal

Comentarios	El jugador puede no saber qué tiene que hacer en esa situación

Tabla 3.79; Test_04_46: paso a créditos

3.2.4.4 Exterior

El exterior de la nave también forma parte del entorno del juego, por lo que también es importante destacarlo.

En esta sección veremos a cerca de: la Tierra, la Luna, el Sol y los cohetes de propulsión de la nave espacial.

3.2.4.4.1.1 Tierra

Primeramente, se puede ver la Tierra. Esta, como se puede ver en la siguiente imagen, tiene asociado un script llamado “Rotar”, el cual hace que la Tierra gire a una determinada velocidad de forma constante.

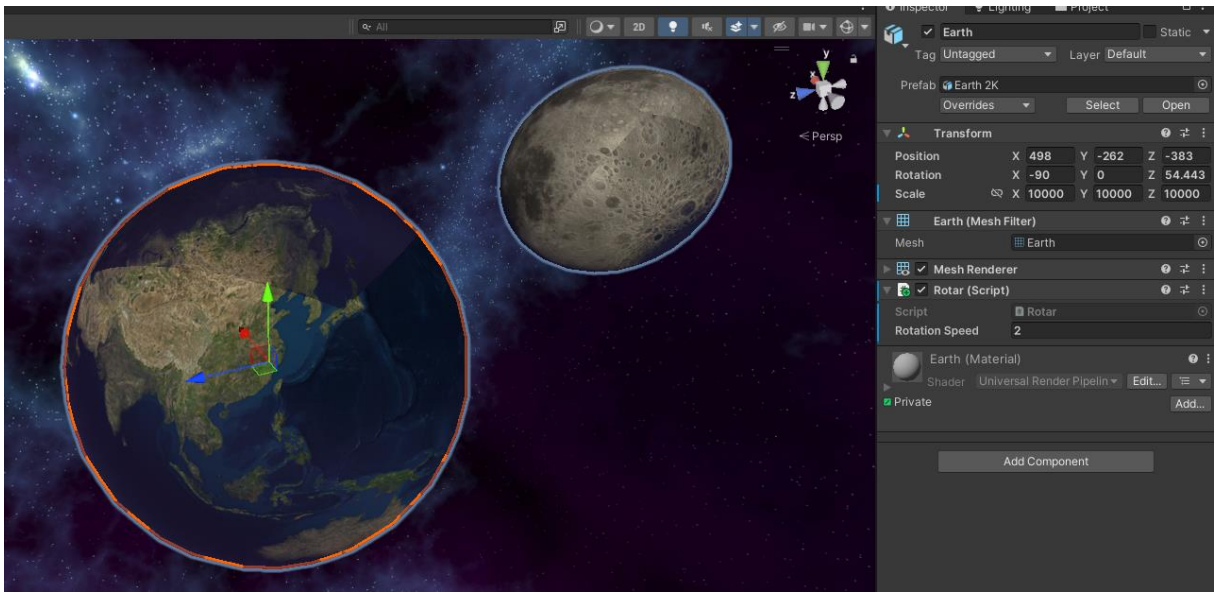


Ilustración 3.117; Tierra, componentes

3.14.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la Tierra, encontrada en el exterior de la nave espacial

Test_04_47	
Nombre de la prueba o test	Rotación de la Tierra
Objetivo	La Tierra debe estar rotando de forma realista
Resultado	La Tierra gira de forma realista sobre sí misma.
Comentarios	El jugador puede pensar que la Tierra no gira si no se detiene a ver su rotación.

Tabla 3.80; Test_04_47: rotación de la Tierra

3.2.4.4.1.2 Luna

Hablando sobre la Luna, se puede apreciar que siempre está acompañada de la Tierra, gracias a su script “RotateAround” con el cual se consigue que la Luna orbite alrededor de un objeto de referencia a una velocidad determinada. Ese objeto de referencia, en este caso, es la Tierra.

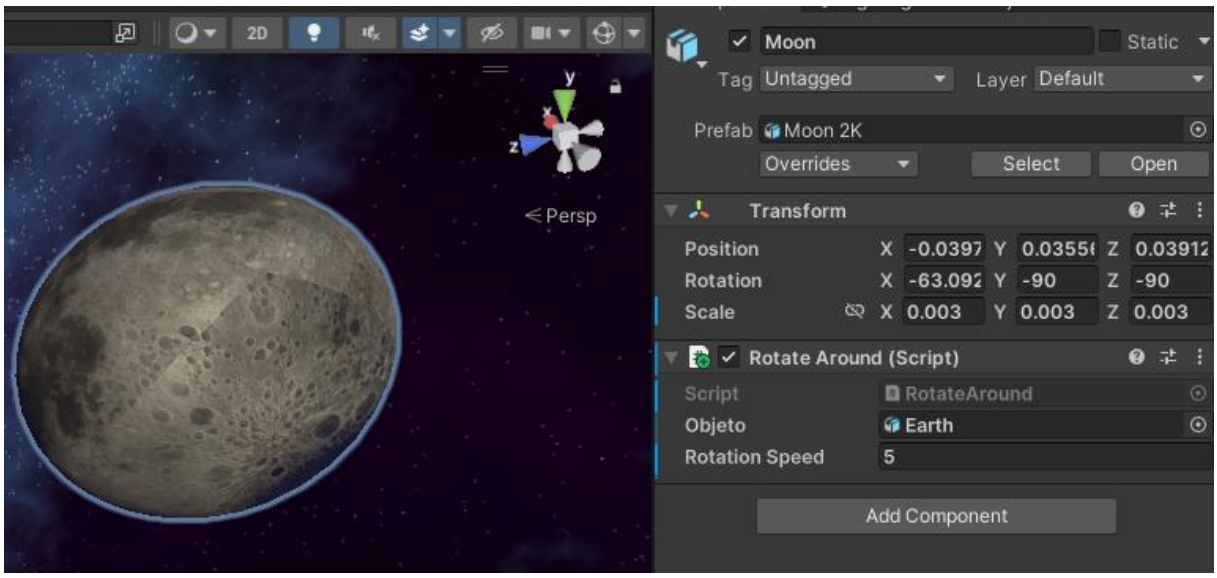


Ilustración 3.118; Luna, componentes

3.16.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la Luna, encontrada en el exterior de la nave espacial

Test_04_48	
Nombre de la prueba o test	Rotación de la Luna
Objetivo	La Luna debe estar rotando sobre sí misma
Resultado	La Luna rota sobre sí misma de forma realista
Comentarios	El jugador puede no darse cuenta de que la Luna está rotando sobre sí misma si este no se detiene a verla.

Tabla 3.81; Test_04_48: rotación de la Luna

Test_04_49	
Nombre de la prueba o test	Rotación de la Luna sobre la Tierra
Objetivo	La Luna debe estar rotando alrededor de la Tierra
Resultado	La Luna rota alrededor de la Tierra de forma realista
Comentarios	Todo OK.

Tabla 3.82; Test_04_49: rotación de la Luna sobre la Tierra

3.2.4.4.1.3 Sol

El Sol tiene una peculiaridad y es que es un efecto visual.

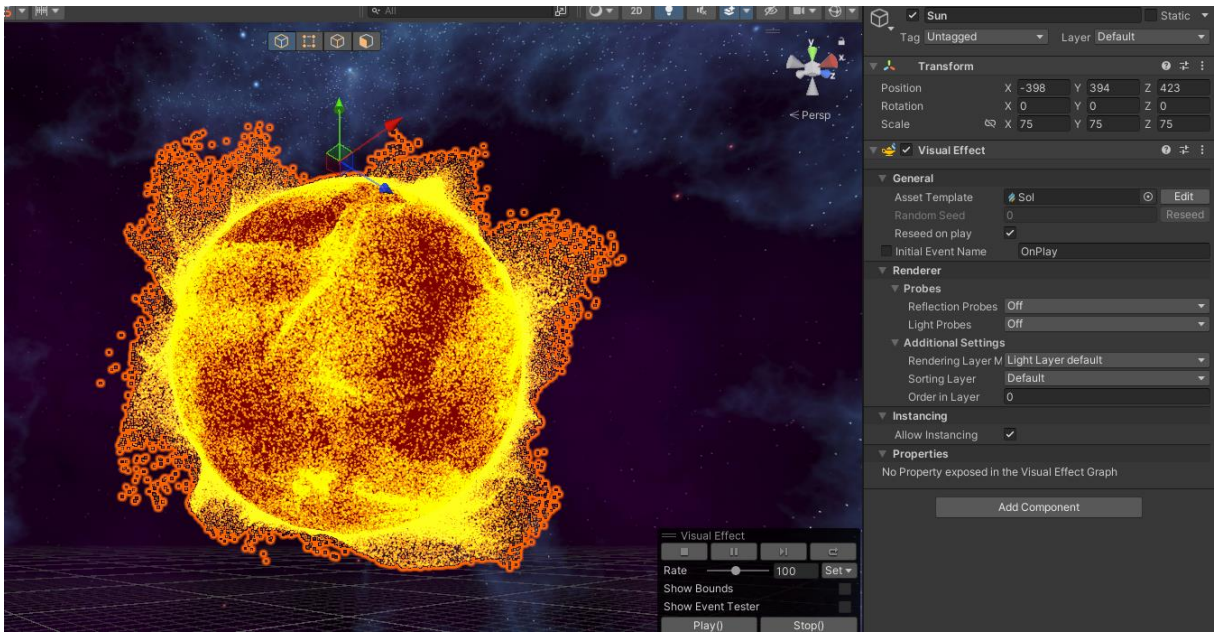


Ilustración 3.119; Sol, componentes

Este efecto visual (o VFX) utiliza un sistema de partículas para crear un Sol.

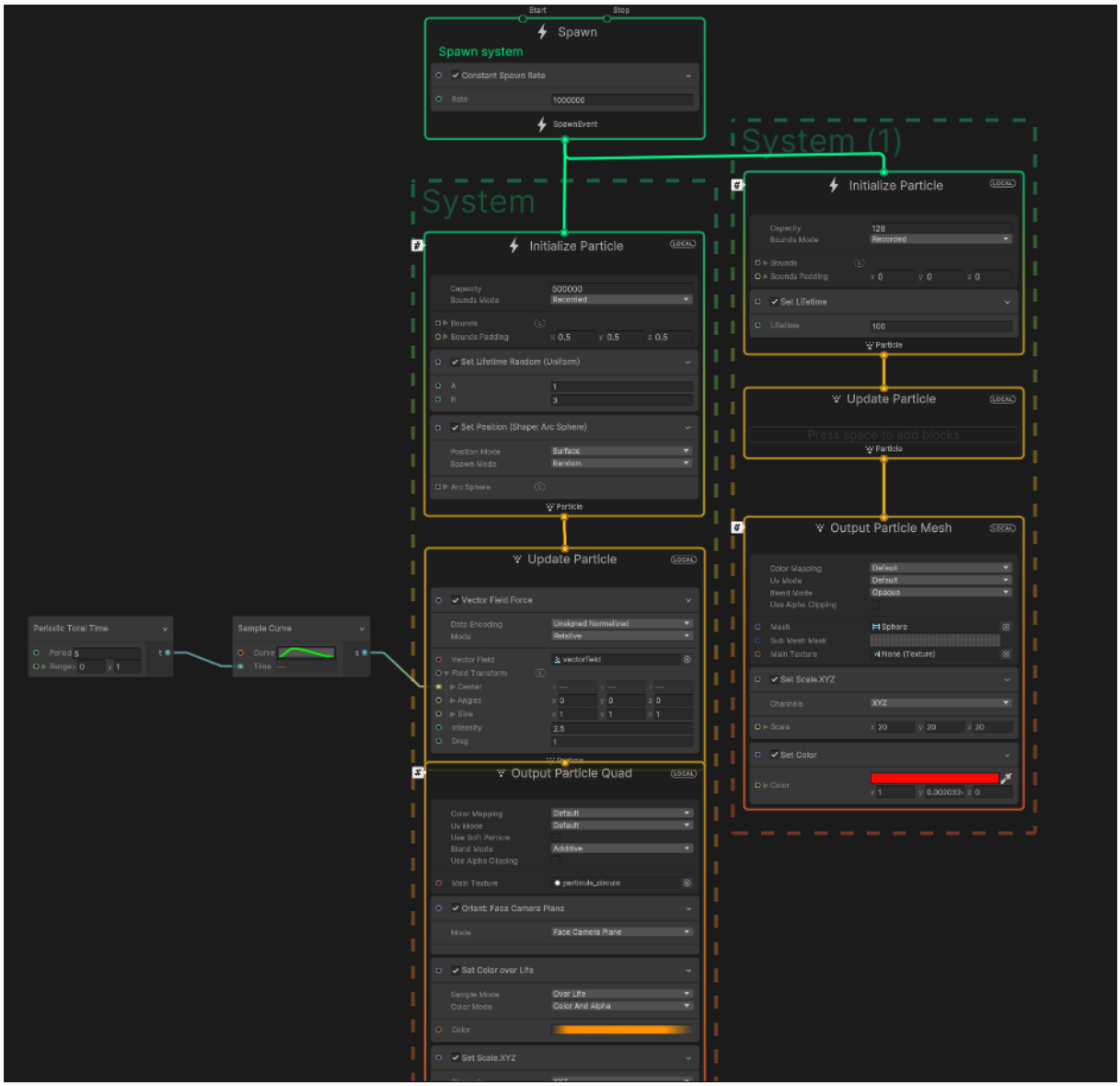


Ilustración 3.120; Sol, efecto visual (VFX)

Al igual que con los Shader Graphs, se va a desglosar este efecto de partículas para poder analizarlo mejor.

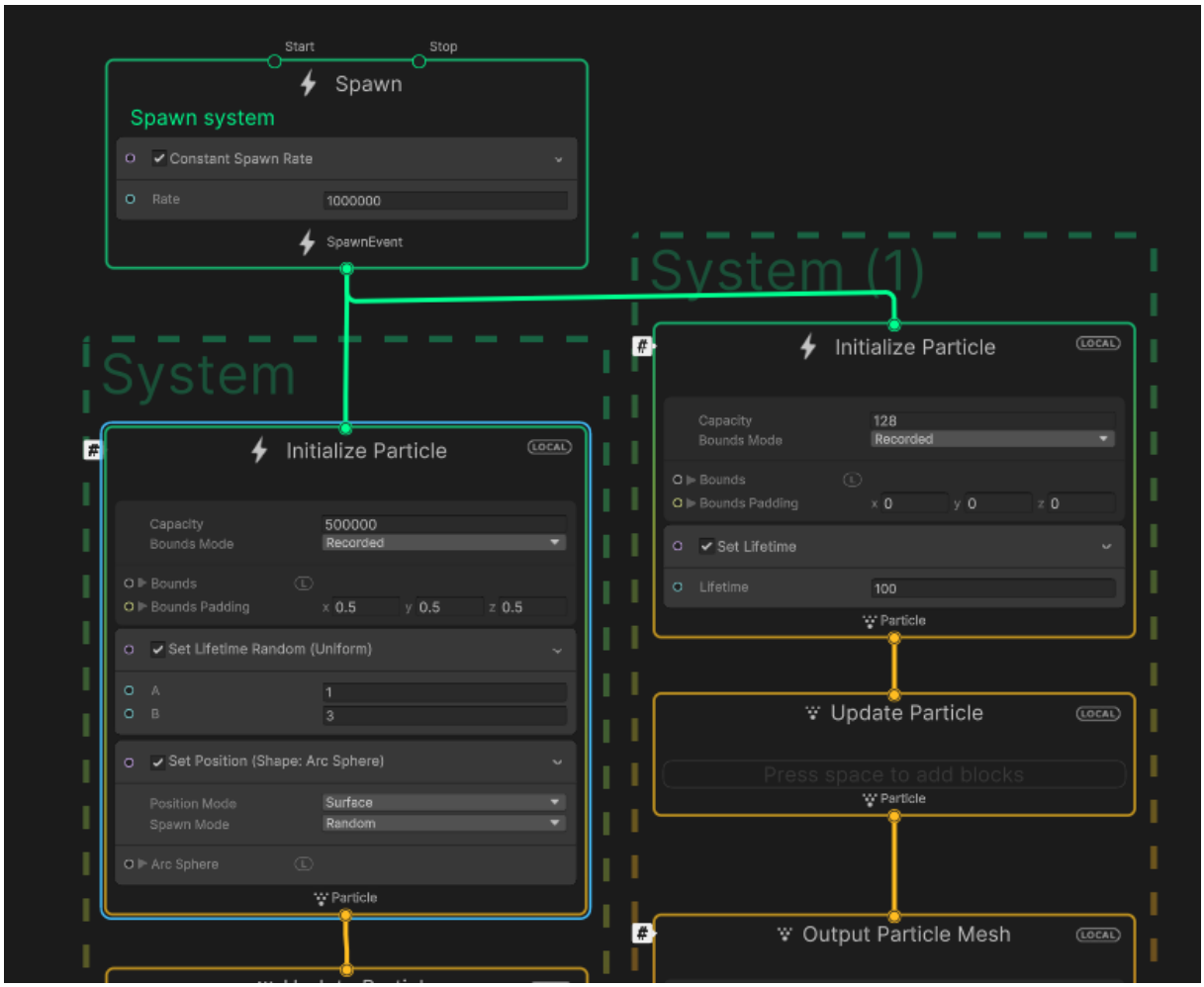


Ilustración 3.121; Sol, efecto visual (VFX), primera parte

La funcionalidad de este efecto de partículas es que, primeramente, en el módulo “Spawn”, se generan una cantidad determinada de partículas de forma constante, en este caso, 1.000.000 de partículas.

Después, en el módulo “Initialize Particle”, se establece la capacidad del pool de partículas, el cual almacena las partículas que el sistema debe renderizar en un frame. Además, se establece parámetros como el tiempo de vida de la partícula, la posicion de la partícula, … Aunque el parámetro más importante que se define es el de la posición de la partícula usando una forma esférica (arc sphere), imprescindible para poder simular el Sol.

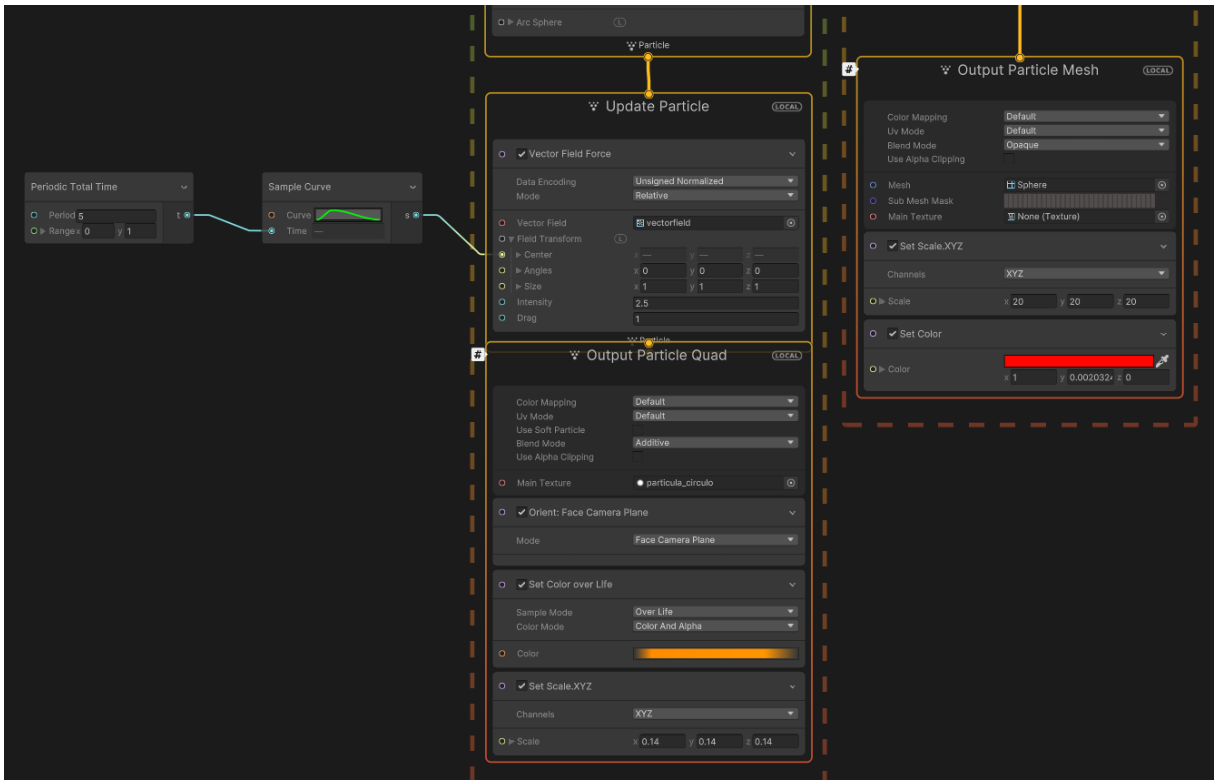


Ilustración 3.122; Sol, efecto visual (VFX), segunda parte

Después de inicializar las partículas, se pasa a actualizar las partículas del pool, con el módulo “Update particle”. En su actualización, se aplica una fuerza basada en un campo vectorial para que esta influya en el movimiento y trayectoria de las partículas. Importante destacar el parámetro “Drag”, con el que cual se aplica una fuerza de arrastre a las partículas para que esta afecte a la velocidad de las partículas.

Por último, el módulo “Output particle quad” se encarga de definir cómo se visualizan las partículas. Por ejemplo, el “Output particle mesh” permite emitir partículas como mallas 3D, lo que aporta mayor realismo a la partícula ya que le aporta una forma tridimensional, además de que las mallas en tres dimensiones pueden interactuar con las luces y las sombras. Mientras que el “Output particle quad” permite emitir partículas como cuadrados 2D (quads) orientados a la cámara.

3.18.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el Sol, encontrado en el exterior de la nave espacial

Test_04_50	
Nombre de la prueba o test	Dirección de luz direccional del Sol
Objetivo	La luz direccional debe apuntar al centro del mundo
Resultado	La luz direccional apunta al centro del mapa, la nave espacial
Comentarios	Todo OK.

Tabla 3.83; Test_04_50: dirección de la luz direccional del Sol

Test_04_51	
Nombre de la prueba o test	Rendimiento del Sol
Objetivo	El Sol debe aparentar estar lejos, pero manteniendo un buen rendimiento de la aplicación
Resultado	El Sol parece estar lejano y el rendimiento de la aplicación es óptimo.
Comentarios	Todo OK.

Tabla 3.84; Test_04_51: rendimiento del Sol

3.2.4.4.1.4 Cohetes de propulsión

Los cohetes a propulsión, aunque forman parte de la nave espacial, no afectan dentro de la nave, por lo que se ha preferido comentarlos fuera del apartado de la nave espacial.

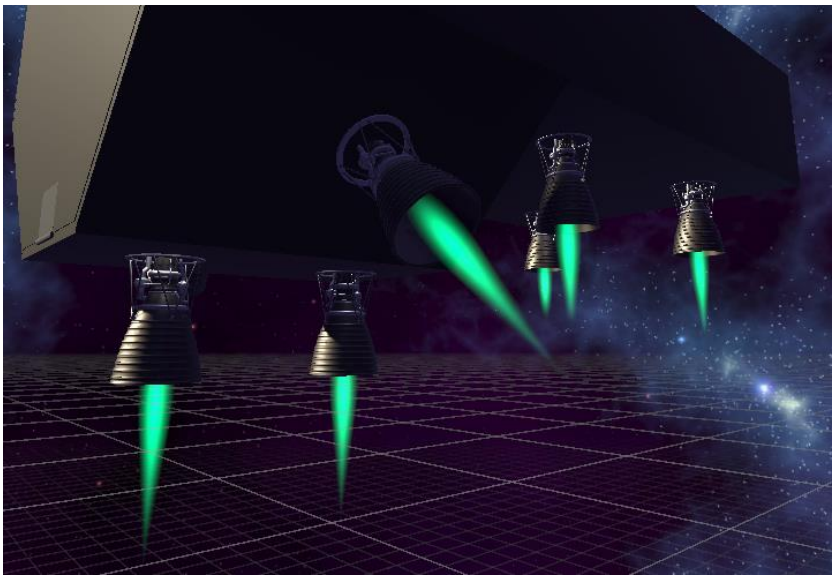


Ilustración 3.123; Cohetes de propulsión de la nave

Cada uno de estos cohetes a propulsión tiene un elemento llamado “RocketFlame” el cual tiene un sistema de partículas incorporado.

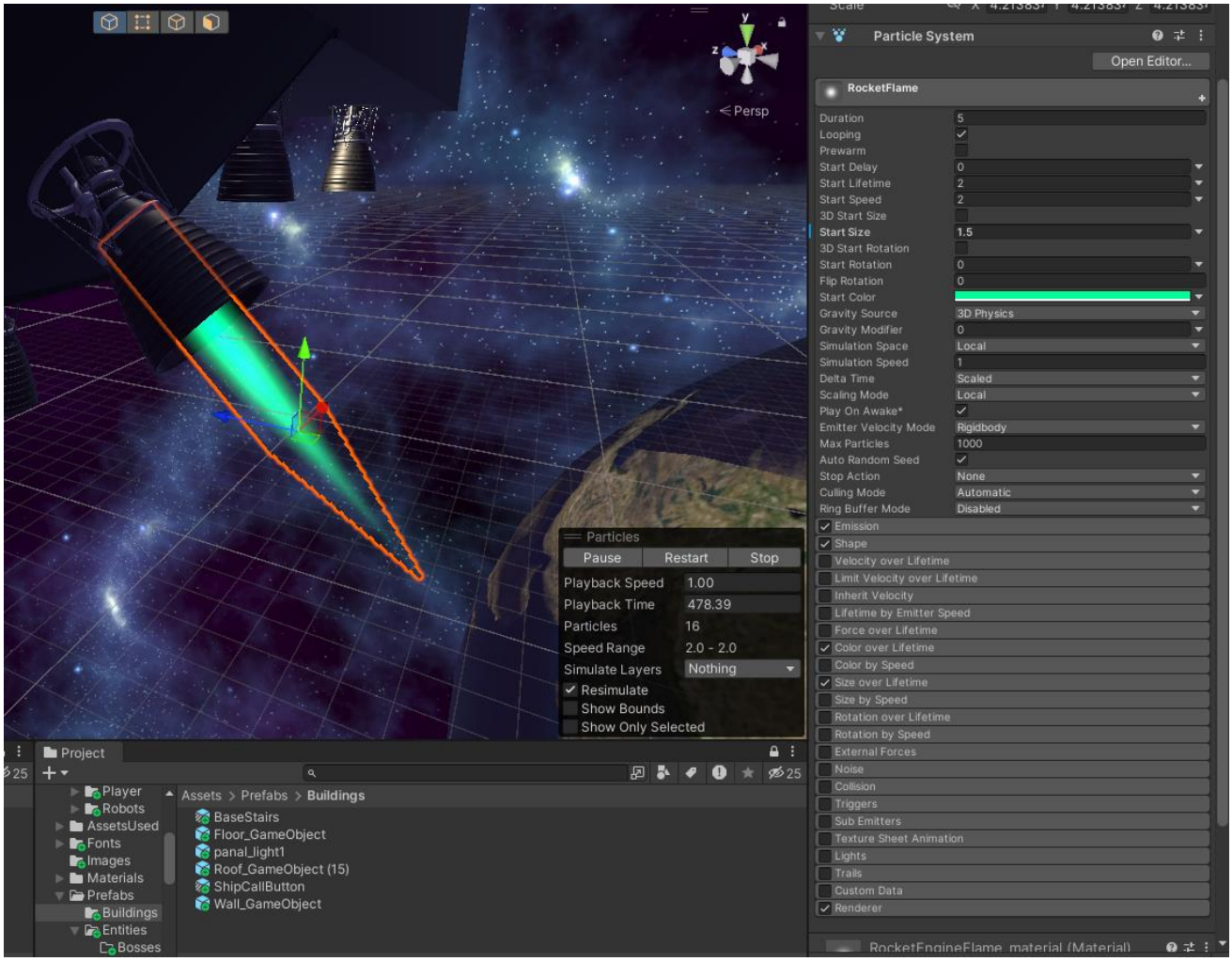


Ilustración 3.124; Cohetes de propulsión de la nave, sistema de partículas

Estos sistemas de partículas son controlados por el script “ShipController”. Lo que hace este script es que, cuando es activado mediante la pulsación del botón central de la cúpula, aumenta el tamaño de las partículas del sistema de partículas. Como se ha comentado en apartados anteriores, al pulsar el botón central de la cúpula, la nave se moverá. Si añadimos a ese movimiento esta variación de los sistemas de partículas, se estará simulando que se han activado los motores y que, por tanto, la nave se va a mover a causa del movimiento de estos motores cohete.

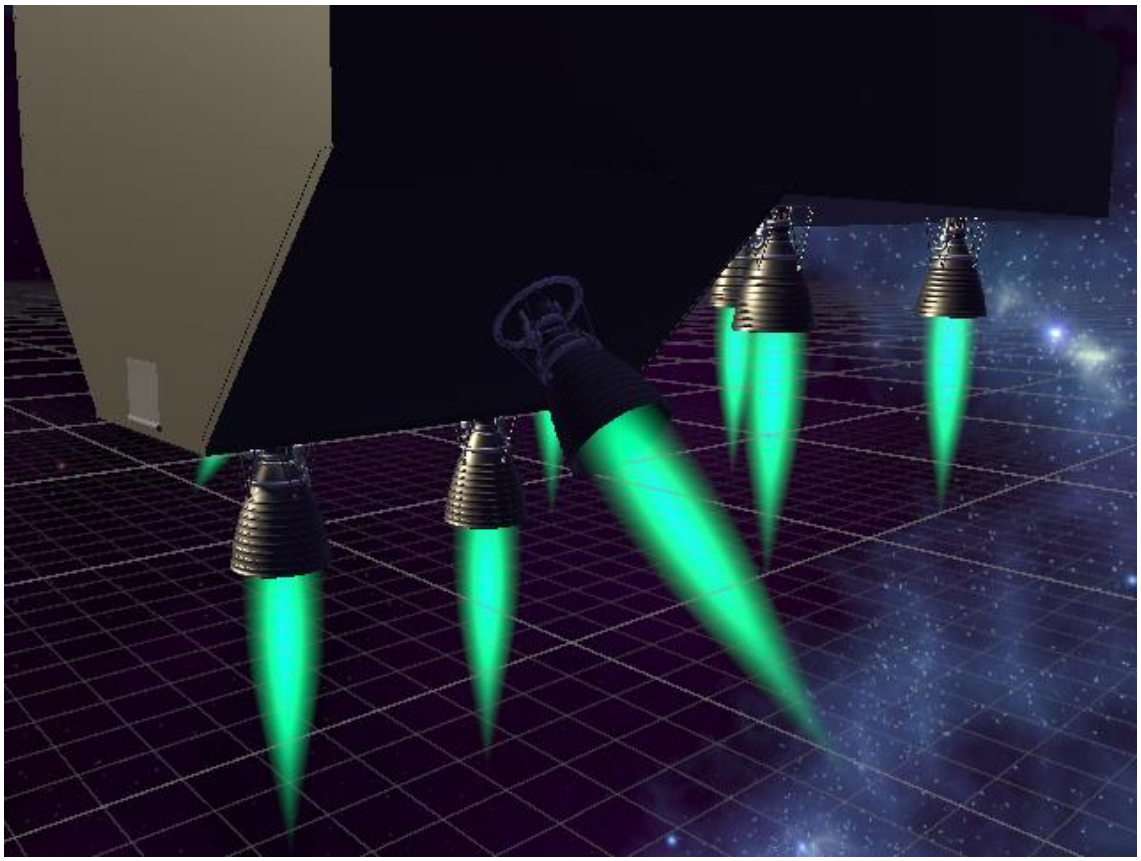


Ilustración 3.125; Cohetes de propulsión de la nave, activados

También destacar que, cuando la nave llegue a su destino, las partículas de los sistemas de partículas volverán a su tamaño original, simulando que la nave ya ha llegado a su destino y no se va a mover más.

3.20.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para los cohetes de propulsión, encontrados en el exterior de la nave espacial

Test_04_52	
Nombre de la prueba o test	Fuerza de los cohetes
Objetivo	La fuerza de los cohetes al ser activados debe ser adecuada
Resultado	Los cohetes, al ser activados, tienen una fuerza que realista, sin ser demasiado grandes
Comentarios	Todo OK.

Tabla 3.85; Test_04_52: fuerza de los cohetes

3.2.5 Personaje

Nuestro personaje, Doomboy, es el protagonista de esta historia y por ello, es fundamental que esté bien diseñado y preparado para ser controlado por el jugador.

Este modelo tiene muchos elementos que son indispensables en el personaje principal del juego, como pueden ser la cámara principal o el rig del modelo.

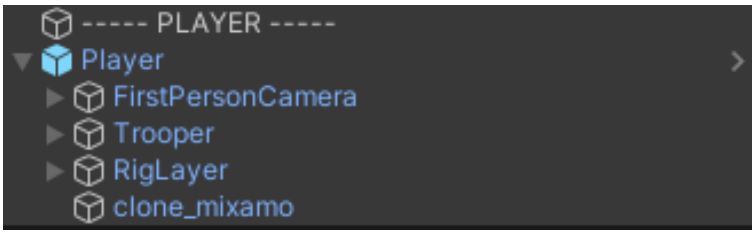


Ilustración 3.126; Elementos del personaje

Además, tiene asociados múltiples componentes que serán necesarios para mover al personaje y poder tener todas las funcionalidades necesarias para jugarlo.

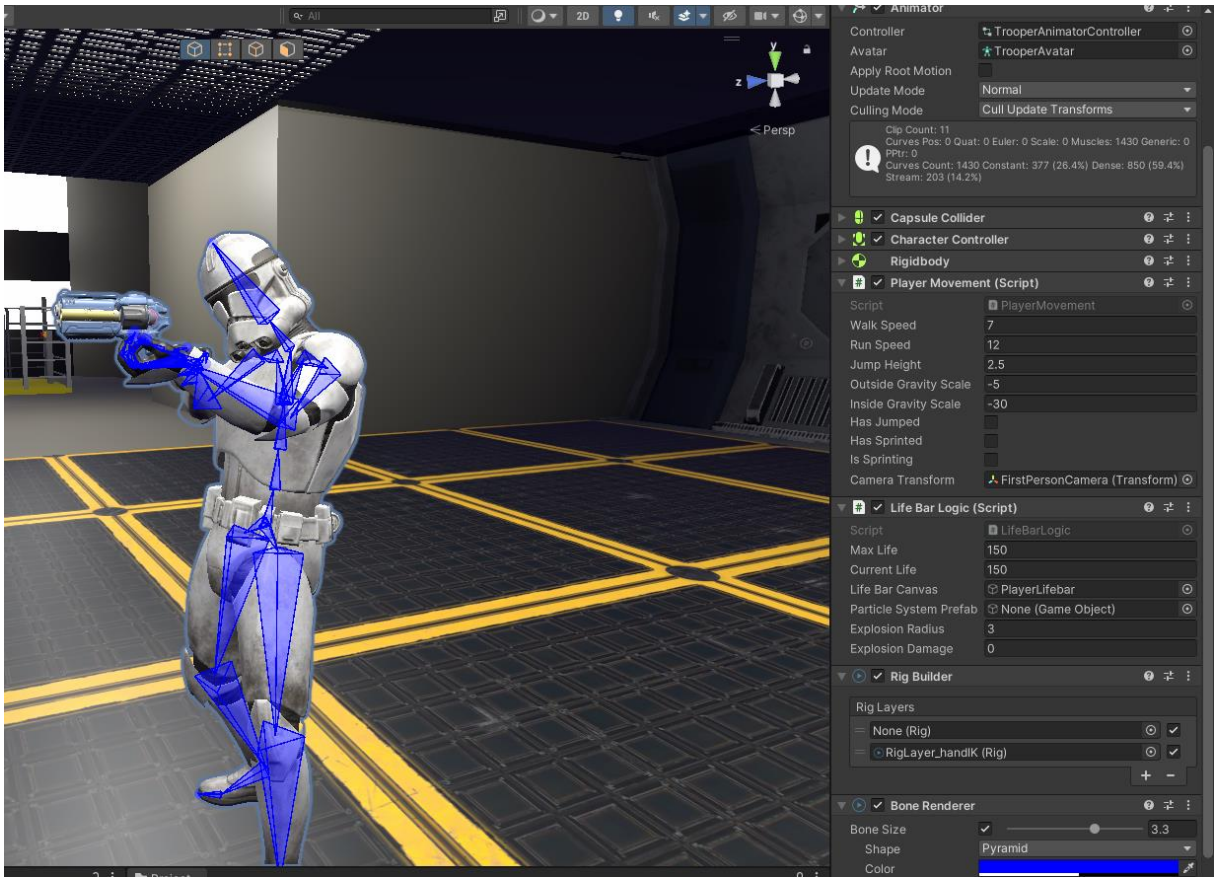


Ilustración 3.127; Componentes del personaje

Entre todos esos componentes, destacamos los siguientes:

1. "**CharacterController**": necesario para manejar y gestionar de forma óptima el movimiento completo del personaje, como moverse, caminar, correr, saltar, …
2. "**RigidBody**" [82]: imprescindible para que personaje interactúe de forma dinámica con otros objetos, físicamente hablando. Además de que permita responder a fuerzas como la gravedad y otros efectos físicos.

3.2.5.1 Animación

Comentar también el componente "Animator" [83] asociado al modelo y con el que se podrán visualizar animaciones al mover al personaje principal. Este componente tiene un componente "AnimatorController" asociado el cual tiene la siguiente estructura:

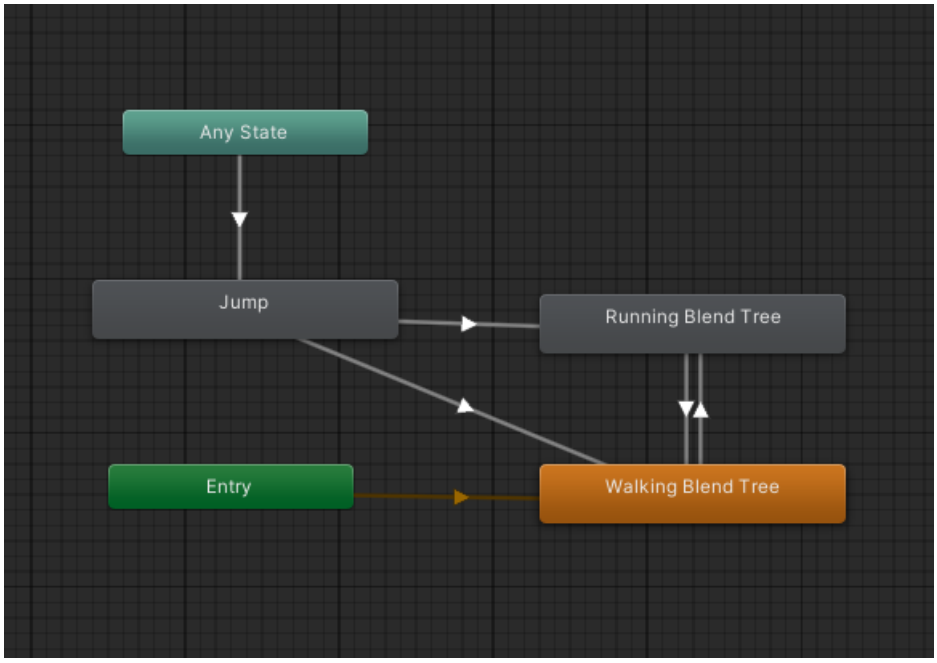


Ilustración 3.128; AnimatorController del personaje

Esta estructura puede parecer simple a primera vista, pero esta estructura cuenta con dos componentes "BlendTree" [84], los cuales realizan una determinada

animación en función de unos parámetros u otros que se actualizan en tiempo real dentro del juego.

Antes de detallar los distintos “BlendTrees”, comentamos que existe un parámetro booleano “isRunning”, con el cual se puede controlar si se activa el “Running Blend Tree” si este parámetro es “true”, o si, por el contrario, se activa el “Walking Blend Tree” si este parámetro es “false”. También tendremos otro parámetro booleano “isJumping”, con el que controlaremos si se debe iniciar la animacion de salto o no en función del valor de este parámetro. Si la animación de salto es activada, cuando se desactive, se activará el “Running Blend Tree” o el “Walking Blend Tree” en función del valor del parámetro “isRunning”. El valor de estos parámetros será controlado por el script “PlayerMovement”, del cual se hablará más adelante.

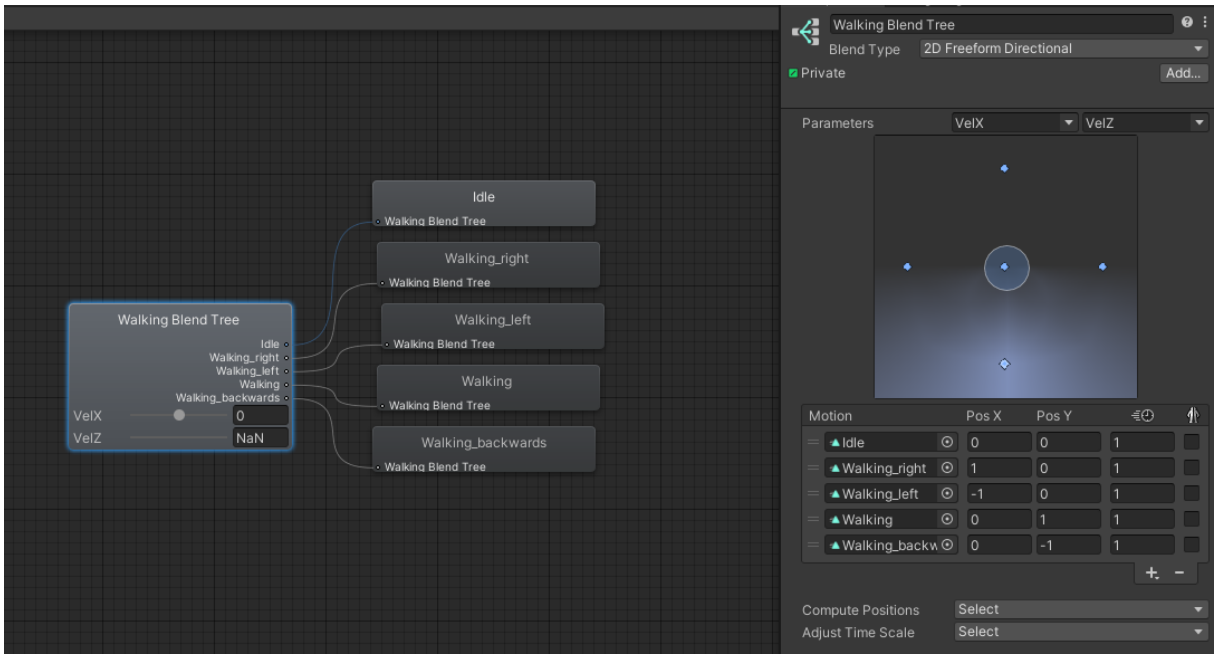


Ilustración 3.129; Walking Blend Tree del AnimatorController del personaje

El “Walking Blend Tree” está compuesto por varios nodos, los cuales representan diferentes estados de movimiento:

1. **"Idle":** parado
2. **“Walking”:** caminar hacia adelante
3. **"Walking_right":** caminar hacia la derecha

4. **"Walking_left":** caminar hacia la izquierda
5. **"Walking_backwards":** caminar hacia atrás

Estos nodos están configurados para responder a dos parámetros, "VelX" y "VelZ", los cuales representan componentes de la velocidad del personaje en las direcciones X y Z. Al variar los valores de estos parámetros en el script “PlayerMovement” a tiempo real, varían los valores “PosX” y “PosY” del “BlendTree” y en función de los valores “PosX” y “PosY”, el “BlendTree” selecciona la animación adecuada.

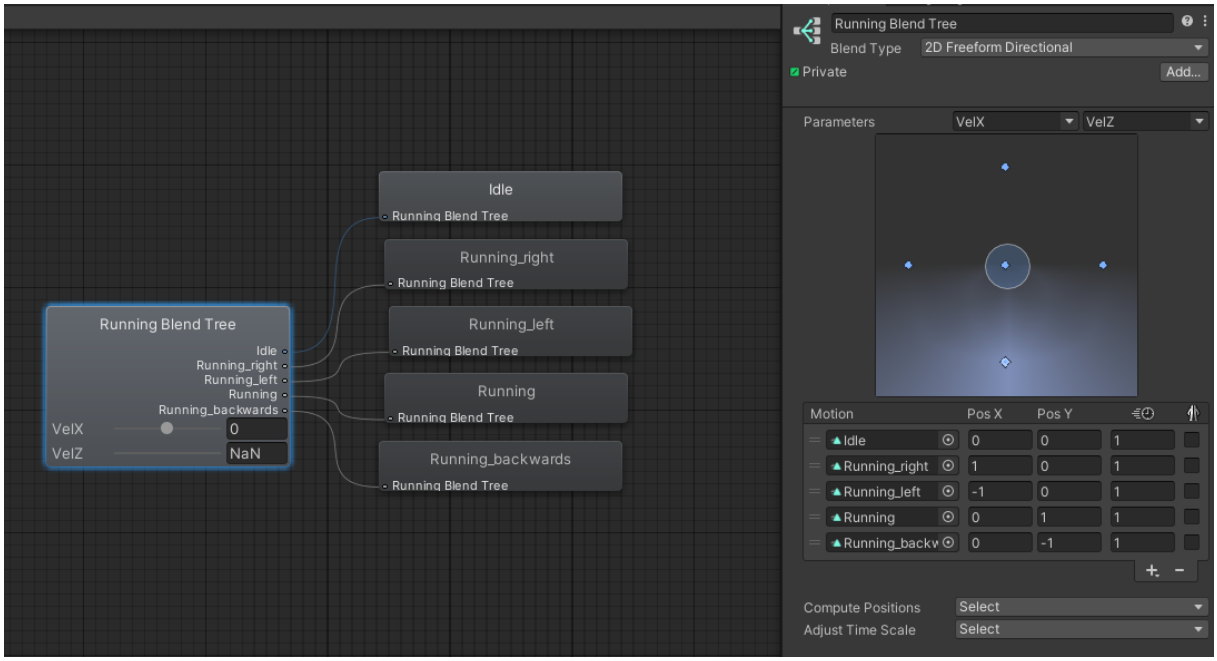


Ilustración 3.130; Running Blend Tree del AnimatorController del personaje

En el caso del “Running Blend Tree” funciona exactamente igual que el “Walking Blend Tree” pero con animaciones en las cuales el personaje corre en vez de caminar.

3.22.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para las animaciones del personaje principal.

Test_05_01	
Nombre de la prueba o test	Flujo de animaciones del personaje
Objetivo	El flujo de las animaciones del personaje debe ser acorde a las acciones tomadas
Resultado	El personaje realiza unas animaciones u otras en función de las acciones realizadas, como saltar, correr, andar, ...
Comentarios	Todo OK.

Tabla 3.86; Test_05_01: flujo de animaciones del personaje

3.2.5.2 Scripts asociados

Como se puede ver en la imagen del principio del capítulo, el modelo del personaje tiene asociado varios scripts.

3.2.5.2.1 "PlayerMovement"

El primero de ellos es "PlayerMovement", con el cual se maneja el movimiento y animaciones del personaje. Este script controla el movimiento del personaje en dos dimensiones (horizontal y vertical) basándose en la entrada del usuario por los dispositivos de entrada y la orientación de la cámara. Además, como se ha comentado, este script diferencia entre caminar y correr gracias a los dispositivos de entrada, lo que resulta en el uso de diferentes velocidades de movimiento del personaje y la activación de animaciones acordes a caminar o correr por la variación de los parámetros del componente "AnimatorController" anteriormente comentados.

A parte de correr y saltar, este script también controla el salto del personaje. Para realizar ese movimiento, se aplica la siguiente fórmula matemática:

$$cambioIntroducidoDelMovimiento.y = \sqrt{alturaSalto * -2 * gravedad}$$

Con esta fórmula, se puede calcular la velocidad que se le aplica al eje Y del personaje para que este alcance la altura determinada, "alturaSalto", teniendo en cuenta la gravedad del entorno donde se encuentra. La gravedad será -30 dentro de

la nave para simular que existe una gran fuerza de gravedad dentro de la nave, y -5 fuera de la nave para simular que estamos en el espacio exterior, donde no hay gravedad.

Cabe destacar que, el script “PlayerMovement”, permite realizar cualquiera de los movimientos comentados solo si el componente “CharacterController” del personaje detecta que está en una superficie sólida. Para ello, se usa la función “isGrounded()” [85] del componente “CharacterController”, el cual detecta si este entra en contacto con un objeto (el suelo) con componente “Collider”, lo que devolverá “true”. Además de tener el personaje que estar en el suelo para poder realizar cualquier movimiento, el personaje no debe estar hablando con ningún NPC del juego. Este estado está controlado por la variable “isTalking”, cuyo valor se puede variar a través de sus getters y setters públicos.

Por último, comentar que existen parámetros como “hasJumped” o “hasSprinted” que sirven para que el script “KyleRobotLogic“ del tutorial, del cual se hablará más adelante, sepa si el jugador ha realizado esos movimientos para proseguir con el tutorial.

El pseudocódigo del script “PlayerMovement” es:

```
Update()
  Si la posición x del transform es mayor o igual a -5
    Establece gravityScale a insideGravityScale: -30

  Si no
    Establece gravityScale a outsideGravityScale: -5

  Llamada método Move()

Move()
  Si el personaje está en el suelo
    Si no está hablando
      x <- valor del eje Horizontal de dispositivo entrada
      z <- valor del eje Vertical de dispistivo entrada

    Si no
      x <- 0
      z <- 0

    Velocidad de animación VelX <- x
    Velocidad de animación VelZ <- z

    Calcula los vectores forward y right basados en la cámara
    Normaliza estos vectores y elimina sus componentes y

    moveInput <- forward*z + right*x
    moveInput limitado a 1

    Si el botón "Sprint" está presionado
      Multiplica moveInput por runSpeed
      Establece hasSprinted e isSprinting a verdadero
      Activa la animación de correr

    Si no
      Establece isSprinting a falso
      Multiplica moveInput por walkSpeed
      Desactiva la animación de correr

    Si el botón "Jump" es presionado
      Aplicamos fórmula a moveInput.y
      Establece hasJumped a verdadero
      Activa la animación de salto

    Si no
      Desactiva la animación de salto

  Aplica la gravedad a moveInput
  Mueve el componente "CharacterController" del personaje según moveInput
```

Ilustración 3.131; Pseudocódigo del script PlayerMovement

3.24.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el movimiento del personaje principal

Test_05_02	
Nombre de la prueba o test	De andar, a correr

Objetivo	El personaje debe poder pasar de andar a correr sin problema.
Resultado	El personaje, al pulsar el jugador la tecla de correr, se pone a correr, aumentando su velocidad de movimiento.
Comentarios	Todo OK.

Tabla 3.87; Test_05_02: de andar, a correr

Test_05_03	
Nombre de la prueba o test	De correr, a andar
Objetivo	El personaje debe poder pasar de correr a andar sin problema.
Resultado	El personaje, al dejar de pulsar el jugador la tecla de correr, se pone a andar, disminuyendo su velocidad de movimiento.
Comentarios	Todo OK.

Tabla 3.88; Test_05_03: de correr, a andar

Test_05_04	
Nombre de la prueba o test	Poder saltar en cualquier situación
Objetivo	El personaje debe poder saltar en cualquier momento siempre y cuando esté tocando el suelo
Resultado	El personaje puede saltar en cualquier momento, siempre que este esté tocando el suelo, independientemente de si el personaje está corriendo o andando.
Comentarios	Todo OK.

Tabla 3.89; Test_05_04: poder saltar en cualquier situación

Test_05_05	
Nombre de la prueba o test	Distancia de salto
Objetivo	El personaje debe poder saltar más o menos en función de la ubicación de este.
Resultado	El personaje salta más si está fuera de la nave y salta menos si está dentro de ella.

Comentarios	El jugador puede no esperarse que el personaje vaya a saltar menos dentro de la nave si antes ha probado a saltar fuera de ella.

Tabla 3.90; Test_05_05: distancia de salto

3.2.5.2.2 "LifeBarLogic"

Este script es genérico, es decir, es usado en todas las entidades con vida. Este script tiene una imagen asociada, la cual representa la barra de vida de la entidad. El funcionamiento de este script es variar el componente "fillAmount" de la imagen en función de la cantidad de vida en ese momento y de la vida máxima de la entidad. Además, se varía el color de la imagen usando la función "Lerp", de forma que se usa para el color una tonalidad entre verde y rojo en función de la cantidad de vida de la entidad, donde, si la cantidad vida de la entidad está al máximo, el color será verde y si la cantidad vida de la entidad está a 0, el color será rojo.

También comentar que, en el caso de que la entidad muera, es decir, su vida llegue a 0, se ejecutará una corrutina llamada "DeathRoutine". Esta corrutina eliminará el canvas de la vida de la entidad asociada y además realizará varias acciones en función de cuál sea la entidad asociada al script. Cabe destacar que, si la entidad asociada al script no es ni "Cyborg" ni "Droid" ni "Player", se instanciará un sistema de partículas para indicar visualmente al jugador que el enemigo ha muerto. Acompañado de ese sistema de partículas irá un daño que se le aplicará a los enemigos cercanos a la entidad. Detallamos estas acciones con el siguiente pseudocódigo:

```
Update()
  fillAmount de lifeBarImage <- currentLife / maxLife
  color de lifeBarImage <- interpolación entre verde y rojo según currentLife / maxLife

  Si currentLife es menor o igual a 0
    Llamada a "DeathRoutine"

DeathRoutine()
  Destruye lifeBarCanvas

  Si nombre de entidad es "Droid"
    Se destryuen objetos "Sparks" de la entidad

  Si nombre de entidad no es "Cyborg" ni "Droid" ni "Player"
    Se instancia sistema de particulas
    Se declara que se destruye la instancia del sistema de partículas después de 1 segundo

    Para cada enemigo de la escena
      Si enemigo no es esta entidad
        Calcula la distancia hasta el enemigo
        Si la distancia es menor o igual a explosionRadius
          Actualiza la vida del enemigo con -explosionDamage

    Espera 1 segundo

    Desactivamos la entidad de la escena
    Destruimos la entidad de la escena

UpdateLife(cantidad)
  Incrementa currentLife por cantidad
  Limita currentLife entre 0 y maxLife para evitar problemas
  fillAmount de lifeBarImage <- currentLife / maxLife
```

Ilustración 3.132; Pseudocódigo del script LifeBarLogic

Es importante destacar que todos los objetos “Canvas” [86] de las barras de vida de las diferentes entidades, exceptuando la del personaje, tienen asociado el script “LifeBarLookingCamera”, con el cual hacemos que el “Canvas” siempre esté mirando hacia la cámara del personaje. Con esto conseguimos que la barra de vida de las entidades siempre sea visible al jugador.

3.26.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la barra de vida del personaje

Test_05_06	
Nombre de la prueba o test	Barra de vida del jugador es visible
Objetivo	La barra de vida del jugador debe estar verse en todo momento
Resultado	La barra de vida del jugador está siempre visible en su interfaz
Comentarios	Todo OK.

Tabla 3.91; Test_05_06: barra de vida del jugador es visible

Test_05_07	
Nombre de la prueba o test	Color de la barra de vida
Objetivo	La barra de vida debe poder cambiar de color en función de su salud
Resultado	La barra de vida cambia de color en función de la vida de la entidad a la que esté asociada.
Comentarios	Todo OK.

Tabla 3.92; Test_05_07: color de la barra de vida

Test_05_08	
Nombre de la prueba o test	Barra de vida refleja daño recibido
Objetivo	La barra de vida del jugador debe reflejar el daño recibido
Resultado	La barra de vida se reduce al recibir daño.
Comentarios	Todo OK.

Tabla 3.93; Test_05_08: barra de vida refleja daño recibido

3.2.6 Cámaras

Las cámaras suponen un papel fundamental en cualquier aplicación o proyecto en el que se renderiza una escena, ya que esta realiza la fase de “culling” [87] del renderizado, en la cual se determina qué se tiene que renderizar y qué no se tiene que renderizar en función de si está dentro de la visión de la cámara o no.

En nuestro proyecto, vamos a tener dos cámaras, las cuales se alternarán en función del estado en el que se encuentre el juego.

3.2.6.1 Cámara del jugador

Esta cámara es la principal del juego, y es la que usa el jugador para sumergirse en él.

3.2.6.1.1 Objetos hijos de la cámara

Los elementos principales de la cámara son los que se pueden ver en la siguiente imagen.

Ilustración 3.133; Elementos de la cámara principal

3.28.4.3.3.2PotionThrower

Este elemento controla el lanzamiento de pociones, dibujando una trayectoria con un componente “LineRenderer” para que el jugador pueda saber cuál va a ser el movimiento de la poción. La funcionalidad de este elemento estará detallada en el **apartado 3.2.12.3.1 PotionThrower****.**

3.28.4.3.3.3WeaponPosition

Este elemento es simplemente un objeto cuya finalidad es establecer el transform del arma del jugador en función de si el jugador está apuntando o no.

Si el jugador está apuntando, el arma tendrá el componente “Transform” [88] de “AimingWeaponPosition”.

Ilustración 3.134; Arma apuntando

Si no, tendrá el componente “Transform” de “WeaponPosition”.

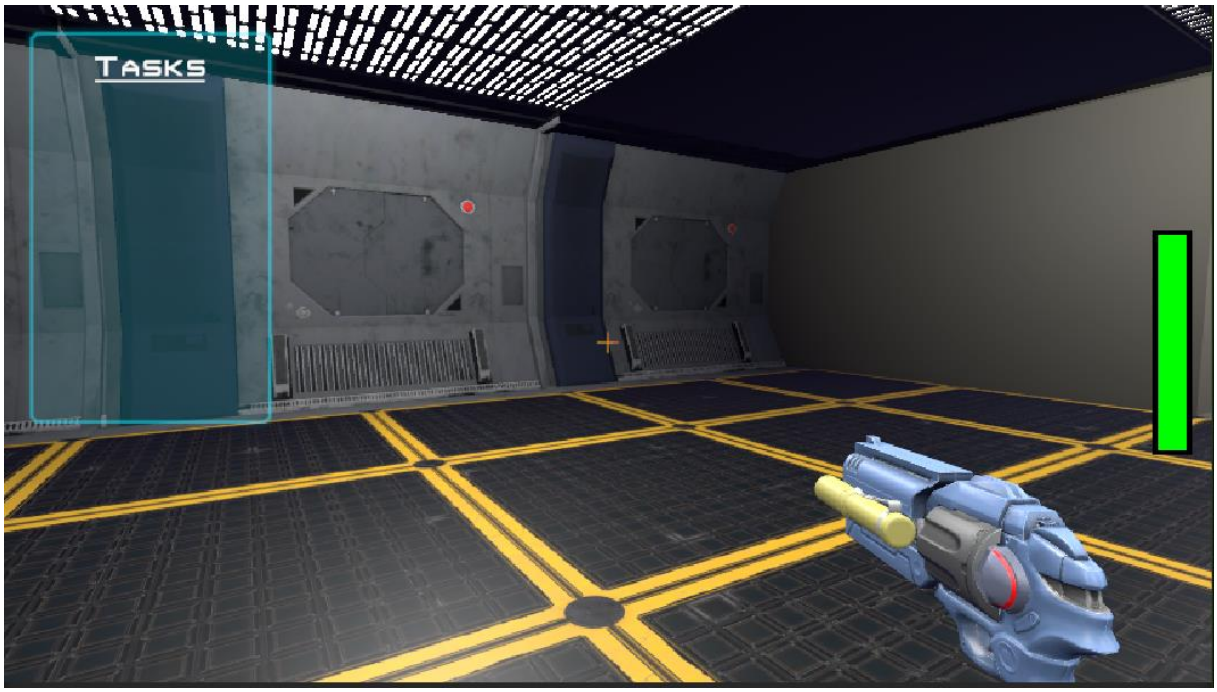


Ilustración 3.135; Arma sin apuntar

Para controlar si el jugador está apuntando o no, se usa el script “CameraLook”, el cual veremos más adelante.

3.28.4.3.3.4PotionPosition

Este es un elemento parecido a “WeaponPosition”, pues controla la posición de las pociones del jugador. Este elemento tiene asociado un script llamado “Potion_switching”, del cual se hablará en el apartado **3.2.12.6 Cambio entre ítems y el arma.**

3.2.6.1.2 Scripts asociados

Esta cámara está situada a la altura de la cabeza del personaje principal para conseguir más realismo en la inmersión del jugador dentro del juego.

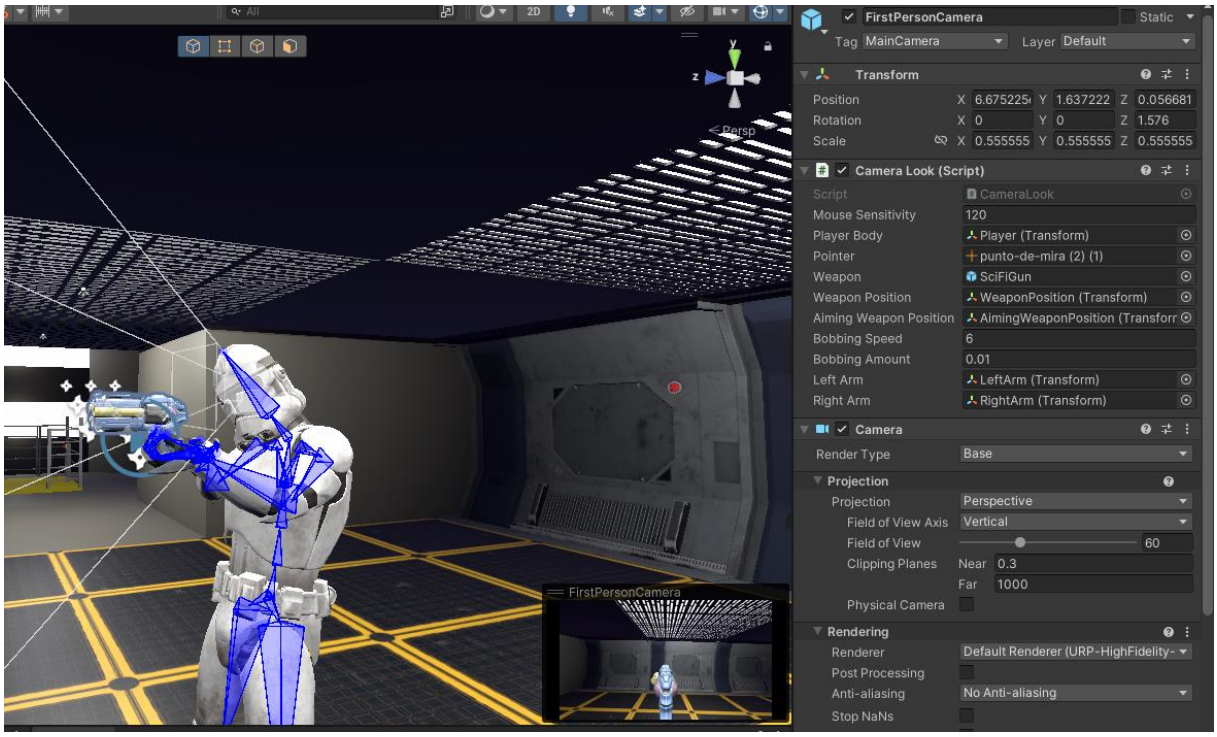


Ilustración 3.136; Componentes de la cámara principal

Para poder manejar su lógica, la cámara tiene asociada un script llamado “CameraLook”, con el que le personaje puede mover la cámara para ver el entorno del juego. Este script tiene varias funcionalidades.

La primera de ellas es el control de la cámara. Cuando el jugador mueve el ratón, el script ajusta la rotación de la cámara basándose en la sensibilidad pasada por parámetro, lo que permite al jugador mirar alrededor en el juego. Simultáneamente, el cuerpo del jugador gira en función del movimiento horizontal del ratón, con lo que se asegura que la dirección de la cámara y el cuerpo estén sincronizados. Cabe destacar que la rotación vertical de la cámara está limitada entre -90 y 90 grados para evitar giros completos y/o no deseados hacia arriba o hacia abajo.

Luego, como comentamos anteriormente, este script controla también si el jugador apunta con el arma o no mediante la pulsación del botón de apuntado, el click derecho, y el arma se moverá hacia una posición u otra de forma suave usando la función “Lerp”. Comentar que apuntar o no tendrá un efecto en la precisión del arma. Si el jugador apunta con el arma, las balas irán al centro de la cruceta de su interfaz,

mientras que, si este no apunta, las balas no irán al centro, recompensando el apuntar dentro del juego.

Ilustración 3.137; Colisión de la bala del arma sin apuntar

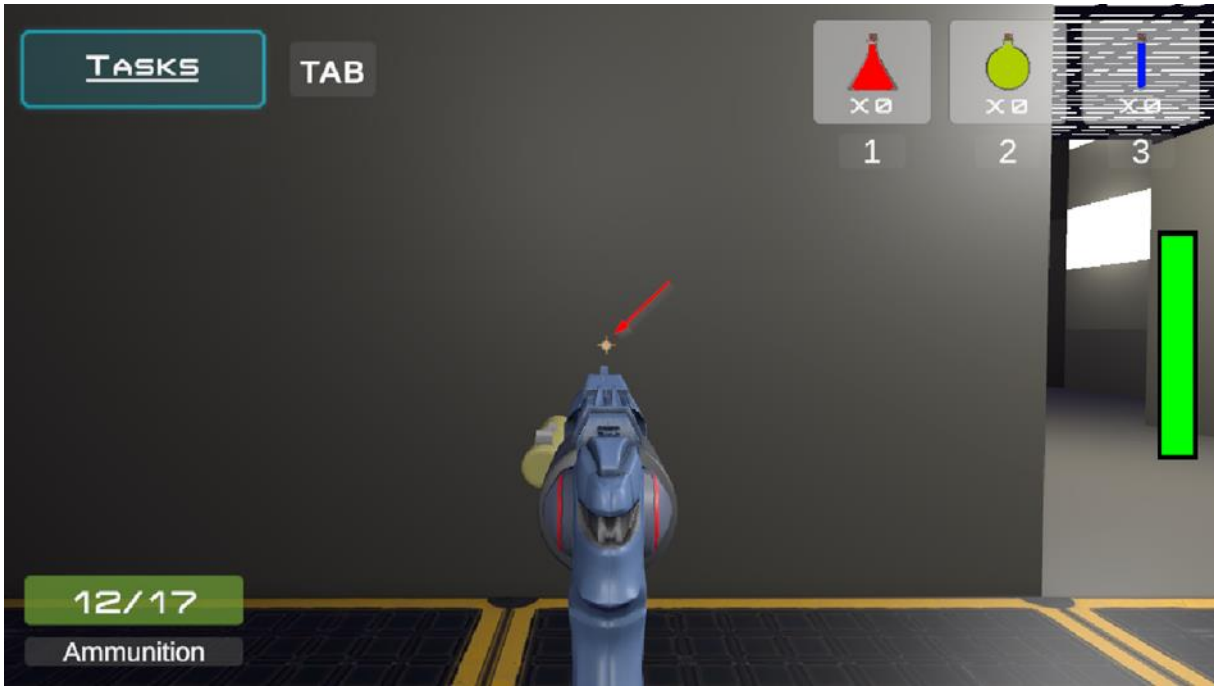


Ilustración 3.138; Colisión de la bala del arma apuntando

Este script también se encarga de balancear el arma al moverse el jugador. Si el jugador está corriendo, la intensidad y la velocidad del bamboleo aumentan, caso contrario si el jugador está andando. El script calcula este efecto usando funciones trigonométricas, con las que se genera el movimiento de balanceo y se ajusta la posición del arma en relación a su posición original y el movimiento del jugador. Este efecto se podrá notar tanto si el jugador corre como si camina. Comentar que, al pararse el jugador, el arma vuelve suavemente a su posición inicial.

Por último, comentar que el script, con el método “OnGui()” [89], dibuja un cursor en el centro de la pantalla para ayudar al jugador a orientarse y apuntar.

Se presenta el pseudocódigo del script:

```
Update()
  mouseX <- Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime
  mouseY <- Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime

  xRotation <- xRotation - mouseY
  xRotation <- Clampeamos xRotation entre -90 y 90 grados

  Rotación local de la cámara <- (xRotation, 0, 0)
  Rota playerBody en Vector3.up * mouseX

  moveInput <- Vector2(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"))

  BobbingEffect()

  leftArm.localRotation <- Quaternion.Euler(xRotation, 0, 0)
  rightArm.localRotation <- Quaternion.Euler(xRotation, 0, 0)

  Si se presiona el botón derecho del ratón y getItemShowed() == ItemShowed.Weapons en GameManager
    isAiming <- verdadero
    currentLerpTime <- 0

  Si se libera el botón derecho del ratón
    isAiming <- falso
    currentLerpTime <- 0

  Si isAiming es verdadero
    currentLerpTime += Time.deltaTime * aimingSpeed
    weapon.transform.position <- Lerp hacia aimingWeaponPosition.position
    weapon.transform.rotation <- Lerp hacia aimingWeaponPosition.rotation
  Si no
    currentLerpTime += Time.deltaTime * aimingSpeed
    weapon.transform.position <- Lerp hacia weaponPosition.position
    weapon.transform.rotation <- Lerp hacia weaponPosition.rotation

BobbingEffect()
  isSprinting <- GameManager.playerIsSprinting()

  currentBobbingSpeed <- isSprinting ? bobbingSpeed * 2 : bobbingSpeed
  currentBobbingAmountY <- isSprinting ? bobbingAmount * 2 : bobbingAmount
  currentBobbingAmountX <- currentBobbingAmountY * 0.5
  currentBobbingAmountZ <- currentBobbingAmountY * 0.3

  Si moveInput.x > 0.1 o moveInput.y > 0.1
    timer += Time.deltaTime * currentBobbingSpeed

    waveSliceX <- Seno de timer * 0.5
    waveSliceY <- Seno de timer
    waveSliceZ <- Coseno de timer * 0.2

    totalMovement <- Clampeamos   moveInput.x + moveInput.y   entre 0 y 1

    offsetX <- waveSliceX * currentBobbingAmountX * totalMovement
    offsetY <- waveSliceY * currentBobbingAmountY * totalMovement
    offsetZ <- waveSliceZ * currentBobbingAmountZ * totalMovement

    bobbingPosition <- (defaultPosX + offsetX, defaultPosY + offsetY, defaultPosZ + offsetZ)
    weaponPosition.localPosition <- Lerp hacia bobbingPosition

  Si no
    timer <- 0
    startPosition <- (defaultPosX, defaultPosY, defaultPosZ)
    weaponPosition.localPosition <- Lerp hacia startPosition

OnGUI()
  Dibuja el cursor como una textura en el centro de la pantalla
```

Ilustración 3.139; Pseudocódigo del script CameraLook

3.28.4.3.3.4.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la sala lateral de la segunda planta

Test_06_01	
Nombre de la prueba o test	Precisión del arma
Objetivo	Las balas del arma deben ser disparadas con mayor precisión si el jugador apunta
Resultado	Las balas del arma son disparadas con mayor precisión si el jugador apunta su arma
Comentarios	El jugador debe saber que la precisión cambia en función de si el jugador apunta su arma o no

Tabla 3.94; Test_06_01: precisión del arma

Test_06_02	
Nombre de la prueba o test	Apuntado del arma
Objetivo	El jugador debe poder apuntar el arma
Resultado	El arma, al apuntar el jugador, se desplaza hacia la posición de apuntado, y si el jugador deja de apuntar, el arma vuelve a su posicion natural.
Comentarios	Todo OK.

Tabla 3.95; Test_06_02: apuntado del arma

Test_06_03	
Nombre de la prueba o test	Movimiento de la cámara
Objetivo	La cámara del jugador debe moverse reaccionando al movimiento del dispositivo de entrada
Resultado	La cámara del jugador se mueve en función del movimiento realizado con el dispositivo de entrada
Comentarios	Todo OK.

Tabla 3.96; Test_06_03: movimiento de la cámara

Test_06_04	
Nombre de la prueba o test	Efecto de bobbing del arma (tambaleo del arma al moverse el jugador)
Objetivo	El efecto de bobbing del arma debe ser claramente visible
Resultado	El efecto de bobbing del arma se ve claramente y no es exagerado
Comentarios	El jugador puede no llegar a verlo debido a la sutileza del efecto

Tabla 3.97; Test_06_04: efecto de bobbing

Test_06_05	
Nombre de la prueba o test	Efecto de bobbing en función del movimiento
Objetivo	El efecto de bobbing del arma del jugador debe ser mayor o menor en función de si el jugador está corriendo o andando
Resultado	El efecto de bobbing del arma del jugador es mayor al correr el jugador, y menor al andar.
Comentarios	Todo OK.

Tabla 3.98; Test_06_05: efecto de bobbing en función del movimiento

3.2.6.2 Cámara del menú

A la hora de realizar el menú de inicio, se ha elegido hacer algo al estilo del famoso videojuego “Minecraft” [90]. Este juego tiene un menú de inicio en el cual se muestran las acciones del menú, pero de fondo se va viendo el mundo para que el jugador se vaya haciendo una idea del entorno en donde va a jugar.

Ilustración 3.140; Menú principal de Minecraft. Imagen obtenida de Minecraft Wiki. © Mojang Studios.

Para poder realizar la misma dinámica que el famoso juego de Mojang, lo primero es reducir el componente de transparencia del “Canvas” del menú para que se vea el fondo.

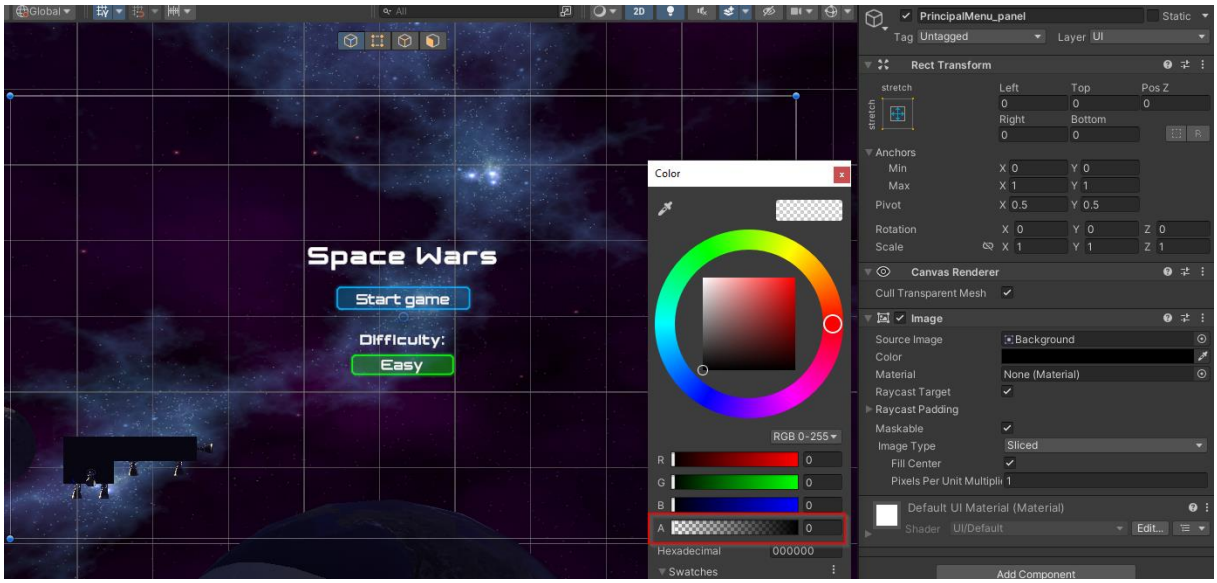


Ilustración 3.141; Componente Alfa del color del canvas del menú principal.

Después, se han colocado diversos puntos posibles en los que puede aparecer la cámara por el entorno del juego.

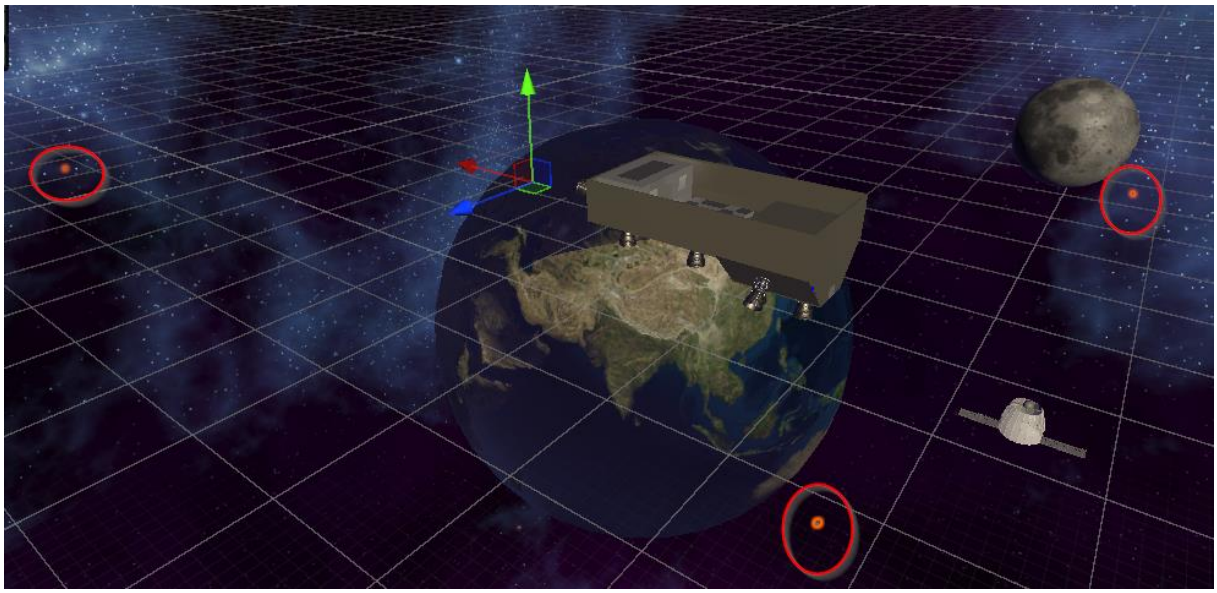


Ilustración 3.142; Posiciones de la cámara del menú principal

Ahora, a la cámara del menú se le ha asignado un script llamado “CameraController”, el cual, habiéndole pasado los distintos puntos de reaparición de la cámara, se escoge uno totalmente al azar y la cámara empieza a rotar en el eje Y hasta 70º desde su estado inicial. Cuando la cámara haya rotado los 70º en el eje Y, la cámara volverá a coger otro punto de reaparición de forma aleatoria, sin tener en cuenta el punto en el que está.

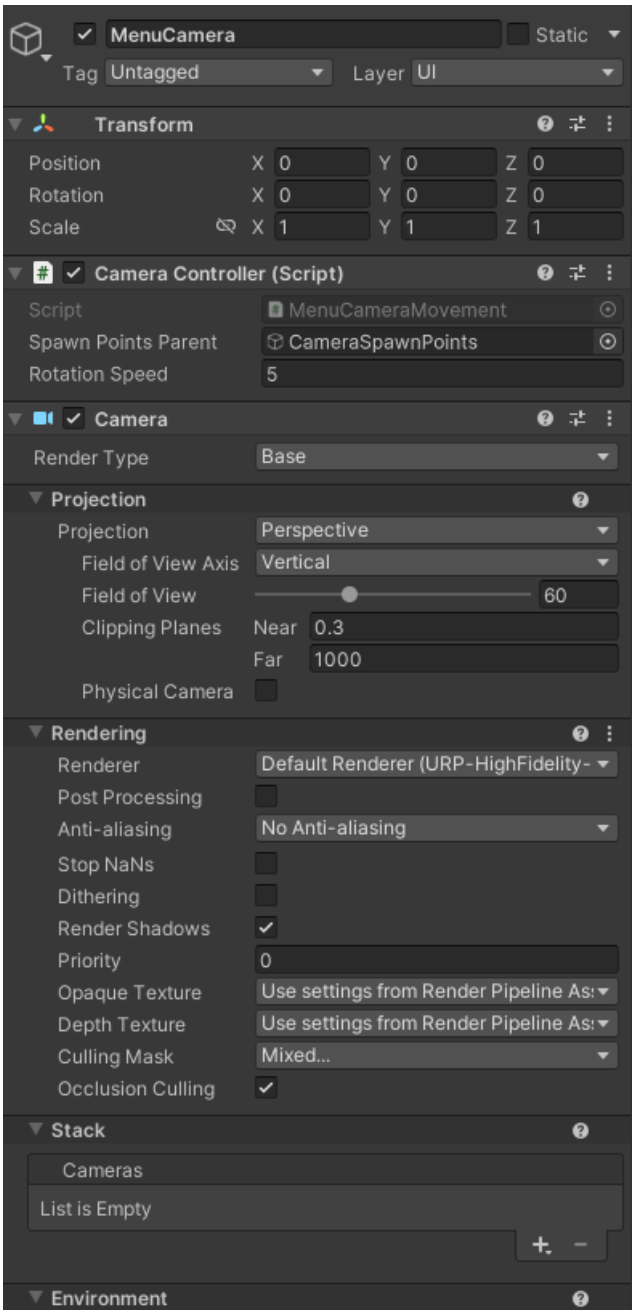


Ilustración 3.143; Componentes de la cámara del menú principal

De esta forma, cuando el jugador esté en el menú de inicio, tendrá todas las opciones del menú disponibles para seleccionar y, además, estará viendo en el fondo el mapa en el que se va a desarrollar el juego.

Ilustración 3.144; Resultado de combinar menú principal con su cámara

3.30.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para la cámara del menú principal

Test_06_06	
Nombre de la prueba o test	Rotación de la cámara del menú principal
Objetivo	La cámara debe rotar de forma que se vea el mapa
Resultado	La cámara rota desde afuera para adentro, con lo que se consigue que se vea todo el mapa.
Comentarios	Todo OK.

Tabla 3.99; Test_06_06: rotación de la cámara del menú principal

Test_06_07	
Nombre de la prueba o test	Velocidad de rotación de la cámara del menú principal
Objetivo	La cámara debe rotar a una velocidad adecuada
Resultado	La cámara rota a una velocidad la cual permite al jugador ver todo el mapa pero sin ser pesado.
Comentarios	En función de la perspectiva del jugador, este puede llegar a pensar que la rotación es muy lenta

Tabla 3.100; Test_06_07: velocidad de rotación de la cámara del menú principal

Test_06_08	
Nombre de la prueba o test	Posicionamiento de la cámara del menú principal
Objetivo	La cámara debe posicionarse en un lugar diferente cada vez que llega al final de su rotación
Resultado	La cámara, al llegar al final de su rotación, se posiciona en una ubicación diferente a la que está actualmente
Comentarios	El jugador puede llegar a pensar que esta funcionalidad puede ser “spoiler”

Tabla 3.101; Test_06_08: posicionamiento de la cámara del menú principal

3.2.7 Arma

El arma es otro de los elementos indispensables de un videojuego FPS, ya que es la herramienta del personaje principal para eliminar enemigos y avanzar en la historia del juego.

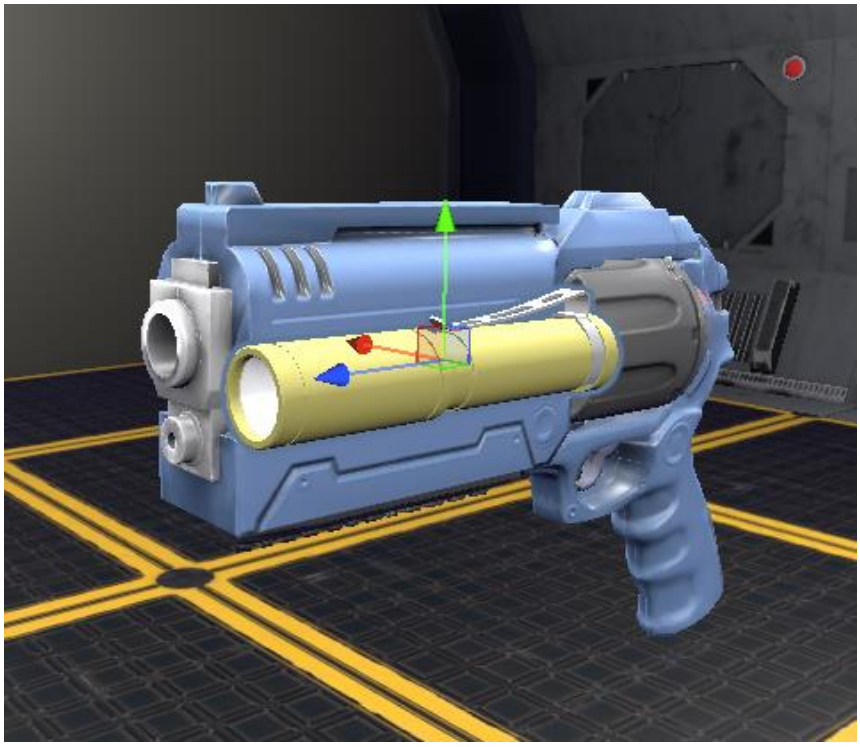


Ilustración 3.145; Modelo del arma

Los componentes de esta arma son los siguientes:

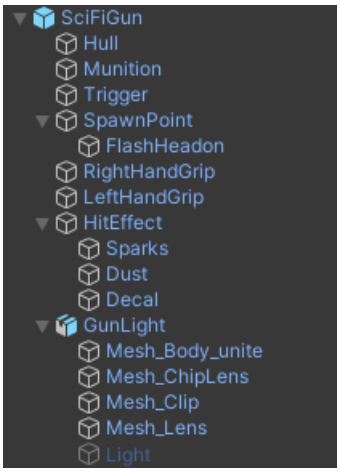


Ilustración 3.146; Elementos del arma

De entre ellos, debemos destacar el elemento llamado “GunLight”. Este será una linterna, la cual el jugador usará para poder ver en la oscuridad.

Por otro lado, este elemento tiene asociado un script llamado “WeaponRayCasting”.

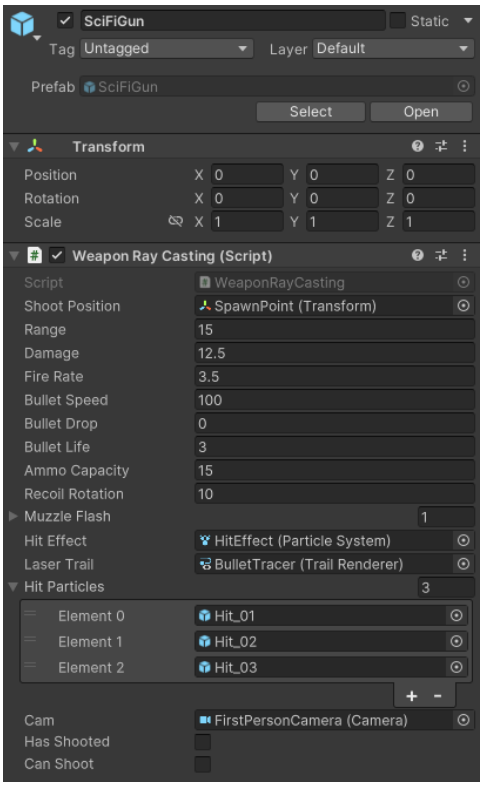


Ilustración 3.147; Componentes del arma

Este script se encarga del funcionamiento general del arma, el cual consta de distintas funcionalidades. Para explicarlas todas, se va a proceder a detallar el script y todas sus variables.

Para empezar, se crea una clase llamada “Bullet” de la cual se van a instanciar objetos que representen las balas disparadas por el arma. Los objetos instanciados de esta clase, tendrán los siguientes atributos:

1. “Time”: Tiempo que ha pasado desde que la bala fue disparada.
2. “InitialPosition”: Posición desde la que la bala va a ser disparada
3. “InitialVelocity”: Velocidad inicial de la bala al ser disparada
4. “Trail”: Referencia a un objeto de tipo “TrailRenderer”, con el cual se va a visualizar la trayectoria de la bala.
5. “DistanceTraveled”: Distancia que la bala ha recorrido desde que ha sido disparada.

Se tendrá una estructura de datos dinámica en la que se almacenen las balas instanciadas llamada “bullets”. Además, se pasarán por parámetro una serie de sistemas de partículas [91] que se almacenarán en la variable “hitParticles” para luego ser instanciados al entrar en contacto una bala con algún enemigo. Por último, se tendrá también una estructura de datos estática llamada ”muzzleFlash”, la cual almacenará los efectos de fogonazo en el bozal del arma al disparar.

Inicialmente, solo se podrá disparar si el valor de la variable booleana de control “canShoot” es “true”. Esto es una forma de controlar, desde cualquier parte de la aplicación, si el jugador puede disparar o no.

Si esta variable tiene valor “true”, lo primero que se hace es actualizar todas las balas que haya instanciadas. Con actualizar se quiere referir a actualizar todos los atributos de las balas instanciadas, como la posición de la bala, la distancia que se ha desplazado, el tiempo de vida, … Se controlará si la bala ha viajado más que el rango del arma, lo que causará la destrucción de la bala. Además, se controlará, mediante

la creación de un elemento “Raycast” [92], si la bala ha colisionado con algún elemento en ese instante de tiempo. En tal caso, se gestionará su colisión con este elemento.

Para ello, se obtiene la información del elemento colisionado. Si este elemento no tiene la etiqueta “Enemy”, no será un enemigo, por lo que, si el elemento colisionado no tiene la etiqueta “NPC”, se instancia un efecto de partículas para simular que la bala ha colisionado con algo sólido, dejando una marca en ese sitio. Por el contrario, si este elemento si tiene la etiqueta de “Enemy”, será considerado un enemigo, por lo que ahora se ha de ver qué enemigo es. Si el nombre del enemigo es “Banana_head”, se llamará al método “HitHead()” del script “HeadCollision”, del cual se hablará más adelante. Si el nombre del enemigo es “Banana_body”, se llamará al método “HitBody()” del script “BodyCollision”, del cual también se hablará más adelante. Si el nombre del enemigo es “Droid” lo que se hará será localizar todas las partículas de chispas del enemigo y activar las correspondientes en función de la vida del enemigo:

1. Si la vida del enemigo es menor del 25%, se activará solo una chispa
2. Si la vida del enemigo es menor del 50%, se activarán dos chispas
3. Si la vida del enemigo es menor del 75%, se activarán tres chispas

Todas las chispas que se activen se harán de forma aleatoria. Todo esto se realizará así para dar sensación de estado debilidad del enemigo en el combate.

Una vez que se han actualizado todas las balas instanciadas, se verifica que se ha pulsado el botón de disparar, que el tiempo entre el último disparo sea menor que la cadencia del arma y que el arma tiene munición cargadas. En tal caso, se realizan varias funcionalidades. La primera de ellas es crear una bala, dándole los valores de posicion, velocidad, tiempo y el componente “TrailRenderer” [93] iniciales. Luego, se emite el efecto del bozal del arma para simular que la bala ha sido disparada. Por último, se realiza un efecto de retroceso en el arma mediante la rotación local de 10º del arma y cuando se haya rotado, vuelta a su rotación original.

Además de comprobar esto, también se valida si el jugador intenta disparar y el arma no tiene munición cargada. En tal caso, sonará un sonido que indicará que el arma no tiene munición cargada.

Destacar el componente “TrailRenderer” que se usa para crear un efecto visual llamativo y rápido, el cual representa la bala.

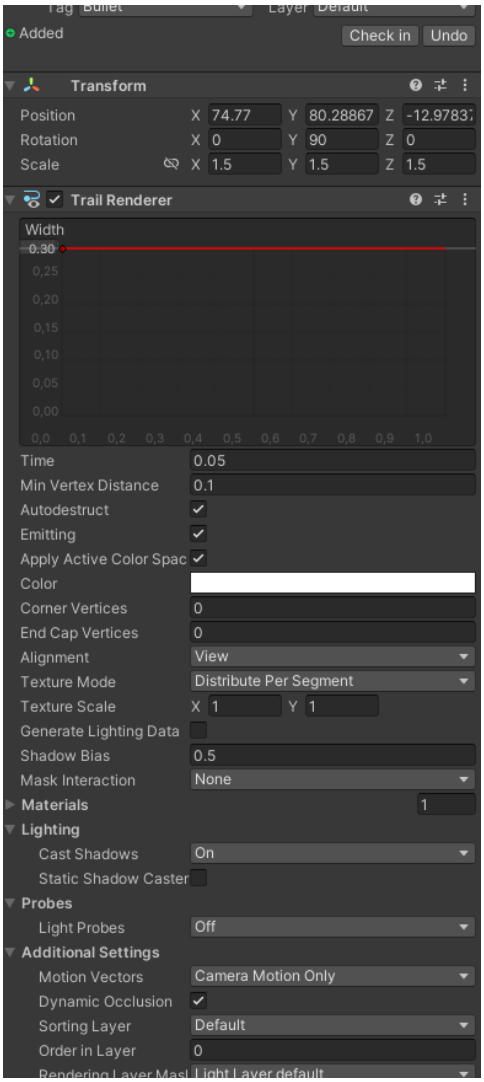


Ilustración 3.148; Componente TrailRenderer de la bala del arma

Por último, también se verá si el jugador quiere recargar. Esto solo será posible si la munición cargada del arma es menor que la capacidad del cargador del arma. En tal caso, se llamará al método “Reload()” del GameManager.

A continuación, se aporta el pseudocódigo del script:

```
Update()
  Si canShoot es verdadero
    Incrementa timeElapsed por Time.deltaTime

    UpdateBullets(Time.deltaTime)

    Si se presiona "ClickIzquierdo" y timeElapsed >= 1/fireRate y GameManager.getGunAmmoLoaded() > 0
      timeElapsed <- 0
      hasShooted <- verdadero

      velocity <- cam.transform.forward * bulletSpeed

      b <- CreateBullet(shootPosition.position, velocity)
      Añade b a bullets

      Para cada muzzle
        muzzle.Emit(1)

      GameManager.updateGunAmmo(-1)

      Recoil()

    Si se presiona "R" y GameManager.getGunAmmoLoaded() < ammoCapacity
      GameManager.reload(ammoCapacity)

UpdateBullets(deltaTime)
  Para cada bullet de todas las bullets instanciadas
    previousPosition <- GetBulletPosition(bullet)
    Incrementa bullet.time por deltaTime
    currentPosition <- GetBulletPosition(bullet)

    distanceMoved <- Distancia entre previousPosition y currentPosition
    Incrementa bullet.distanceTraveled por distanceMoved

    Si bullet.distanceTraveled > rango del arma
      Elimina bullet de bullets
      continue();

    Se crea Raycast desde previousPosition hasta currentPosition

    Si hay colisión
      HandleBulletHit(colisión, bullet)
      bullet.time <- bulletLife

    Si no
      Actualiza posición de bullet.trail a currentPosition
      Actualizamos bullet.trail con currentPosition

    Si bullet.time >= bulletLife
      Elimina bullet del array bullets y se destruye bullet

HandleBulletHit(hit, bullet)
  posicion del trail de la bala <- punto de contacto con el elemento colisionado

  Si elemento con el que la bala ha colisionado tiene etiqueta "Enemy"
    Se instancia una de las particulas de colision de forma aleatoria
    Se instancia uno de los sistemas de particulas de contacto en el punto de contacto

    Si nombre del elemento con el que la bala ha colisionado empieza con "Banana_head"
      Se coge el script "HeadCollision" asociado al elemento y se llama al método HeadCollision.HitHead()

    Si nombre del elemento con el que la bala ha colisionado empieza con "Banana_body"
      Se coge el script "BodyCollision" asociado al elemento y se llama al método BodyCollision.HitBody()

    Si nombre del elemento con el que la bala ha colisionado empieza con "Droid"
      Se gestiona la activación de chispas según la vida de enemigo

  Si no
    Posiciona hitEffect en punto de contacto
    Se inicia el sistema de particulas de contacto

CreateBullet(position, velocity)
  Se instancia un trail

  b <- nueva bala con position, velocity, time a 0 y el nuevo trail instanciado
  b.trail.AddPosition(position)

  Se devuelve b, la bala creada

GetBulletPosition(bullet)
  Calcula la posición de bullet usando su tiempo, posición inicial, velocidad inicial y caída de bala y se

Recoil()
  originalRotation <- rotacion local del arma
  Ajusta la rotación local del arma incluyendo el retroceso
  Lerp de vuelta a la rotación original
```

Ilustración 3.149; Pseudocódigo del script WeaponRaycasting

3.32.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_07_01	
Nombre de la prueba o test	Activación de la linterna del arma
Objetivo	La linterna del arma debe encenderse si el jugador pulsa tecla de activación de la linterna
Resultado	El jugador puede alternar entre encender y apagar la linterna del arma si este pulsa la tecla de encendido de la linterna.
Comentarios	Todo OK.

Tabla 3.102; Test_07_01: activación de la linterna del arma

Test_07_02	
Nombre de la prueba o test	Intensidad de luz de la linterna del arma
Objetivo	La cantidad de luz emitida por la linterna debe ser adecuada
Resultado	La luz proporcionada por la linterna permite al jugador ver en la oscuridad
Comentarios	Si el jugador enciende la linterna cuando hay luz, puede llegar a ser molesto y a no ser útil.

Tabla 3.103; Test_07_02: intensida de luz de la linterna del arma

Test_07_03	
Nombre de la prueba o test	Fogonazo del bozal del arma
Objetivo	Los efectos de fogonazo del bozal del arma deben aparecer al disparar el arma
Resultado	Al ser el arma disparada, aparecen efectos de fogonazos del bozal del arma
Comentarios	El efecto puede ser un poco grande o pequeño en función de la perspectiva del jugador y puede llegar a no apreciarse.

Tabla 3.104; Test_07_03: fogonazo del bozal del arma

Test_07_04	
Nombre de la prueba o test	Activación de partículas de colisión con el enemigo
Objetivo	Cuando una de las balas del arma impacte con algún enemigo, debe aparecer un efecto de colisión
Resultado	Las balas del arma, al impactar a un enemigo, producen un efecto de colisión que le permite al jugador saber que las balas han impactado y que se le ha hecho daño al enemigo.
Comentarios	El jugador puede pensar que, ademas de una función estética y visual, este efecto tiene alguna funcionalidad más.

Tabla 3.105; Test_07_04: activación de partículas de colisión con el enemigo

Test_07_05	
Nombre de la prueba o test	Aleatoriedad de las partículas de colisión con el enemigo
Objetivo	Los efectos que aparezcan al colisionar las balas con los enemigos deben ser aleatorios.
Resultado	Cada vez que una bala impacta en un enemigo, aparece un efecto aleatorio.
Comentarios	El jugador puede llegar a pensar que cada efecto tiene su propio significado

Tabla 3.106; Test_07_05: aleatoriedad de las partíclas de colisión con el enemigo

Test_07_06	
Nombre de la prueba o test	Activación de la bala del arma
Objetivo	Una bala debe aparecer si el jugador dispara
Resultado	Cuando el jugador dispara el arma, si esta está cargada, aparece una bala
Comentarios	El jugador puede no saber si va a aparecer o no una bala.

Tabla 3.107; Test_07_06: activación de la bala del arma

Test_07_07	
Nombre de la prueba o test	Velocidad de la bala del arma
Objetivo	La bala debe volar a una velocidad adecuada
Resultado	El tiempo que tarda una bala en llegar a colisión es adecuado
Comentarios	El jugador puede no llegar a saber el tiempo que va a tardar la bala en llegar al punto de impacto.

Tabla 3.108; Test_07_07: velocidad de la bala del arma

Test_07_08	
Nombre de la prueba o test	Rango del arma
Objetivo	Las balas disparadas por el arma deben desaparecer al llegar al rango límite del arma
Resultado	Las balas del arma, cuando salen del rango del arma, desaparecen.
Comentarios	El jugador puede no saber hasta dónde llega la bala

Tabla 3.109; Test_07_08: rango del arma

Test_07_09	
Nombre de la prueba o test	Cadencia del arma
Objetivo	El arma debe tener una cadencia
Resultado	El jugador no puede disparar continuamente, el jugador puede disparar cada cierto tiempo
Comentarios	El jugador puede llegar a no saber cuándo puede y no puede disparar

Tabla 3.110; Test_07_09: candencia del arma

Test_07_10	
Nombre de la prueba o test	Caída de la bala
Objetivo	Las balas disparadas deben tener caída

Resultado	Las balas disparadas, a mayor distancia recorrida, más han caído.
Comentarios	El jugador puede no esperarse esta funcionalidad.

Tabla 3.111; Test_07_10: caída de la bala

Test_07_11	
Nombre de la prueba o test	Caída controlada de la bala
Objetivo	La caída de las balas disparadas debe ser adecuada
Resultado	Las balas disparadas tienen una caída adecuada, no son un láser con un disparo totalmente recto, ni son un misil, con muchísima caída.
Comentarios	El jugador puede necesitar controlar y tener en cuenta esta caída a la hora de disparar y acertar

Tabla 3.112; Test_07_11: caída controlada de la bala

Test_07_12	
Nombre de la prueba o test	Daño al impactar con el enemigo
Objetivo	Las balas deben dañar al enemigo con el que colisiona
Resultado	Las balas, al impactar a un enemigo, le aplica daño a este enemigo
Comentarios	El jugador puede no saber exactamente cuánto daño le va a aplicar al enemigo

Tabla 3.113; Test_07_12: daño al impactar con el enemigo

Test_07_13	
Nombre de la prueba o test	Salida de la bala del arma
Objetivo	La bala disparada debe salir desde la boca del arma
Resultado	La bala sale disparada desde la boca del arma
Comentarios	Todo OK.

Tabla 3.114; Test_07_13: salida de la bala del arma

Test_07_14	
Nombre de la prueba o test	Dirección vuelo de la bala
Objetivo	La bala debe volar en la dirección a la que ha sido disparada
Resultado	La bala vuela hacia delante
Comentarios	El jugador puede llegar a pensar que las balas disparadas no salen para adelante, sino que tienen dispersión como una escopeta.

Tabla 3.115; Test_07_14: dirección de vuelo de la bala

Test_07_15	
Nombre de la prueba o test	Impacto en cuerpo o cabeza
Objetivo	Se debe poder detectar si el impacto de la bala ha sido en la cabeza o en el cuerpo
Resultado	La bala, en función de en donde impacte, realiza una acción u otra
Comentarios	El jugador puede no llegar a saber esta distinción.

Tabla 3.116; Test_07_15: impacto en cuerpo o cabeza

Test_307_16	
Nombre de la prueba o test	Impacto en sólido
Objetivo	La bala debe dejar una marca si impacta en sólido
Resultado	La bala, si impacta en sólido como pared, suelo o techo, deja una marca.
Comentarios	El jugador puede no saber si este efecto visual tiene alguna otra funcionalidad o no.

Tabla 3.117; Test_07_16: impacto en sólido

Test_07_17	
Nombre de la prueba o test	Impacto en sólido desaparece

Objetivo	La marca dejada por la bala al impactar en sólido debe irse tras un tiempo
Resultado	La marca desaparece después de un tiempo tras haber impactado
Comentarios	Todo OK.

Tabla 3.118; Test_07_17: impacto en sólido desaparece

Test_07_18	
Nombre de la prueba o test	Poder disparar
Objetivo	El jugador debe poder disparar
Resultado	El jugador dispara su arma si esta está cargada
Comentarios	Puede darse casos en los que el arma no dispare y el jugador no sepa por qué

Tabla 3.119; Test_07_18: poder disparar

Test_07_19	
Nombre de la prueba o test	Activación de chispas
Objetivo	Deben aparecer chispas en el cuerpo de un dron de combate tras haberle quitado parte de la vida
Resultado	Tras haber dañado lo suficiente a un dron de combate, aparecen chispas que indican la debilidad del enemigo al estar dañado.
Comentarios	El jugador puede no saber si este efecto tiene algún motivo más a parte del visual

Tabla 3.120; Test_07_19: activación de chispas

Test_07_20	
Nombre de la prueba o test	Activación de chispas en función de la vida
Objetivo	Deben aparecer chispas en el cuerpo de un dron de combate en función de la vida

Resultado	El número de efectos de chispas que aparecen en el cuerpo del enemigo varía en función de la cantidad de vida que se le ha quitado al enemigo.
Comentarios	El jugador asociará el número de efectos a la debilidad del enemigo, por lo que puede no saber cuánto le falta al enemigo para ser derrotado.

Tabla 3.121; Test_07_20: activación de chispas en función de la vida

Test_07_21	
Nombre de la prueba o test	Activación aleatoria de las chispas
Objetivo	Las chispas deben aparecer de forma aleatoria
Resultado	La ubicación de aparición de las chispas es totalmente aleatoria
Comentarios	Todo OK.

Tabla 3.122; Test_07_21: activación aleatoria de chispas

Test_07_22	
Nombre de la prueba o test	Retroceso del arma
Objetivo	El arma debe tener un efecto de retroceso visible
Resultado	Se puede apreciar un efecto de retroceso del arma al disparar
Comentarios	El jugador puede no saber si este efecto va a afectar a su disparo o no. Además, la linterna, al estar pegada al arma, se mueve junto con el efecto de retroceso del arma, por lo que la luz, la cual se mueve también al ser movida la linterna por el retroceso, puede llegar a confundir e incluso a distraer al jugador.

Tabla 3.123; Test_07_22: retroceso del arma

Test_07_23	
Nombre de la prueba o test	Cantidad de retroceso del arma
Objetivo	Cantidad de movimiento del arma por su retroceso debe ser adecuado

Resultado	El movimiento del arma por el retroceso es adecuado y apreciable
Comentarios	Todo OK.

Tabla 3.124; Test_07_23: cantidad de retroceso del arma

Test_07_24	
Nombre de la prueba o test	Momento de retroceso del arma
Objetivo	Se debe aplicar el efecto de retroceso del arma justo cuando esta dispara
Resultado	El arma sufre un movimiento de retroceso justo cuando esta es disparada.
Comentarios	El jugador puede no esperarse este efecto y distraerle.

Tabla 3.125; Test_07_24: momento de retroceso del arma

Test_07_25	
Nombre de la prueba o test	No poder disparar
Objetivo	El jugador no debe poder disparar si no tiene munición cargada en su arma
Resultado	Si el arma no tiene munición cargada, no puede disparar
Comentarios	El jugador puede no saber por qué el arma no dispara

Tabla 3.126; Test_07_25: no poder disparar

Test_07_26	
Nombre de la prueba o test	Recarga del arma
Objetivo	El jugador debe poder recargar el arma
Resultado	El jugador puede recargar su arma para tener munición cargada, con la que disparar
Comentarios	El jugador puede no saber cuánta munición se va a recargar en su arma.

Tabla 3.127; Test_07_26: recarga del arma

Test_07_27	
Nombre de la prueba o test	No poder recargar
Objetivo	El arma no debe poder recargarse en ciertas situaciones
Resultado	El arma solo se puede recargar si esta no tiene cargada la cantidad máxima de munición del cargador y si tiene munición suficiente guardada
Comentarios	Puede darse casos en los que el jugador no sepa por qué no puede recargar el arma

Tabla 3.128; Test_07_27: no poder recargar

3.2.8 Enemigos

Los enemigos forman una parte indispensable de un juego FPS ya que estos son los encargados de dificultar el juego. Se va a detallar el funcionamiento de los distintos enemigos del juego.

3.2.8.1 Banana-man

Este enemigo no es ofensivo, simplemente está para que el jugador practique y vea cómo es el sistema de combate y de disparo. Por estos motivos, este enemigo se encontrará en el tutorial y será indestructible.

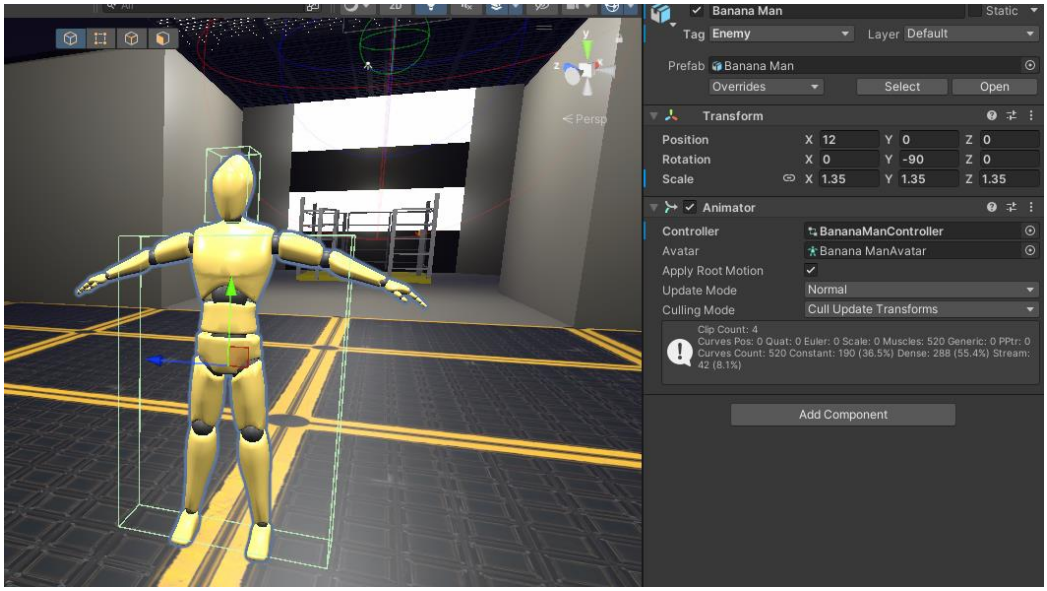


Ilustración 3.150; Componentes del enemigo Banana-man

Este enemigo tendrá un componente “Animator” el cual estará regido por el componente “BananaManController” el cual tendrá esta estructura:

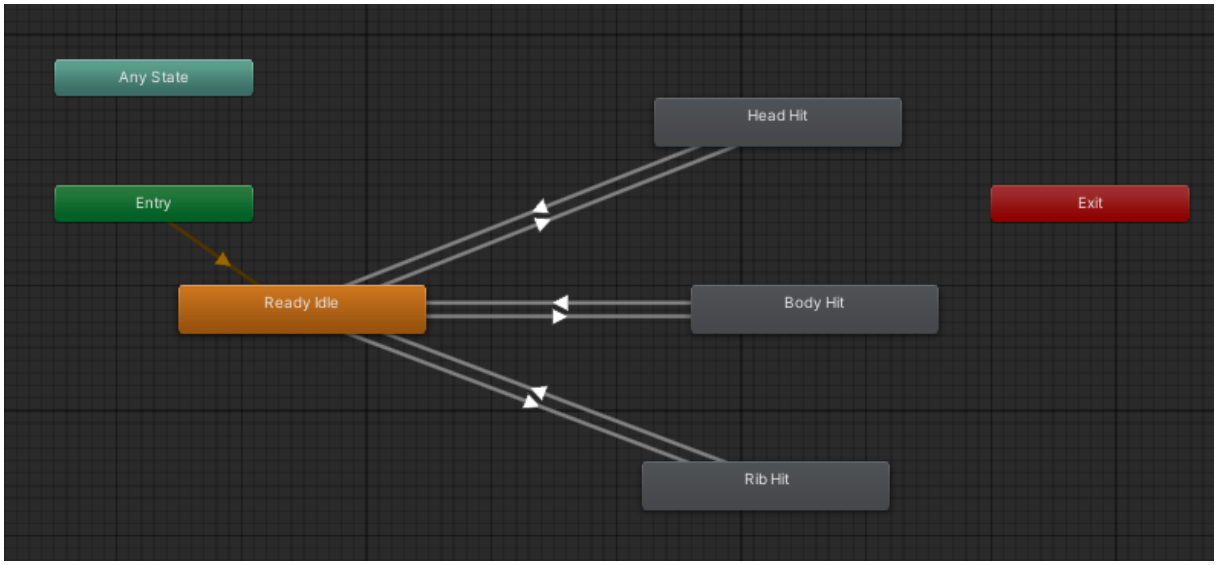


Ilustración 3.151; AnimatorController del enemigo Banana-man

El enemigo estará constantemente realizando la animación “Ready Idle”. No obstante, se podrán dar estos casos:

1. Pasará a la animacion “Head Hit” cuando el parámetro booleano “head” esté a “true” y luego volverá a la animación “Ready Idle” al ponerse el parámetro a valor “false”
2. Pasará a la animacion “Body Hit” cuando el parámetro booleano “body” esté a “true” y luego volverá a la animación “Ready Idle” al ponerse el parámetro a valor “false”
3. Pasará a la animacion “Rib Hit” cuando el parámetro booleano “ribs” esté a “true” y luego volverá a la animación “Ready Idle” al ponerse el parámetro a valor “false”

El enemigo tiene como hijos dos elementos destacables.

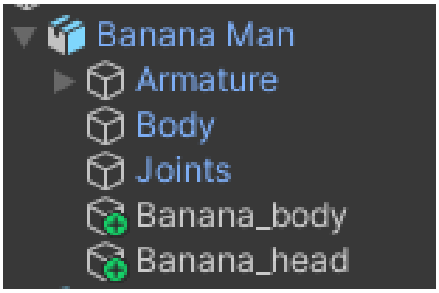


Ilustración 3.152; Elementos del enemigo Banana-man

El primero es “Banana_body”, el cual hace referencia al **cuerpo** del enemigo. Este elemento tiene un componente “BoxCollider” asociado. Además, tiene un script llamado “BodyCollision” asociado. Este script lo que hace es que, cuando el método “HitBody()” es llamado (desde el método “HandleHit()” del script “WeaponRaycasting”, como se ha comentado antes), lo que se hace es poner a valor “true” el parámetro “ribs” o “body” del “AnimatorController” en función de si previamente se ha activado la animacion “Body Hit” o “Rib Hit”. Esto se hace para que, cuando se golpee el cuerpo del enemigo, una vez se realice la animacion de “Body Hit” y que, a la siguiente vez, se haga la animación de “Rib Hit” de forma alterna.

El segundo es “Banana_head”, el cual hace referencia a la **cabeza** del enemigo. Este elemento funciona prácticamente igual al anterior, con la diferencia de que varía el parámetro de control del componente “AnimatorController”, en este caso es “head”, y que la animación siempre será “Head Hit”.

3.34.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del enemigo “Banana-man”.

Test_08_01	
Nombre de la prueba o test	Animaciones adecuadas
Objetivo	Las animaciones realizadas en las diferentes situaciones deben ser acordes a la situación y tienen que reflejar la acción del enemigo

Resultado	Las animaciones reflejan exactamente lo que está haciendo el enemigo
Comentarios	Todo OK.

Tabla 3.129; Test_08_01: animaciones adecuadas

Test_08_02	
Nombre de la prueba o test	Animaciones en función del lugar del impacto
Objetivo	El enemigo debe realizar una animación u otra en función de la parte en la que reciba un impacto
Resultado	Si el enemigo recibe un impacto en la cabeza, realiza una animacion. Si el enemigo recibe un impacto en el cuerpo, realiza otra animación.
Comentarios	El jugador puede llegar a pensar que existe alguna diferencia al disparar a una parte del enemigo o a otra.

Tabla 3.130; Test_08_02: animaciones en funcion del lugar del impacto

Test_08_03	
Nombre de la prueba o test	Entidad inmortal
Objetivo	La entidad debe ser considerada como un enemigo pero no debe poder ser eliminado
Resultado	Por más que el jugador dispare al enemigo, este no será eliminado.
Comentarios	Al ser el primer enemigo con el que se encuentra el jugador, este puede llegar a pensar que el enemigo puede ser eliminado si le dispara mucho, o incluso, puede llegar a pensar que puede dañarlo al dispararle.

Tabla 3.131; Test_08_03: entidad inmortal

3.2.8.2 Dron de combate

Este enemigo, cuyo modelo es el del droide de batalla B1 de StarWars, solo usará ataques cuerpo a cuerpo para dañar al jugador.

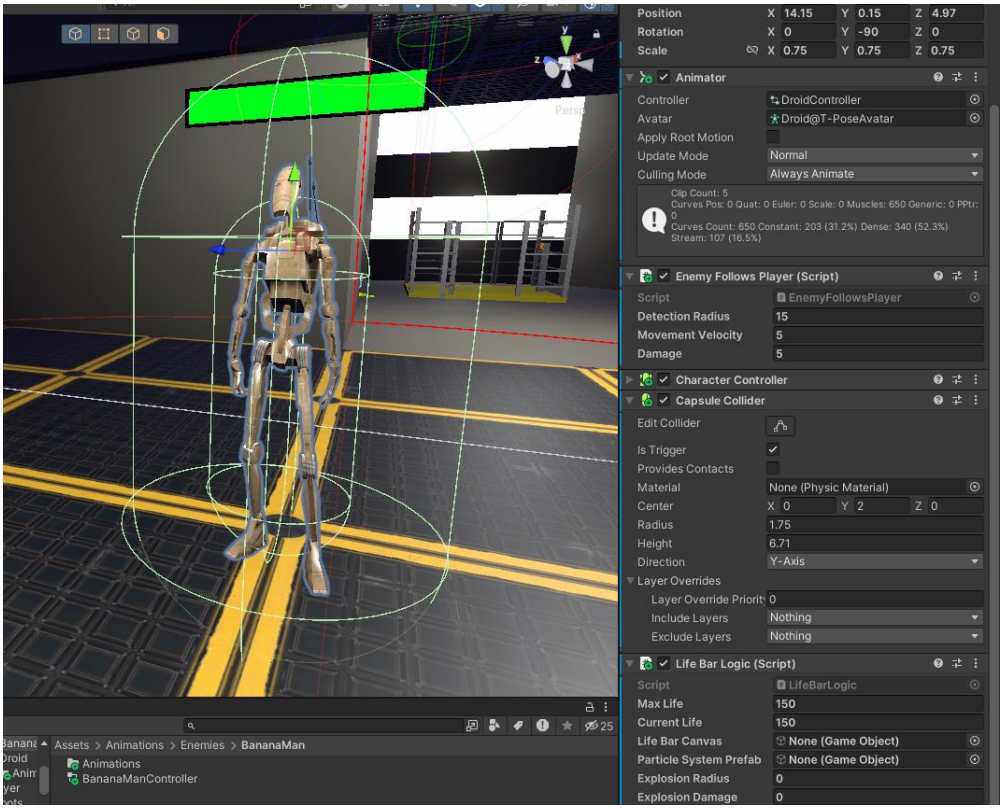


Ilustración 3.153; Componentes del enemigo Dron de combate

El enemigo tiene asociado un componente “Animator” regido por el componente “DroidController”, el cual maneja el flujo de animaciones del objeto. Aquí se muestra su estructura.

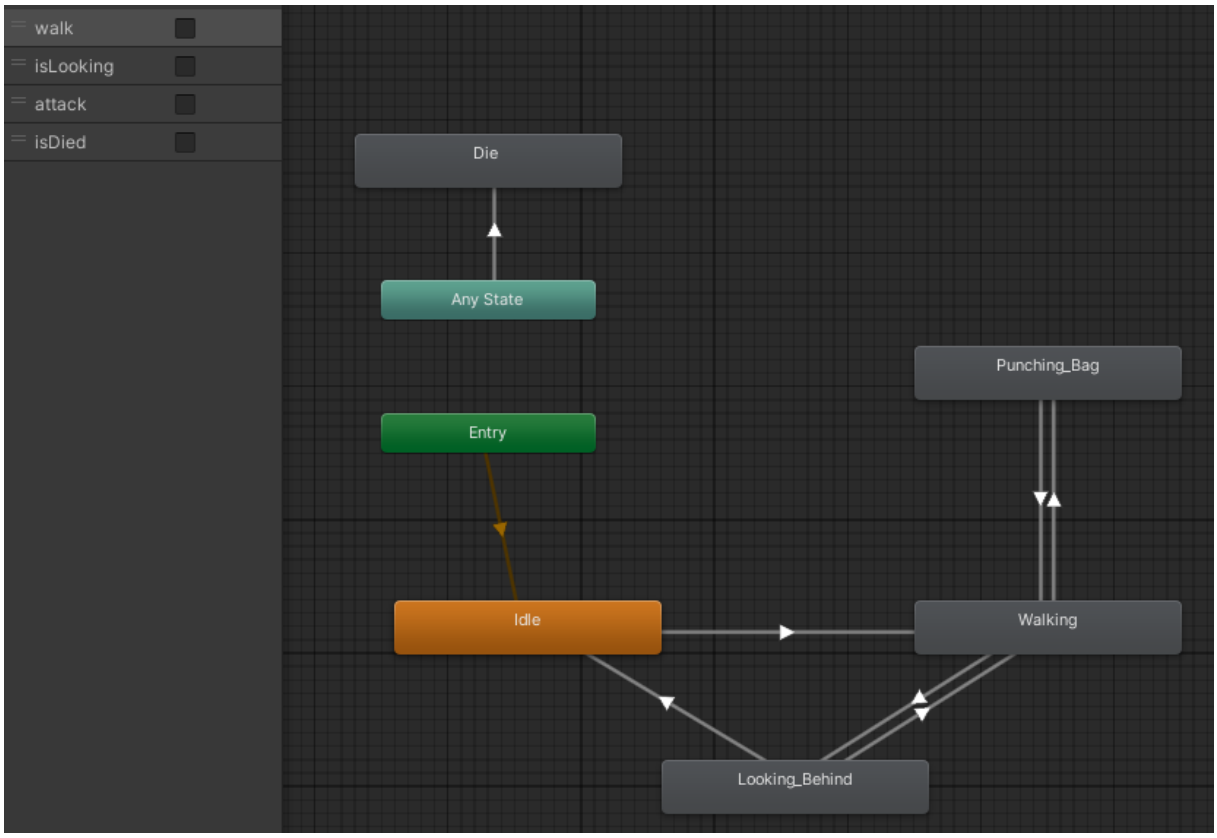


Ilustración 3.154; AnimatorController del enemigo Dron de combate

Este enemigo realizará la animación “Idle” desde el principio. Cuando el parámetro booleano “walk” se ponga a valor “true”, pasará a realizar la animación “Walking” para andar. Estando en “Walking”, si el parámetro booleano “attack” se pone a valor “true”, pasará a realizar la animación “Punching_bag”, con la que atacará al jugador, y si este parámetro se pone a valor “false”, volverá a “Walking”. Desde “Walking” también podrá realizar la animación “Looking_Behind”, como simulando que está buscando al jugador, si el parámetro “isLooking” tiene valor “true”, y volverá a “Walking” si el parámetro “isLooking” tiene valor “false” y el parámetro “walk” tiene valor “true”. Finalmente, estando en “Looking_Behind”, si el parámetro “isLooking” tiene valor “false”, se realizará de nuevo la animación “Idle”. También comentar que, en cualquier momento del flujo de la animación, si el parámetro booleano “isDied” toma el valor “true”, el enemigo realizará la animación “Die” para dar a entender que el enemigo ha muerto.

Todos los parámetros del “AnimatorController” serán controlados por el script “EnemyFollowsPlayer”, del cual hablaremos a continuación.

El enemigo, para gestionar su lógica, tiene asociado le script “EnemyFollowsPlayer”. Con este script, se controlan varias cosas. Primeramente, si el enemigo ha muerto, en tal caso, se actualiza el valor del parámetro “isDied” del “AnimatorController” a valor “true”. Si el enemigo no está muerto, se crea una esfera de detección con el método “CheckSphere()” [94] de la clase “Physics” [95] y con un radio determinado. Si el jugador entra dentro de la esfera de detección, se pondrá a “true” el parámetro “walk” del “AnimatorController”, se hará que el enemigo mire constantemente al jugador y se actualizará la posición del componente “CharacterController” del enemigo para que el enemigo se vaya aproximando al jugador.

El enemigo tendrá un componente “CapsuleCollider” con la opción “isTrigger” marcada, por lo que el script “EnemyFollowsPlayer” tendrá un método “OnTriggerEnter()” en el que se activará la animación de atacar, al poner a valor “true” el parámetro “attack” del “AnimatorController”.

También se declarará un método público llamado “Attack()”, con el cual se le aplicará un daño al jugador.

Por último, se usa el método “OnDrawGizmos()” [96] para dibujar una esfera con el radio de detección del enemigo en la ventana de edición de la escena y así poder ver cómo el jugador entra en contacto con la esfera creada con el método “CheckSphere()” de la clase “Physics”, como se ha comentado antes.

Además del script “EnemyFollowsPlayer”, este enemigo también tiene asignado el script “LifeBarLogic”. Como este script se ha comentado previamente en el **apartado 3.2.5.2.2 “LifeBarLogic”**, no se va a volver a detallar puesto que tiene el mismo funcionamiento.

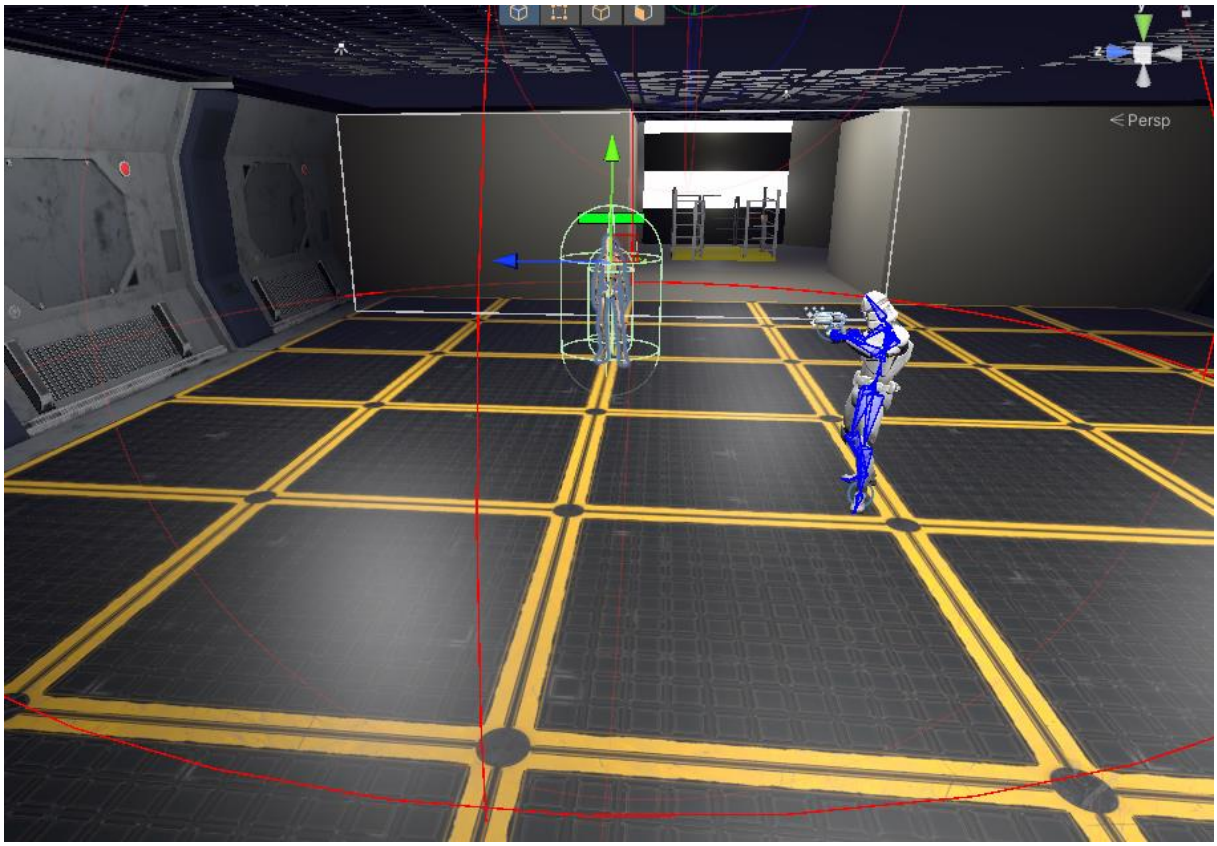


Ilustración 3.155; Área de detección del enemigo Dron de combate

Para hacer daño al jugador, lo que se ha hecho es añadir un evento [97] en la animación de ataque del enemigo, la animación “Punching_Bag”:

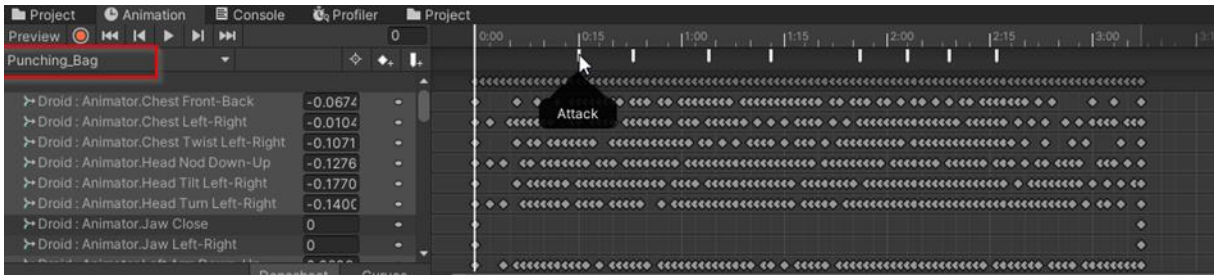


Ilustración 3.156; Eventos dentro de la animación “Punching_Bag” del enemigo Dron de combate

De esta forma, en los “frames” de la animación marcados, se llamará al método “Attack()” del script “EnemyFollowsPlayer”, con lo que se consigue sincronizar el golpe con la actualización de la vida del jugador.

Ilustración 3.157; Elementos de las chispas del enemigo Dron de combate

Comentar también que este enemigo tiene un elemento llamado “SparksPositions” que almacena diversos sistemas de partículas de chispas, ubicados en diferentes partes del cuerpo del enemigo, inicialmente desactivados.

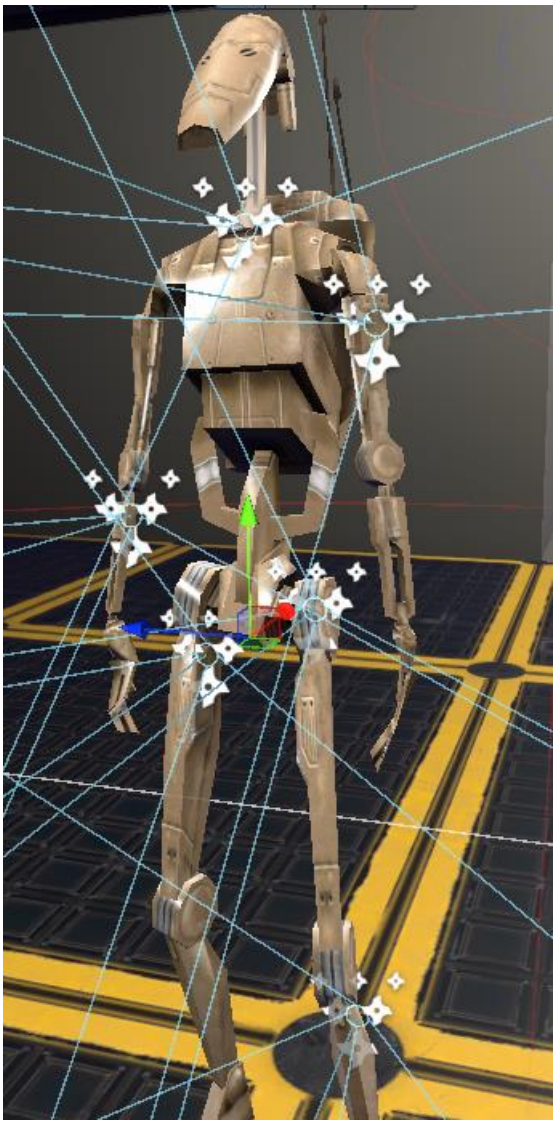


Ilustración 3.158; Posicion de las chispas del enemigo Dron de combate

Lo que se intenta es que, como se ha comentado en el **apartado 3.2.7 Arma**, en función de la vida del enemigo, se activen unos sistemas de partículas de chispas u otros, de forma aleatoria.

3.36.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Animaciones adecuadas
2. Barra de vida refleja daño recibido
3. Color de barra de vida

Test_08_04	
Nombre de la prueba o test	Flujo de animaciones adecuado
Objetivo	El flujo de animaciones del enemigo debe ser adecuado
Resultado	En función de la situación en la que se encuentre el enemigo, este realizará unas animaciones u otras
Comentarios	Todo OK.

Tabla 3.132; Test_08_04: flujo de animaciones adecuado

Test_08_05	
Nombre de la prueba o test	Enemigo va hacia el jugador
Objetivo	El enemigo debe moverse en dirección del jugador cuando este lo detecte
Resultado	El enemigo, cuando detecte al jugador, irá hacia él.
Comentarios	Todo OK.

Tabla 3.133; Test_08_05: enemigo va hacia el jugador

Test_08_06	
Nombre de la prueba o test	Enemigo mira al jugador
Objetivo	El enemigo, al detectar al jugador, aparte de ir hacia él, debe mirarlo constantemente
Resultado	El enemigo mira al jugador independientemente de la posición o perspectiva del jugador.
Comentarios	Todo OK.

Tabla 3.134; Test_08_06: enemigo mira al jugador

Test_08_07	
Nombre de la prueba o test	Movimiento sin variar Y
Objetivo	Al derrotar a una entidad llamada “Cyborg” o “Droid”, se debe instanciar una explosión
Resultado	La explosión es instanciada al derrotar a una entidad cuyo nombre sea “Cyborg” o “Droid”.
Comentarios	El jugador puede no esperarse la explosión o puede no saber en qué afecta la explosión instanciada.

Tabla 3.135; Test_08_07: movimiento sin variar Y

Test_08_08	
Nombre de la prueba o test	Giro solo en eje Y
Objetivo	La explosión instanciada al derrotar a una entidad llamada “Cyborg” o “Droid”, debe hacer daño a los enemigos cercanos
Resultado	La explosión instanciada aplica daño a los enemigos cercanos.
Comentarios	El jugador puede no anticipar o no saber que la explosión hace daño a los enemigos cercanos y puede no saber con precisión lo cerca que debe estar un enemigo para que le afecte la explosión.

Tabla 3.136; Test_08_08: girar solo en eje Y

Test_08_09	
Nombre de la prueba o test	Detección del jugador
Objetivo	El jugador debe poder ser detectado por el enemigo
Resultado	El jugador, a una distancia determinada, es detectado por el enemigo
Comentarios	El jugador no sabe a qué distancia es detectado por el enemigo.

Tabla 3.137; Test_08_09: detección del jugador

Test_08_10	
Nombre de la prueba o test	Daño por golpe
Objetivo	El enemigo dañará al jugador en cada golpe que este le aseste
Resultado	El enemigo infringe daño al jugador en cada golpe que este recibe del enemigo y además, justo cuando el golpe del enemigo golpea al jugador.
Comentarios	El jugador no sabe cuánta vida le quita su ataque.

Tabla 3.138; Test_08_10: daño por golpe

Test_08_11	
Nombre de la prueba o test	Barra de vida mirando siempre a la cámara del jugador
Objetivo	La barra de vida del enemigo debe mirar siempre a la cámara.
Resultado	Independientemente de la posición o perspectiva del jugador, la barra de vida del enemigo siempre está mirando hacia la cámara del jugador.
Comentarios	Todo OK.

Tabla 3.139; Test_08_11: barra de vida mirando siempre a la cámara del jugador

Test_08_12	
Nombre de la prueba o test	Enemigo no desaparece al morir
Objetivo	El enemigo no debe desaparecer al morir

Resultado	Cuando la vida del enemigo llegue a cero, este no desaparece, si no que su “cadáver” queda en el suelo
Comentarios	El jugador puede llegar a pensar que el enemigo se puede volver a levantar, lo que le lleve a tener que estar pendiente del cadáver del enemigo y no de lo que le rodea.

Tabla 3.140; Test_08_12: enemigo no desaparece al morir

3.2.8.3 Dron terrestre

Este enemigo va por tierra y solo tiene ataques a distancia.

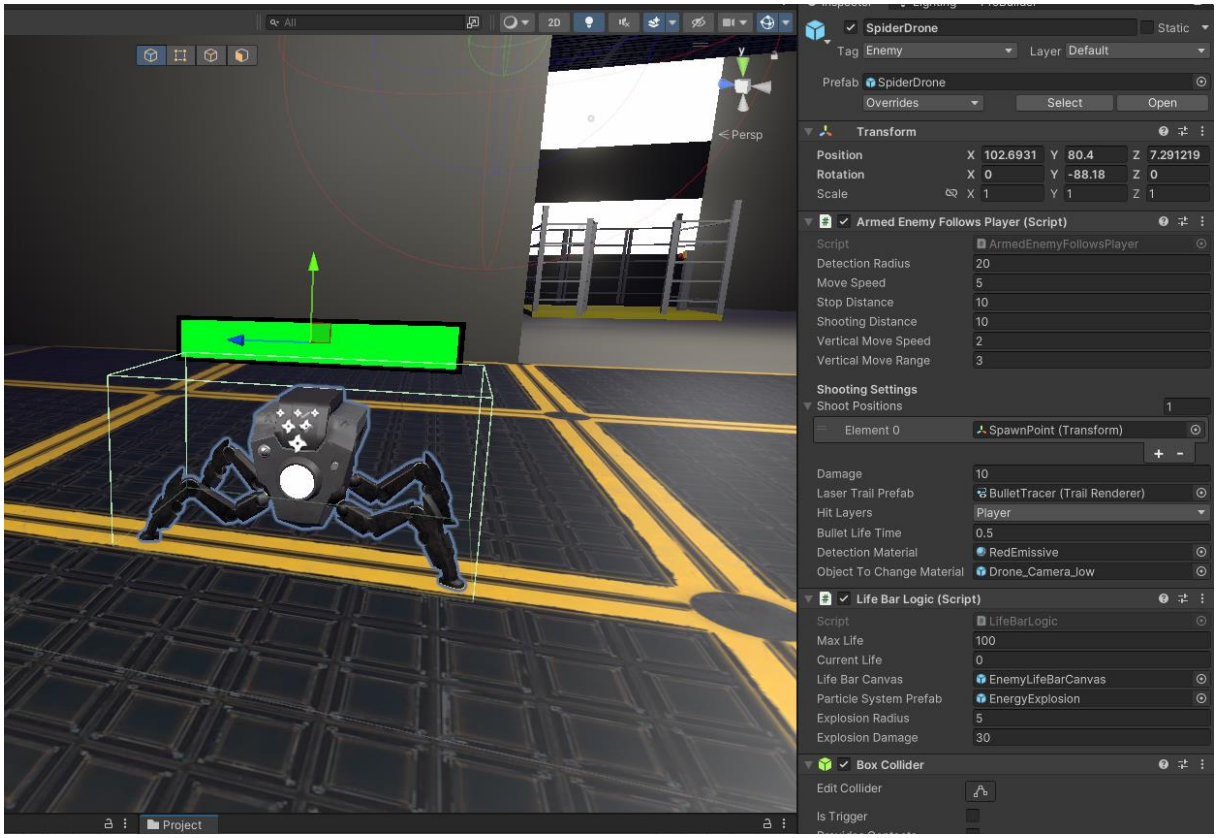


Ilustración 3.159; Componentes del enemigo Dron terrestre

A parte de un componente “BoxCollider”, este modelo tiene dos scripts asociados.

El primero a comentar es el script “LifeBarLogic”, del que ya se ha comentado en el **apartado 3.2.5.2.2 “LifeBarLogic”.** No obstante, se ha de comentar que al script asociado a este enemigo si se le ha pasado un sistema de partículas, por lo que se

instanciará ese sistema de partículas en el momento de la muerte del enemigo, acompañado con un daño a los enemigos cercanos en ese momento.

El otro es el que se llama “ArmedEnemyFollowsPlayer”. Este script, primero calcula la distancia del jugador con el enemigo y controla si el jugador está dentro del rango de detección del enemigo.

Si el jugador ha estado dentro del rango de detección del enemigo, pero ya ha salido de su rango, el material de la luz del enemigo cambia a su color original, el blanco.

Por el contrario, si el jugador está dentro del rango de detección del enemigo, se cambia el material de la luz del enemigo a rojo, lo que le indicará al jugador que el enemigo le ha detectado. Además, se hará al enemigo mirar constantemente al jugador mientras que este esté dentro del rango de detección del enemigo mediante la rotación de su cuerpo, pero solo en el eje Y para evitar que el cuerpo del enemigo gire en el eje X y/o Z y su cuerpo se meta dentro del suelo.

Ilustración 3.160; Dron terrestre cuando ha detectado al jugador

Ahora se verificará si el jugador está dentro del rango de detección, pero no dentro del rango de ataque a distancia del enemigo. En tal caso, él enemigo se moverá hacia el jugador, pero sin variar su posicion en el eje Y, para evitar que el enemigo "vuele". En caso contrario, el enemigo disparará. Para ello, lo que se hace es que, al igual que con el script "WeaponRaycasting", se instancia un "TrailRenderer" para visualizar la bala y se crea un objeto "Raycast" para ver si la bala del enemigo ha golpeado al jugador en su trayectoria y así poder aplicarle daño. También se verificará si la bala enemiga ha golpeado al jugador al final de su tiempo de vida. Para ello, se guarda la posición del jugador en el momento del disparo y, si la distancia entre esa posición con la posición del jugador cuando el tiempo de vida de la bala acaba, es menor que una cantidad determinada, también se le aplicará daño al jugador.

Se aporta pseudocódigo del script:

```
Update()
  distanceToPlayer <- calcula distancia entre posicion del enemigo y player

  Se crea esfera de detección con Physics.CheckSphere()
  Variable booleana isPlayerInRange es true si Physics.CheckSphere() detecta player y false si no

  Si nombre del enemigo empieza con "Flying"
    newYPosition <- oscila en el eje Y usando Sin de Time.time * verticalMoveSpeed) * verticalMoveRange +
    Actualiza posición en Y del enemigo en función de newYPosition

  Si isPlayerInRange y no playerDetected
    playerDetected <- verdadero
    ChangeMaterial(detectionMaterial)

  Si no es isPlayerInRange y playerDetected
    playerDetected <- falso
    RestoreOriginalMaterial()

  Si playerDetected
    LookAtPlayerOnlyYAxis()

    Si distanceToPlayer > stopDistance y distanceToPlayer > shootingDistance
      MoveTowardsPlayer(distanceToPlayer)

    Si distanceToPlayer <= shootingDistance y Time.time - lastShotTime >= shootingInterval
      Fire()
      lastShotTime <- Time.time

Fire()
  Si no hay posiciones de disparo, return;

  currentShootPosition <- obtiene el punto de disparo actual de shootPositions
  direction <- calcula dirección normalizada desde currentShootPosition a player

  Se crea raycast para ver si hay obstáculo entre currentShootPosition y direction hasta shootingDistance

  Si raycast golpea algo y es el player
    Se instancia trail de laserTrailPrefab en currentShootPosition
    Se inicia SpawnTrail con el punto golpeado y la posición del jugador en el momento del disparo

  Si hay más posiciones de disparo, se cambia a la siguiente posicion

LookAtPlayerOnlyYAxis()
  Se calcula dirección a player eliminando componente en Y
  Se almacena rotación original
  Se rota hacia player solo en eje Y
  Se restablece rotación excepto en eje Y

MoveTowardsPlayer(distanceToPlayer)
  Si distanceToPlayer es mayor a stopDistance
    Se calcula dirección de movimiento normalizada hacia player eliminando componente en Y
    Se actualiza posición del enemigo en la dirección de movimiento

SpawnTrail(trail, endPoint, playerPositionAtFireTime)
  Mientras tiempo < bulletLifeTime
    Si trail no existe, break;
    Si trail existe, se interpola posición de trail hacia endPoint

  Si trail existe cuando tiempo >= bulletLifeTime
    posicion de trail <- endPoint
    Se destruye el trail.

    Si player está cerca de la posicion en la que estaba justo cuando el enemigo disparó
      Se aplica daño al jugador

ChangeMaterial(newMaterial)
  Se cambia material de la luz del enemigo a newMaterial

RestoreOriginalMaterial()
  Se restaura material original de la luz del enemigo
```

Ilustración 3.161; Pseudocódigo del script ArmedEnemyFollowsPlayer

3.38.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Detección del jugador
2. Barra de vida refleja daño
3. Enemigo hacia el jugador
4. Enemigo mira al jugador
5. Color barra de vida
6. Barra de vida siempre mirando a la camara del jugador
7. Eliminar barra de vida
8. Movimiento sin variar Y
9. Giro solo en eje Y

Test_08_13	
Nombre de la prueba o test	Instanciar explosión
Objetivo	Al derrotar a una entidad llamada “Cyborg” o “Droid”, se debe instanciar una explosión
Resultado	La explosión es instanciada al derrotar a una entidad cuyo nombre sea “Cyborg” o “Droid”.
Comentarios	El jugador puede no esperarse la explosión o puede no saber en qué afecta la explosión instanciada.

Tabla 3.141; Test_08_13: instanciar explosión

Test_08_14	
Nombre de la prueba o test	Daño de la explosión

Objetivo	La explosión instanciada al derrotar a una entidad llamada “Cyborg” o “Droid”, debe hacer daño a los enemigos cercanos
Resultado	La explosión instanciada aplica daño a los enemigos cercanos.
Comentarios	El jugador puede no anticipar o no saber que la explosión hace daño a los enemigos cercanos y puede no saber con precisión lo cerca que debe estar un enemigo para que le afecte la explosión.

Tabla 3.142; Test_08_14: daño de la explosión

Test_08_15	
Nombre de la prueba o test	Materiales adecuados
Objetivo	El material asociado al enemigo debe concordar con el estado en el que se encuentre el enemigo
Resultado	Si el enemigo ha detectado al jugador, su material es de color rojo, pero si no lo ha detectado, su material es el original.
Comentarios	El jugador puede no saber qué significa cada material.

Tabla 3.143; Test_08_15: materiales adecuados

Test_08_16	
Nombre de la prueba o test	Cambio de materiales
Objetivo	El material asociado al enemigo debe cambiar en función del estado del enemigo
Resultado	Si el enemigo ha detectado al jugador, su material cambia y si lo pierde de vista, vuelve al material original.
Comentarios	El jugador puede no saber qué significa cada material.

Tabla 3.144; Test_08_16: cambio de materiales

Test_08_17	
Nombre de la prueba o test	Disparo al jugador

Objetivo	El enemigo debe disparar al jugador si este entra en el rango de disparo del enemigo
Resultado	Si el jugador entra dentro del rango de disparo del enemigo, este es disparado por el enemigo
Comentarios	El jugador puede no anticipar o no saber el rango de disparo del enemigo.

Tabla 3.145; Test_08_17: disparo al jugador

Test_08_18	
Nombre de la prueba o test	Instancia de bala del enemigo
Objetivo	Se debe instanciar una bala cada vez que el enemigo dispara al jugador
Resultado	Si el enemigo dispara al jugador, se instancia una bala.
Comentarios	El jugador puede no anticipar o no saber la velocidad y/o trayectoria de la bala del enemigo

Tabla 3.146; Test_08_18: instancia de bala del enemigo

Test_08_19	
Nombre de la prueba o test	Bala del enemigo daña al jugador
Objetivo	La bala del enemigo debe dañar al jugador si esta colisiona con él
Resultado	La bala del enemigo daña al jugador si la bala le alcanza
Comentarios	El jugador puede no saber cuánta vida le reduce o en qué le afecta que la bala le haya dado

Tabla 3.147; Test_08_19: bala del enemigo daña al jugador

3.2.8.4 Dron aéreo

Este es el único enemigo aéreo del juego. No obstante, solo tiene ataques a distancia.

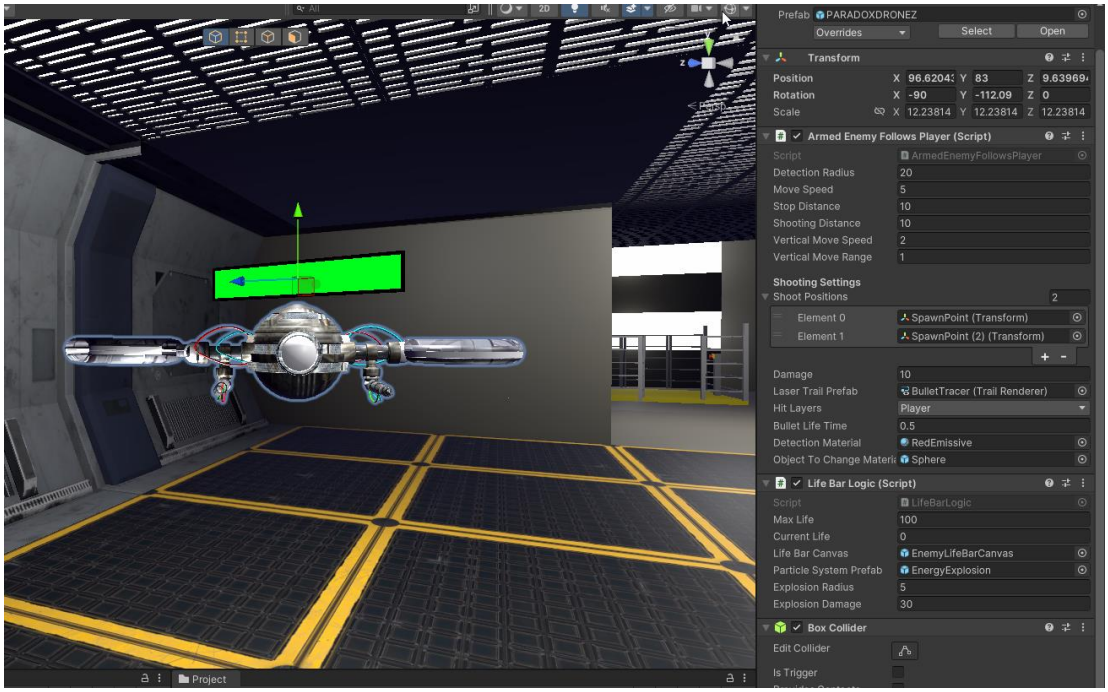


Ilustración 3.162; Componentes del enemigo Dron aéreo

Este enemigo, al tener los mismos componentes que el enemigo del dron terrestre, funciona de la misma manera.

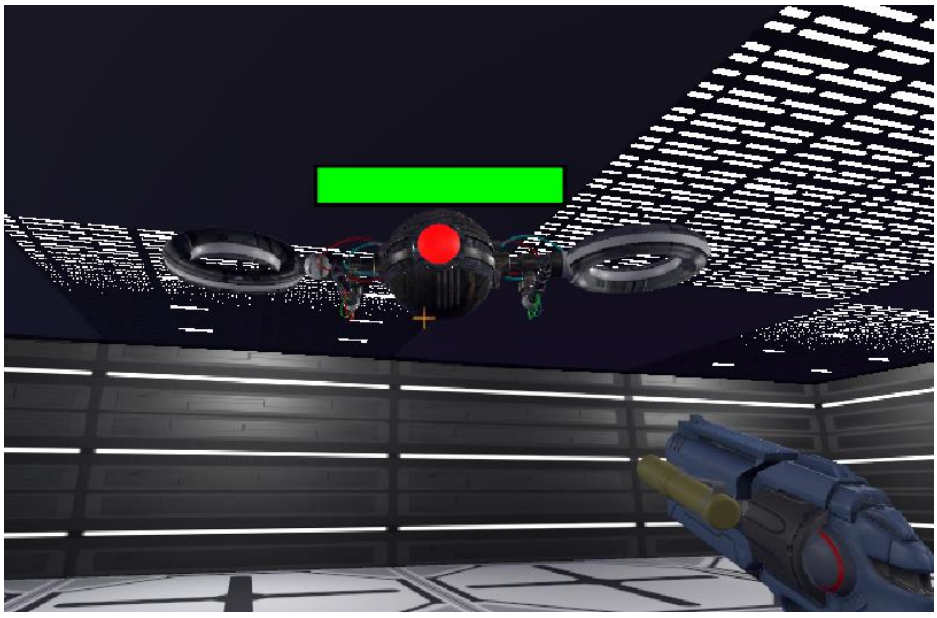


Ilustración 3.163; Dron aéreo cuando ha detectado al jugador

No obstante, cabe destacar que este enemigo, a diferencia del dron terrestre, en el script “ArmedEnemyFollowsPlayer”, se actualiza constantemente su posición en el eje Y tanto positivamente como negativamente en forma de “vaivén”, lo que le permite simular el vuelo.

3.40.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Detección del jugador
2. Barra de vida refleja daño
3. Color barra de vida
4. Enemigo hacia el jugador
5. Enemigo mira al jugador
6. Cambio de materiales
7. Materiales adecuados
8. Barra de vida siempre mirando a la camara del jugador
9. Eliminar barra de vida
10. Disparo al jugador
11. Instancia bala del enemigo
12. Bala del enemigo daña jugador
13. Giro solo en eje Y
14. Instanciar explosión
15. Daño de la explosión

Test_08_20	
Nombre de la prueba o test	Movimiento oscilatorio en Y
Objetivo	El enemigo debe realizar un movimiento oscilatorio constante en Y
Resultado	El enemigo realiza un movimiento de vaivén en un rango a lo largo del eje Y.
Comentarios	Todo OK.

Tabla 3.148; Test_08_20: movimiento oscilatorio en Y

3.2.8.5 StarLord

StarLord es el enemigo final del juego. Este tiene tanto ataques cuerpo a cuerpo como ataques a distancia.

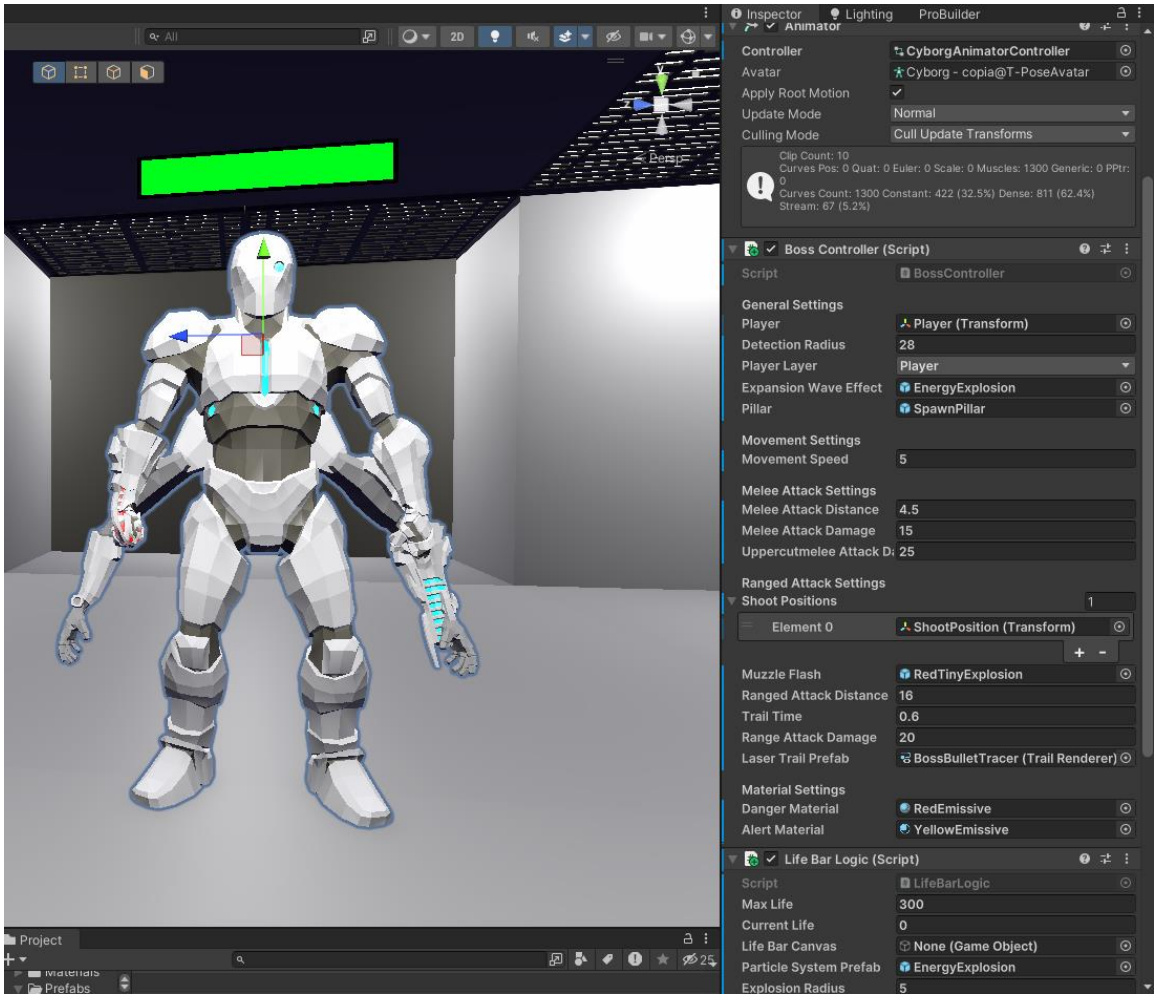


Ilustración 3.164; Componentes del enemigo StarLord

3.2.8.5.1 Animación

Este objeto tiene asociado un componente “Animator”, el cual funciona en base al componente “AnimatorController” asociado. Este “AnimatorController” tiene la siguiente estructura:

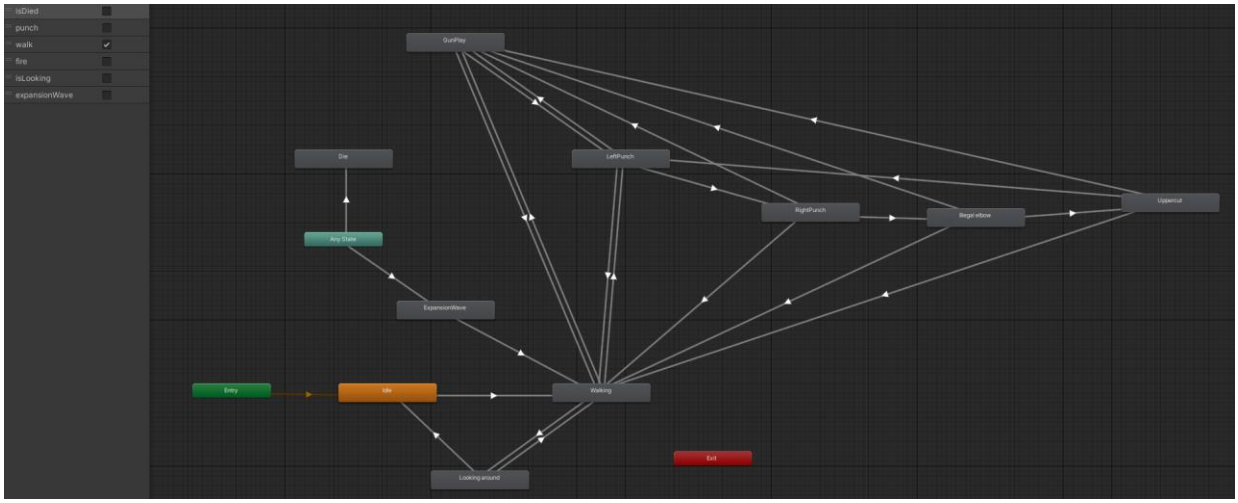


Ilustración 3.165; AnimatorController del enemigo StarLord

Este componente controla el flujo de la animación del enemigo. Este flujo funciona de la siguiente manera:

Este enemigo comenzará en la animación "Idle". Cuando el parámetro booleano "walk" se ponga a valor "true", cambiará a la animación "Walking". Si mientras está en "Walking", el parámetro "punch" se activa a "true", podrá realizar una serie de animaciones de ataque: "LeftPunch", "RightPunch", "Illegal Elbow", y "Uppercut". Estas animaciones de ataque se repetirán siguiendo ese ciclo hasta que el parámetro "punch" se ponga a "false", momento en el cual el enemigo volverá a la animación "Walking".

Además, desde "GunPlay", si el parámetro "punch" se activa a "true", el personaje podrá transitar a "LeftPunch", permitiendo una transición de disparar a golpear directamente. Por otro lado, cualquier animación de ataque ("LeftPunch", "RightPunch", "Illegal Elbow", "Uppercut") puede volver a "GunPlay" si el parámetro "fire" es "true", permitiendo que el enemigo pase de golpear a disparar.

Durante "Walking", si el parámetro "isLooking" se establece en "true" y "walk" tiene valor "false", el enemigo puede cambiar a la animación "Looking around". Si "isLooking" se vuelve "false", el enemigo regresará a "Walking".

Comentar que, si el parámetro "isDied" se establece en "true" en cualquier momento, el enemigo realizará la animación "Die", para representar que este ha muerto.

3.42.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Flujo animaciones adecuado
2. Animaciones adecuadas

3.2.8.5.2 Scripts

Todos los parámetros del componente “AnimatorController” estarán controlados por uno de los scripts asociados al enemigo, el cual es llamado “BossController”.

Este script controla todas las funcionalidades del enemigo. Primeramente, verifica que su vida no sea 2/3 de su vida total. En tal caso, el jefe “se enfurecerá”, instanciando un sistema de partículas simulando una explosión, junto con dos pilares de spawn de enemigos que aparecerán a ambos lados del jefe.

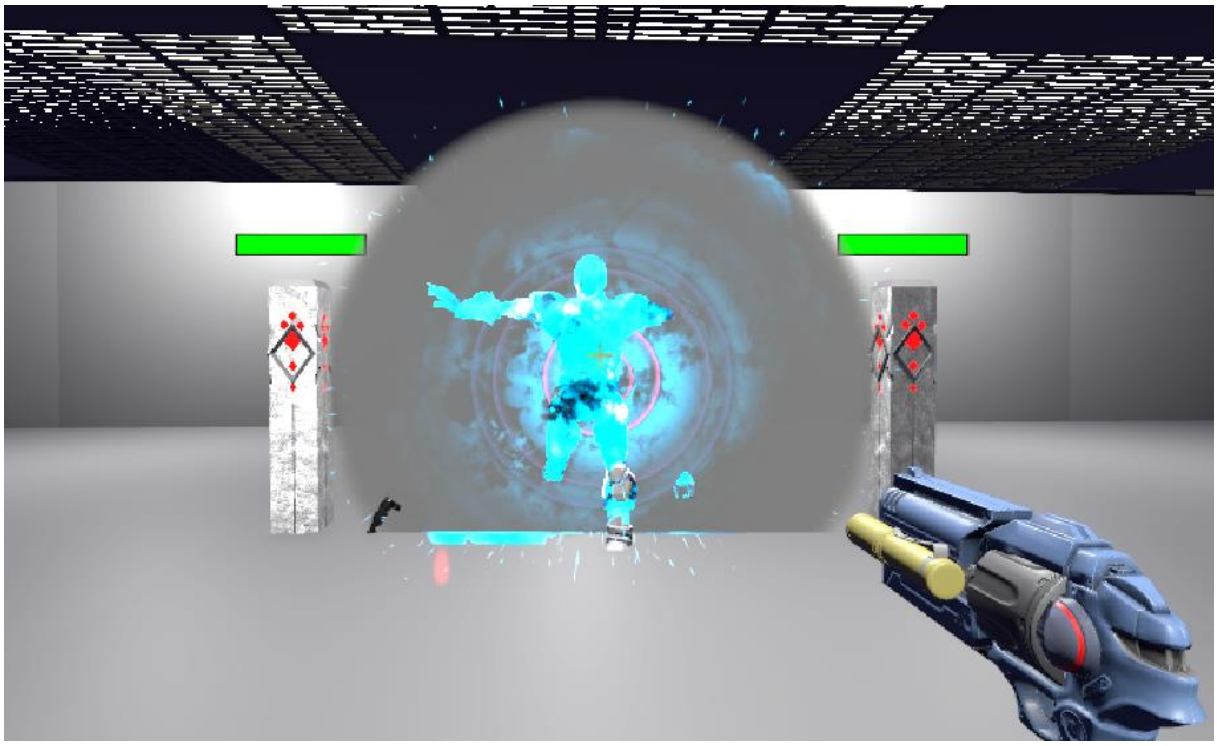


Ilustración 3.166; Explosión del enemigo StarLord

El script también controlará si la vida del enemigo está a cero o no. Si la vida del enemigo está a cero, quiere decir que está muerto. En ese momento, se iniciará la animación de muerte asociado al enemigo y este dejará de atacar.

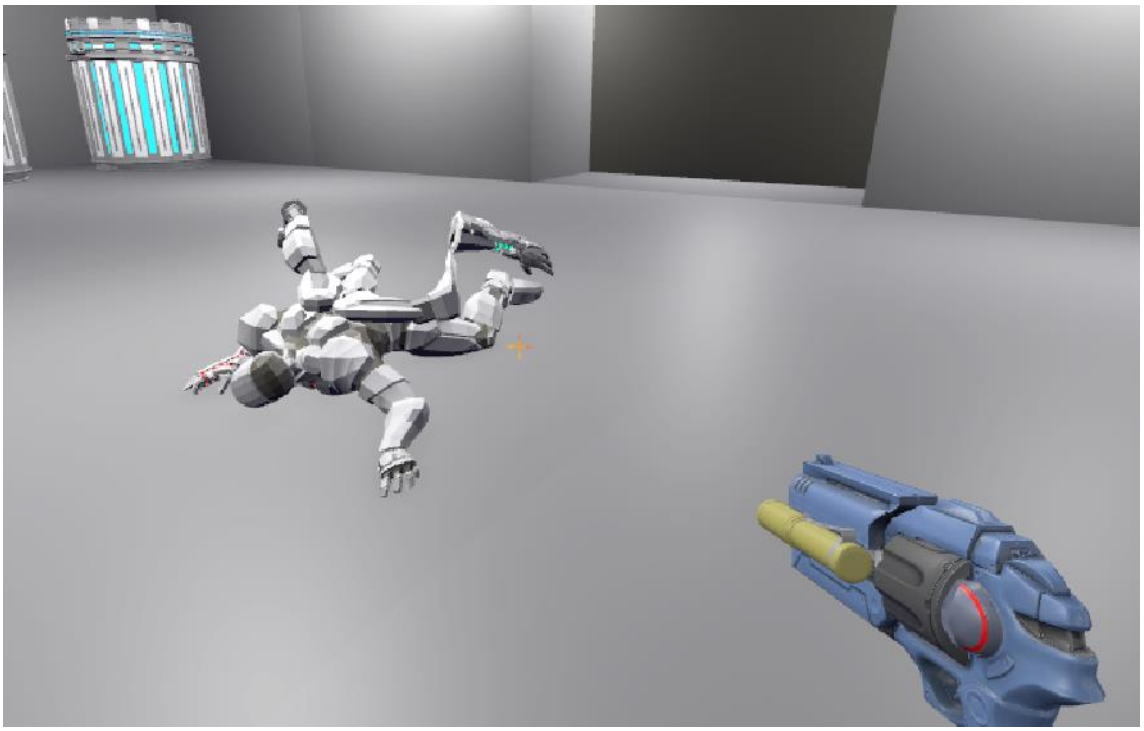


Ilustración 3.167; Muerte del enemigo StarLord

Por el contrario, si la vida del enemigo no está a cero, es decir, que aún no ha muerto, entonces entraremos dentro de la lógica de combate del enemigo. Primeramente, se creará una esfera de detección con un radio determinado con el método “CheckSphere()” de la clase “Physics”. Este método devolverá un valor booleano en función de si detecta al jugador o no. Si el jugador ha sido detectado, se hará que el enemigo mire constantemente al jugador y se cambiará el material de sus luces, como se explicará más adelante. Ahora, se comprobará a qué distancia y dentro de qué rango de detección del enemigo está el jugador, ya que el enemigo tiene tres rangos en los cuales, el enemigo toma un comportamiento diferente. Cada uno de los rangos están definidos por una esfera de detección con colores diferentes. Estos son los tres posibles casos:

1. El jugador está entre la esfera roja y la azul: el enemigo irá a por el jugador
2. El jugador está entre la esfera azul y la verde: el enemigo disparará al jugador.
3. El jugador está dentro de la esfera verde: el enemigo usará ataques cuerpo a cuerpo.

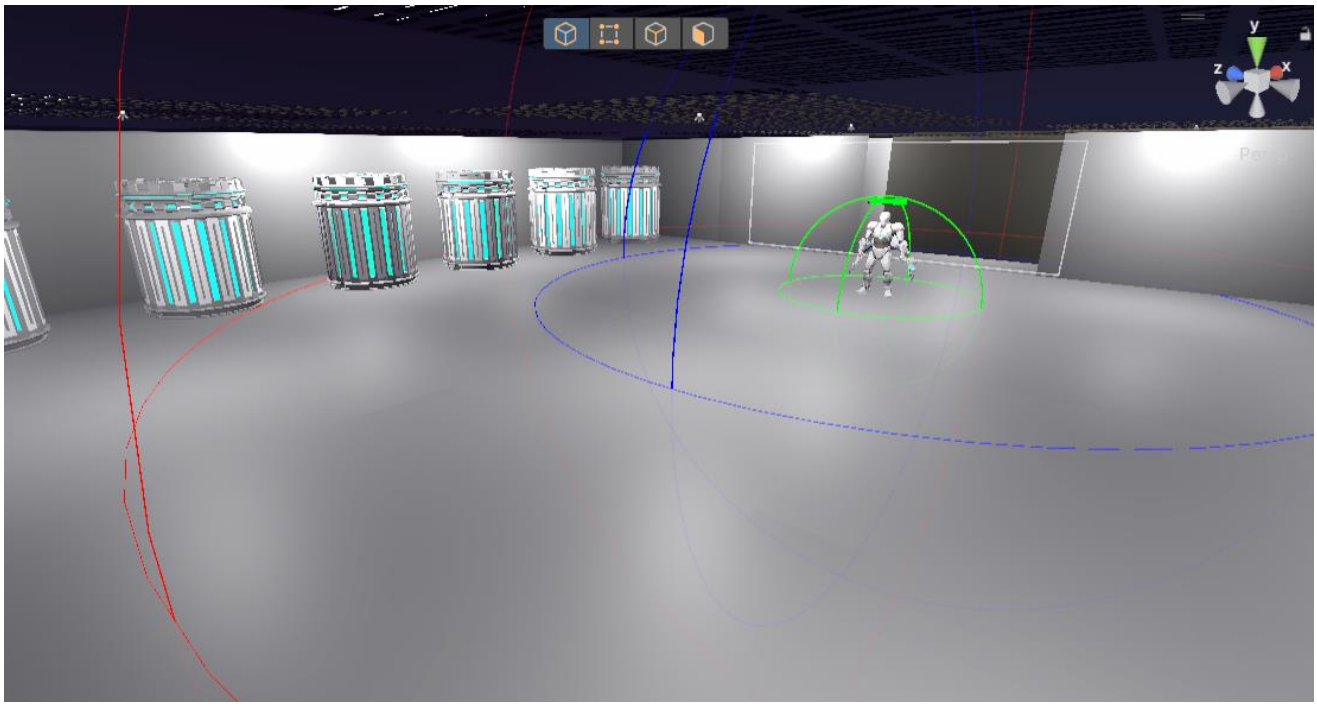


Ilustración 3.168; Radios de detección del enemigo StarLord

Para poder realizar los diferentes comportamientos, exceptuando el de moverse hacia el jugador, el cual se desarrolla en el script, lo que se ha hecho es que en el script “BossController”, en cada caso se inician las animaciones correspondientes, y en cada animación se ha añadido un evento justo en el momento en el que tiene que realizar la acción correspondiente. Cada uno de esos eventos llama a métodos públicos que proporciona el script “BossController” para que se realice la acción adecuada.

Por ejemplo, en el caso dos, en el que el enemigo dispara al jugador, lo que se hace es que en el script “BossController” se inicia la animación de disparar, la cual tiene varios eventos que llaman al método “Fire()” del script. Este método hace disparar el arma del enemigo con el mismo comportamiento que el de los enemigos de ataque a distancia, el dron terrestre y el dron aéreo, los comentados en los **apartados 3.2.8.3** y **3.2.8.4** respectivamente, por lo que no se va a detallar más este comportamiento. La única diferencia destacable es que, en el método “Fire()”, se instancia un sistema de partículas de una explosión de fuego roja, para simular y hacer que el jugador vea que el enemigo ha disparado.

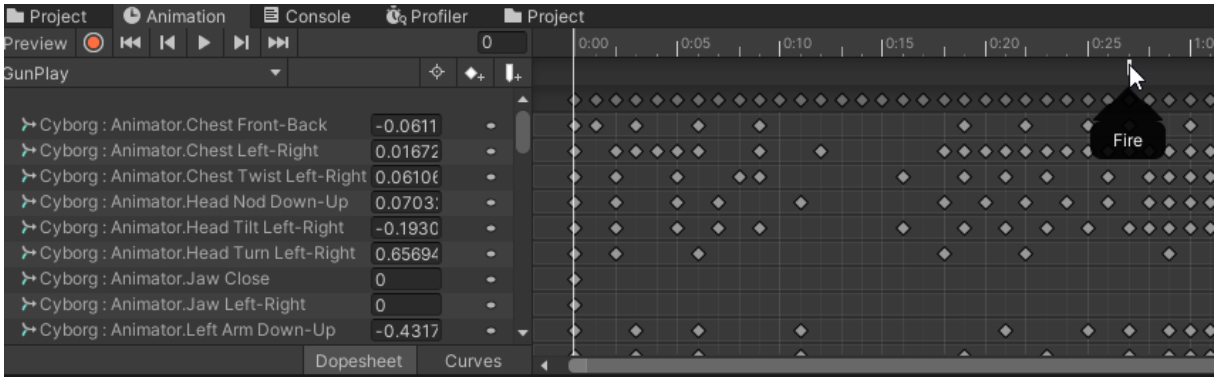


Ilustración 3.169; Eventos de la animacion “GunPlay” del enemigo StarLord

Por ejemplo, en el otro caso, el caso tres, en el que el enemigo usa ataques cuerpo a cuerpo contra el jugador, lo que se hace es que en el script “BossController” se inicia el ciclo de animaciones de ataque cuerpo a cuerpo. Este ciclo de animaciones tiene varias animaciones y todas ellas tienen eventos implementados en los que se llama al método “MeleeAttack()” del script. Este método hace atacar al jugador con golpes cuerpo a cuerpo de la misma forma en que lo hace el enemigo cuyo único

ataque es el del cuerpo a cuerpo, el dron de combate, comentado en el **apartado 3.2.8.2**, por lo que no se va a detallar más este comportamiento.

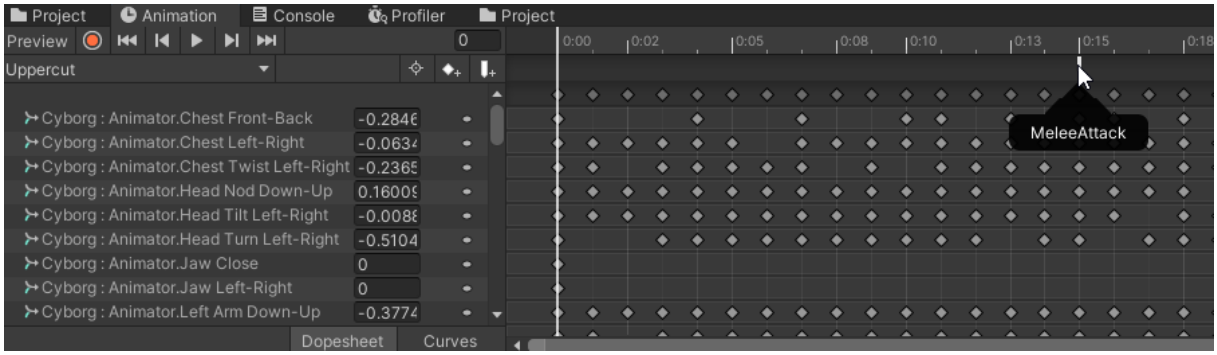


Ilustración 3.170; Eventos de la animacion "UpperCut" del enemigo StarLord

Cabe destacar que el "TrailRenderer" asociado al jefe es un poco distinto al asociado al jugador y a los enemigos. Este, al hacer más daño que los demás, es más grande, como se puede ver en la siguiente imagen:

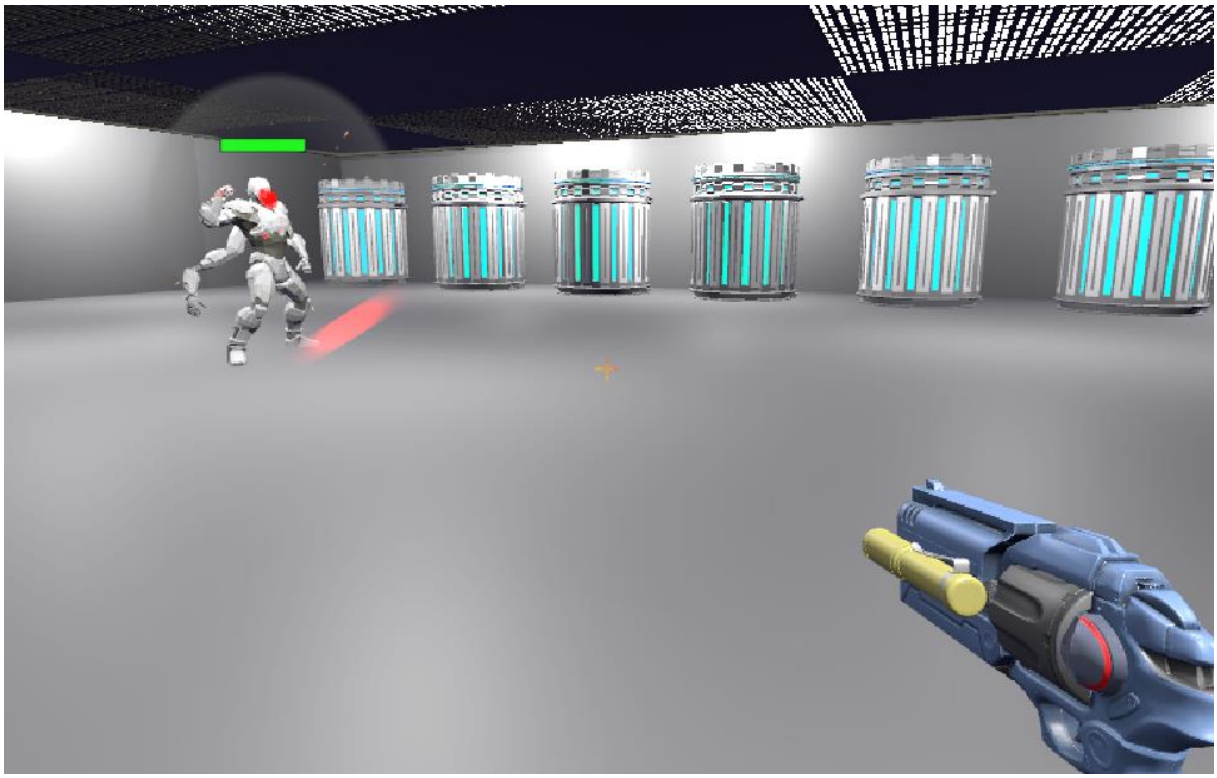


Ilustración 3.171; Disparo del enemigo StarLord

En la imagen anterior también se puede apreciar el efecto de explosión de fuego rojo que instancia el script “BossController” justo cuando el enemigo dispara, como se ha comentado antes.

El otro script asociado al enemigo es el “LifeBarLogic”, del cual hablamos en el **apartado 3.2.5.2.2 “LifeBarLogic”,** por lo que no vamos a detallar más este script.

A continuación, se va a presentar el pseudocódigo del script “BossController”:

```
Update()
  Si la vida actual del jefe es <= 2/3 de la vida máxima
    Se instancian pilares de spawn de enemigos a ambos lados del jefe
    Se activa e instancia el sistema de particulas de onda expansiva
    Se configura el sistema de partículas para que sea destruida después de 1.5 segundos

  Si la vida actual del jefe es <= 0
    Activa animación "isDied" solo una vez

  Si no
    distanceToPlayer <- se calcula distancia entre jefe y player
    playerDetected <- verifica si player está dentro de detectionRadius

    Si playerDetected
      Jefe rota hacia player solo en eje Y
      Cambia material a dangerMaterial

      Si la distancia al player es > a rangedAttackDistance y puede caminar
        Desactiva animaciones "fire", "punch", "isLooking"
        Se activa animación "walk"
        Jefe se mueve hacia el player

      Si distancia <= rangedAttackDistance y > meleeAttackDistance
        Se activa animación "fire", desactiva "punch"

      Si distancia <= meleeAttackDistance
        Se activa animación "punch", desactiva "fire"

    Si no
      Se desactiva animaciones "fire", "punch", "walk"
      Se activa animación "isLooking"

      Si la animación "Idle" está activa
        Se restaura material original

      Si la animación "Looking around" está activa
        Se desactiva "isLooking"
        Se cambia material a alertMaterial

MeleeAttack()
  Si player está a distancia de ataque cuerpo a cuerpo

    Si la animación "Uppercut" está activa
      player recibe daño de uppercutmeleeAttackDamage

    Si no
      player recibe daño de meleeAttackDamage

Fire()
  Si no hay posiciones de disparo, return
  currentShootPosition <- posición de disparo actual
  direction <- dirección normalizada desde currentShootPosition a player

  SpawnTrail(trail del jefe, posición de impacto, posición del player)

SpawnTrail(trail, endPoint, playerPositionAtFireTime)
  Mientras tiempo < bulletLifeTime
    Si trail no existe, break;
    Si trail existe, se interpola posición de trail hacia endPoint

  Si trail existe cuando tiempo >= bulletLifeTime
    posicion de trail <- endPoint
    Se destruye el trail.

    Si player está cerca de la posicion en la que estaba justo cuando el enemigo disparó
      Se aplica daño al jugador
```

Ilustración 3.172; Pseudocódigo del script "BossController"

3.44.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Barra de vida siempre mirando a la camara del jugador
2. Eliminar barra de vida
3. Enemigo hacia el jugador
4. Enemigo mira al jugador
5. Daño por golpe
6. Disparo al jugador
7. Instancia bala del enemigo
8. Bala del enemigo daña jugador
9. Giro solo en eje Y
10. Instanciar explosión
11. Detección del jugador
12. Barra de vida refleja daño
13. Color barra de vida
14. Movimiento sin variar Y

Test_08_21	
Nombre de la prueba o test	Instanciación de enemigos
Objetivo	El enemigo debe instanciar a otros enemigos al llegar a cierto momento del combate
Resultado	El enemigo, cuando su vida baja una determinada cantidad, instancia a otros enemigos
Comentarios	El jugador puede no esperarse que el enemigo va a instanciar a otros enemigos

Tabla 3.149; Test_08_21: instanciación de enemigos

Test_08_22	
Nombre de la prueba o test	Ubicación de instanciación de enemigos
Objetivo	El enemigo debe instanciar a otros enemigos en una ubicación y posición adecuada
Resultado	El enemigo, cuando su vida baja una determinada cantidad, instancia a otros enemigos a sus lados, sin interferir al enemigo y mirando hacia el jugador.
Comentarios	Todo OK.

Tabla 3.150; Test_08_22: ubicación de instanciación de enemigos

Test_08_23	
Nombre de la prueba o test	Ataque según distancia con jugador
Objetivo	El enemigo debe atacar de una manera u otra en función de la distancia a la que se encuentre el jugador
Resultado	Si el jugador está a una determinada distancia, el enemigo realizará un ataque, pero si está a otra distancia, realizará otro ataque diferente
Comentarios	El jugador puede no saber a qué distancia el enemigo hace un ataque u otro.

Tabla 3.151; Test_08_23: ataque según distancia con jugador

3.2.8.5.3 Lógica de materiales

Como se ha comentado previamente, en el script “BossController” se controlará toda la lógica de materiales de las luces del enemigo. El color de estas luces significará el estado del enemigo:

1. Si la luz es roja: el enemigo ha detectado al jugador
2. Si la luz es amarilla: el enemigo había detectado al jugador, pero le ha perdido de vista y está buscándolo
3. Si la luz es azul: el enemigo está tranquilo.

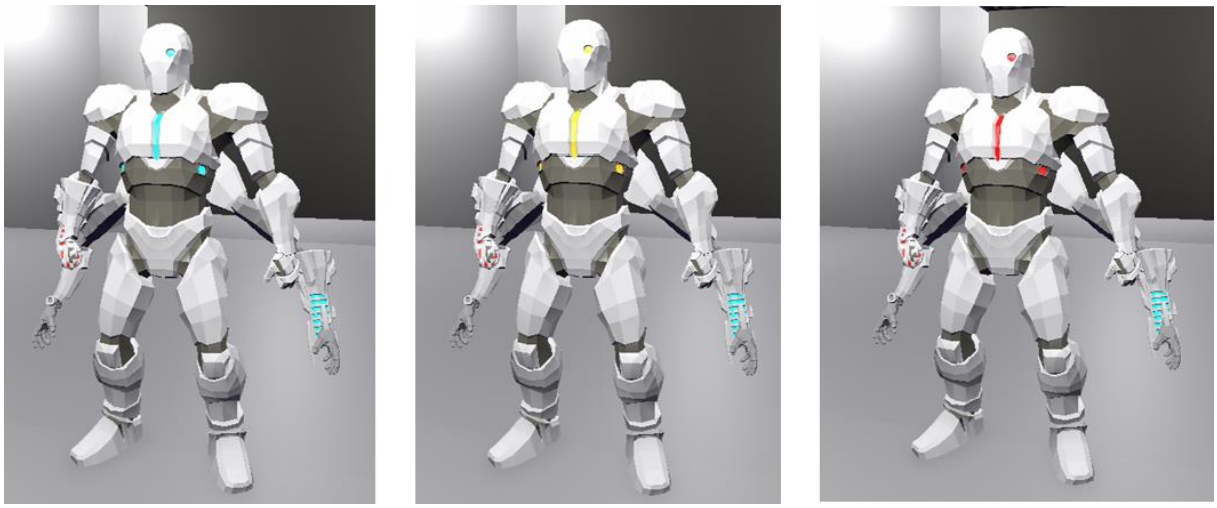


Ilustración 3.173; Cambios de materiales de StarLord en función de su estado de detección

3.46.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Cambio de materiales
2. Materiales adecuados

3.2.8.6 Pilar de invocación de enemigos

Este no es un enemigo como tal, sino que es un objeto que instancia enemigos a su alrededor.

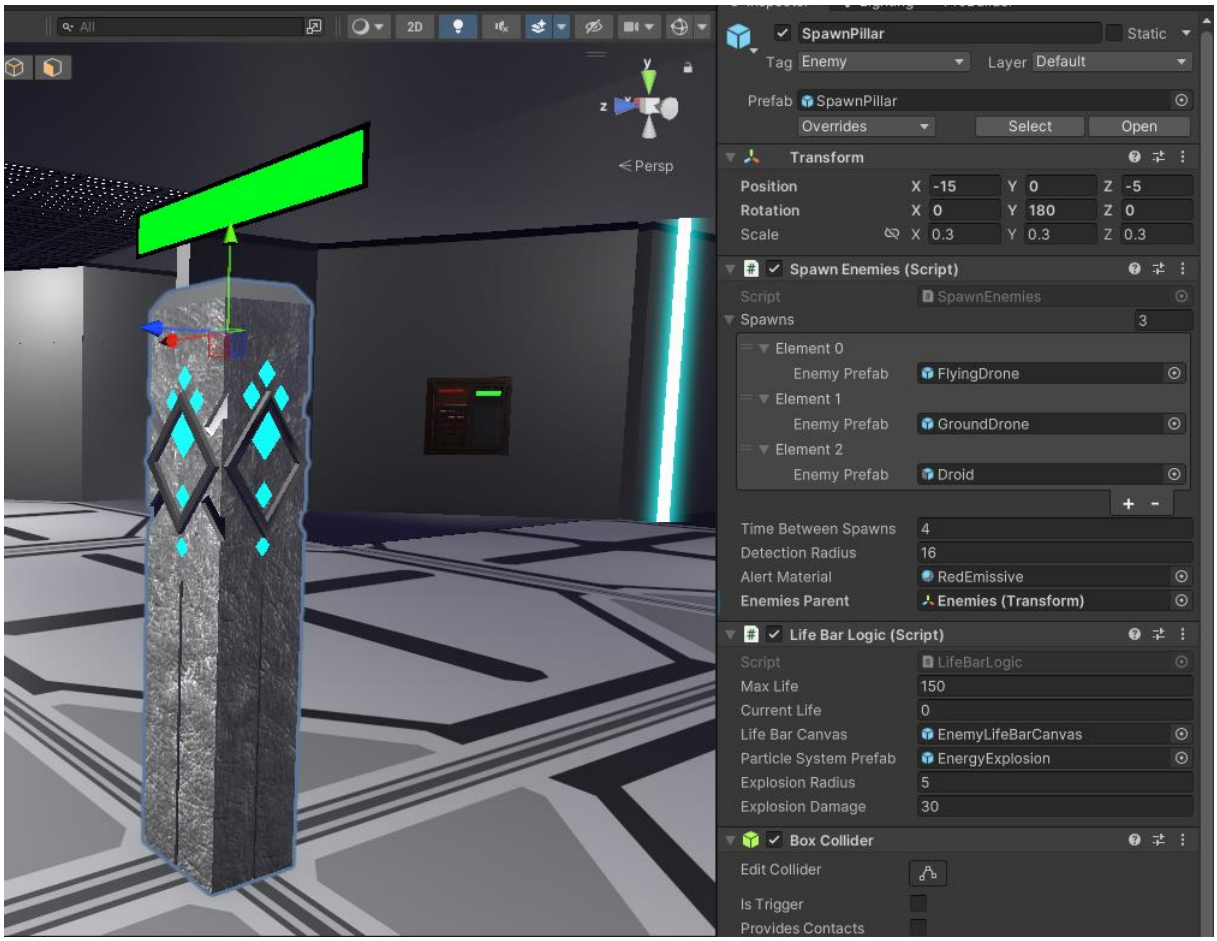


Ilustración 3.174; Componentes del enemigo Pilar de invocación de enemigos

Para instanciar a los enemigos, el pilar tiene asignado un script llamado “SpawnEnemies”, el cual, primeramente, crea una estructura llamada “Spawn”, la cual almacena el prefab [98] del enemigo y la cantidad de ese prefab que se va a instanciar. Se pasan como parámetros los prefabs de los enemigos, mientras que, la cantidad de instancias de ese prefab que van a aparecer se calcula cogiendo un número aleatorio entre un rango de valores determinados por la dificultad en la que se encuentre el juego.

Este script, crea una esfera de detección usando el método “CheckSphere()” de la clase “Physics” con un radio de detección determinado para detectar si el jugador entra dentro de esa área de detección.

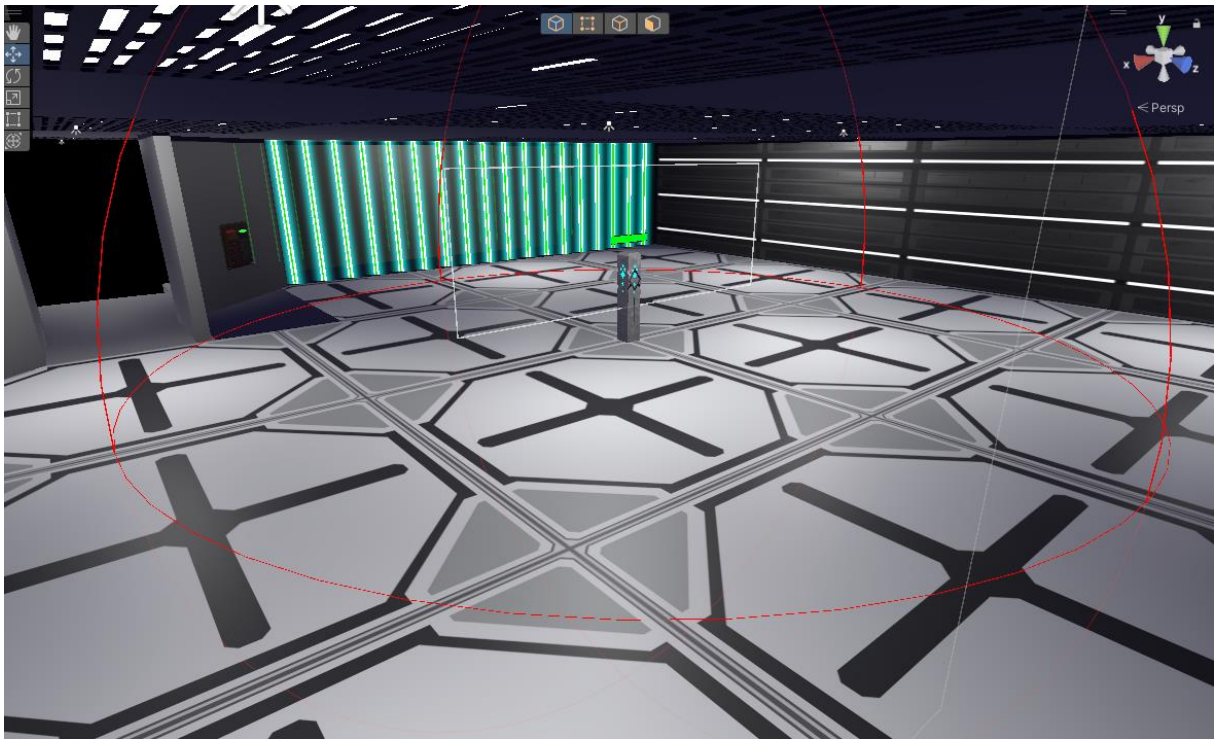


Ilustración 3.175; Área de detección del enemigo Pilar de invocación de enemigos

Si el jugador es detectado por el pilar haciendo uso de la esfera de detección, el script cambiará el material de las luces del pilar a rojas, para indicar al jugador que ha sido detectado por el pilar y que este va a empezar a instanciar enemigos. Después de cambiar el material de las luces, se inicia la corrutina “SpawnEnemy()”.

Esta corrutina, primero calcula la cantidad de objetos “Spawn” que hay cuya cantidad de enemigos a instanciar es distinto de cero. En base a esa cantidad de objetos, se coge uno al azar, de forma que el enemigo que se va a instanciar será aleatorio. Para instanciar el enemigo, antes se ha de calcular la posición de aparición del enemigo. Esta es calculada de forma que el enemigo siempre aparezca delante del jugador y nunca detrás o a los lados, con lo que conseguimos que los enemigos instanciados sean ubicados por el jugador y no le pillen por sorpresa. Para ello, lo que se hace es instanciar enemigos en la media-luna opuesta del jugador. Esto se consigue calculando el vector de dirección desde el jugador hacia el pilar. Luego, en base a ese vector, se calcula el vector que "divide" el área en la que se van a instanciar los enemigos. Esta área comprenderá el rango de [-90º, 90º], siendo el vector calculado el 0º. A continuación, se adjunta una imagen en la que se aclara esta última explicación:

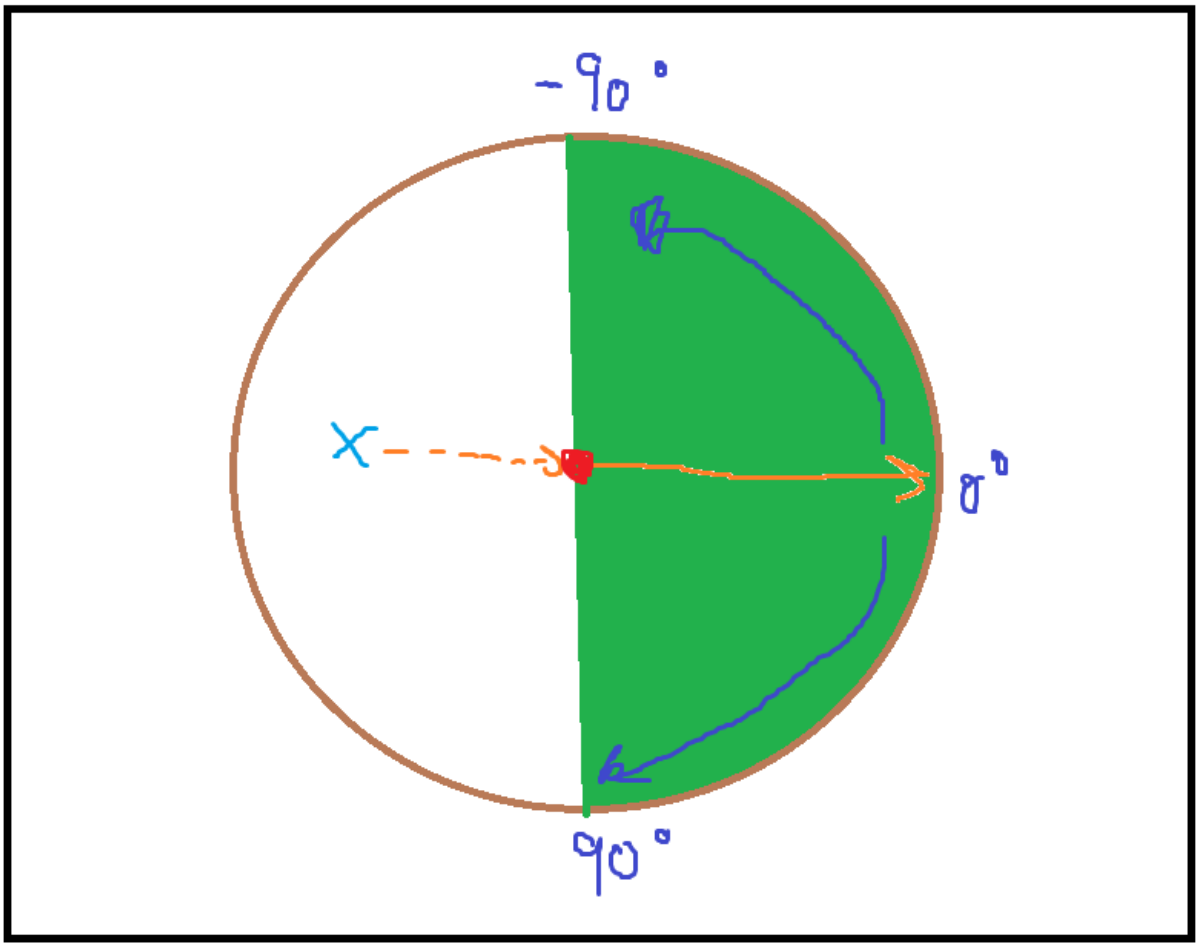


Ilustración 3.176; Explicación de la invocación de enemigos del enemigo Pilar de invocación de enemigos

En ella, se puede ver como el jugador (aspa azul) tiene un vector de dirección (líneas discontinuas naranjas) hacia el pilar (punto rojo) y ese vector, si se alarga (linea naranja) será el 0º del área de aparición del enemigo a instanciar (área verde). A continuación, se adjunta una imagen en la que se podrán ver diversas situaciones en base a este principio:

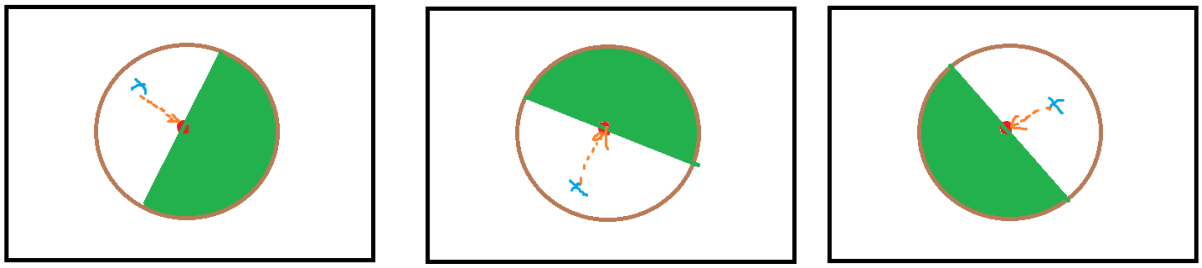


Ilustración 3.177; Casos de invocación de enemigos del enemigo Pilar de invocación de enemigos

Una vez calculada el área de aparición del enemigo, se obtiene un punto aleatorio dentro de esa área. Esa posición obtenida, por defecto para la mayoría de enemigos, tendrá la componente vertical Y en cero puesto que son enemigos terrestres, pero si el enemigo es volador, se calculará un valor aleatorio para la altura con la que se instanciará el enemigo volador.

Finalmente, habiendo calculado la posición exacta en la que instanciar el enemigo, se obtiene una rotación en el eje Y aleatoria en la que instanciar el enemigo para que este no siempre aparezca mirando hacia el mismo punto siempre, y se instancia el enemigo.

Después de instanciar el enemigo, se reduce el número de enemigos del objeto “Spawn” del cual se ha instanciado el enemigo.

Cabe destacar que, antes de instanciar el siguiente enemigo, el script se espera un cierto tiempo determinado.

Ademas de las funcionalidades comentadas, el script constantemente actualiza las posiciones en Y de los enemigos terrestres de forma que las corrige si hay algún desbalance. Esto se hace puesto que, al haber muchos enemigos instanciados en algún momento concreto, había veces en las que los enemigos terrestres se montaban encima de los otros enemigos y se quedaban flotando. De esta forma, si alguno de los enemigos terrestres tiene una componente Y de su posición distinta a la de sus compañeros, lo que se hace es que se corrige y se actualiza por el valor de la componente Y de la posicion de los otros enemigos del mismo tipo.

Por último, cabe comentar que este objeto tiene también el script “LifeBarLogic” asociado, por lo que consta de una barra de vida, la cual, si llega a cero, este objeto desaparecerá y ya no podrá seguir instanciando enemigos.

Se aporta el pseudocódigo del script:

```
Update()
  CheckPlayerDetection()
  NormalizeDroidHeight()
  NormalizeGroundDroneHeight()

CheckPlayerDetection()
  detected <- verifica si hay un jugador dentro del detectionRadius

  Si detected y no isPlayerDetected
    isPlayerDetected <- verdadero
    Cambia el material luces a rojas.

    SpawnEnemy()

  Si no detected y isPlayerDetected
    isPlayerDetected <- falso
    Restablece el material de luces a originalMaterial

SpawnEnemy()
  Mientras isPlayerDetected y playerTransform no es nulo
    availableSpawns <- cantidad de spawns donde la cantidad enemigos a instanciar > 0
    Selecciona un spawn aleatorio de availableSpawns
    Se calcula posición de instanciación de enemigos
    Instancia el prefab de enemigo en posicion con rotation aleatoria
    Decrementa la cantidad del spawn seleccionado

NormalizeDroidHeight()
  Para cada droid instanciado
    Ajusta la posición en y

NormalizeGroundDroneHeight()
  Para cada drone instanciado
    Ajusta la posición en y
```

Ilustración 3.178; Pseudocódigo del script "SpawnEnemies"

3.48.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Barra de vida siempre mirando a la camara del jugador
2. Eliminar barra de vida
3. Cambio de materiales
4. Materiales adecuados
5. Instanciar explosión

6. Detección del jugador
7. Barra de vida refleja daño
8. Color barra de vida
9. Instanciación de enemigos

Test_08_24	
Nombre de la prueba o test	Ubicación de los enemigos instanciados en base a posicion del jugador
Objetivo	Los enemigos se deben de instanciar en función de la posición del jugador en el momento de la instanciación
Resultado	A la hora de instanciar enemigos, estos solo aparecen en frente del jugador
Comentarios	El jugador puede que no sepa que solo se pueden instanciar delante suya y puede llegar a pensar que pueden instanciarse en otras ubicaciones, como a sus lados o detrás.

Tabla 3.152; Test_08_24: ubicación de los enemigos instanciados en base a posición del jugador.

3.2.9 Interacción

A la hora de hablar de interacción, el jugador puede interactuar con elementos del entorno, con NPCs aliados y con los ítems que se encontrará a lo largo del juego.

Todas las interacciones se basan en una clase interfaz llamada “IAction” la cual tiene un método llamado “Activate()” que los scripts de la lógica de interacción deberán sobrescribir para realizar la acción de la interacción.

3.2.9.1 Interacción con NPCs aliados

Para interactuar con los diferentes NPCs aliados que el jugador se encontrará a lo largo del juego, se hace uso del script llamado “NPCInteraction”. Este script lo que hace es crear un “raycast” desde la cámara hasta una distancia de interacción establecida. Si el “raycast” choca con algún elemento y ese elemento tiene la etiqueta “NPC”, se actualiza la variable “highlight” con ese elemento, la cual almacena la información del elemento con el que se va a interactuar. Además, se activa el panel

de interacción con el que el jugador sabe que puede interactuar con el NPC y además sabe qué tiene que hacer para interactuar con él.

Ilustración 3.179; Interacción con un NPC

Después de haber activado el panel de interacción y de haber guardado la información del elemento con el que se va a interactuar, si se pulsa el botón de interacción, la tecla “e”, se recogen todos los componentes del elemento que sobrescriban el método “Activate()” de la clase interfaz “IAction” para ejecutarlo y realizar la acción de la interacción. Además, se desactiva el panel de interacción para que el jugador sepa que no se puede interactuar con el NPC.

3.50.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_09_01	
Nombre de la prueba o test	Aparición del panel de interacción
Objetivo	El panel de interacción debe aparecer cuando se pueda interactuar

Resultado	El panel de interacción aparece cuando el jugador puede interactuar con el elemento
Comentarios	El jugador puede que no sepa qué hacer para interactuar con el elemento. Además, puede no saber qué efecto tiene interactuar con él.

Tabla 3.153; Test_09_01: aparición del panel de interacción

Test_09_02	
Nombre de la prueba o test	Acción de interactuar
Objetivo	Tras el jugador interactuar con el elemento, se debe realizar la acción de interacción asociada al elemento.
Resultado	Al interactuar el jugador con el elemento, se realiza la acción asociada a su interacción.
Comentarios	El jugador puede que no sepa qué pasará al interactuar con el elemento.

Tabla 3.154; Test_09_02: acción de interactuar

3.2.9.2 Interacción con elementos del entorno e ítems

Para que el jugador sepa que puede interactuar con ese elemento del juego, además de activar el "Canvas" de interacción, se usa un asset sacado de la "AssetStore" de Unity, el cual se llama "Quick Outline" [99] y marca el contorno del elemento de un color específico haciendo uso de un script llamado "Outline" que debe estar asociado al elemento y es aportado por el asset.

La interacción con elementos del entorno e ítems se realiza gracias al script "SelectObject", el cual se encarga tanto de marcar el contorno del elemento con el asset comentado antes, como de ejecutar la acción de interacción con el elemento.

El funcionamiento de este script es similar al del script "NCPInteraction" pero con algunas diferencias. Primeramente, como el script "NCPInteraction", lo que hace el script es crear un "raycast" desde la cámara del jugador hasta una distancia de interacción determinada para activar el "Canvas" de interacción y almacenar

información del elemento con el que colisiona el rayo, pero, a diferencia del script "NCPInteraction", solo si este elemento tiene la etiqueta "Interactive". Además de almacenar la información del elemento con el que colisiona, se coge el script "Outline" asociado al elemento (si no está asociado, se le asocia desde este script) y se activa el color del contorno del elemento.

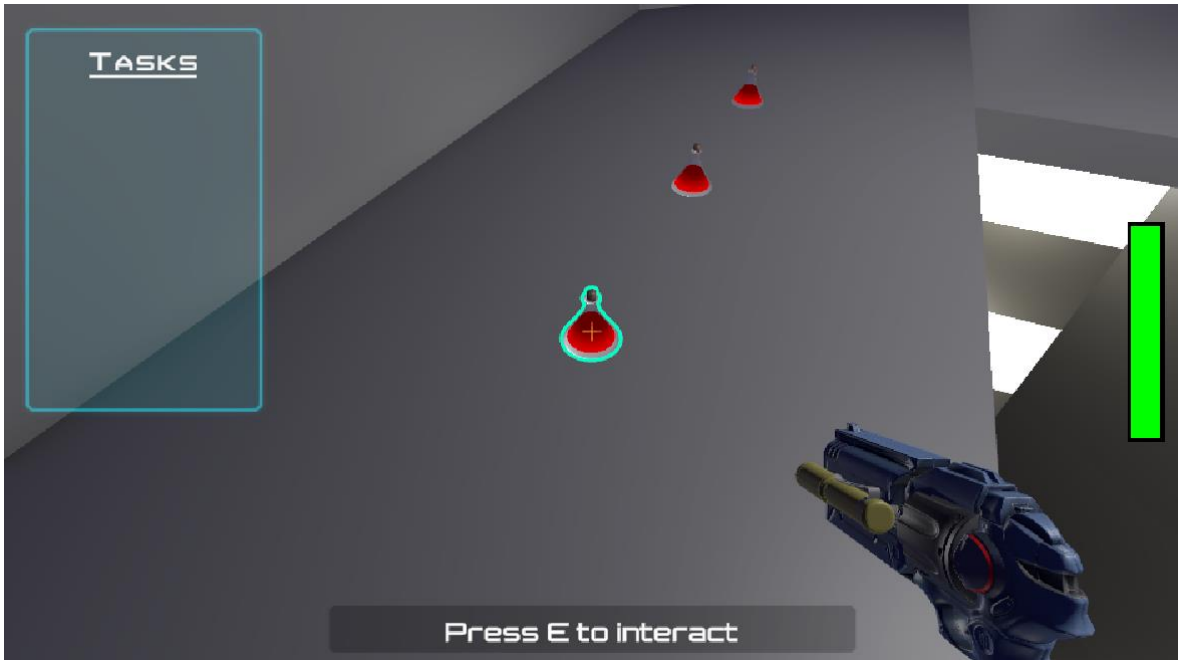


Ilustración 3.180; Interacción con poción de curación

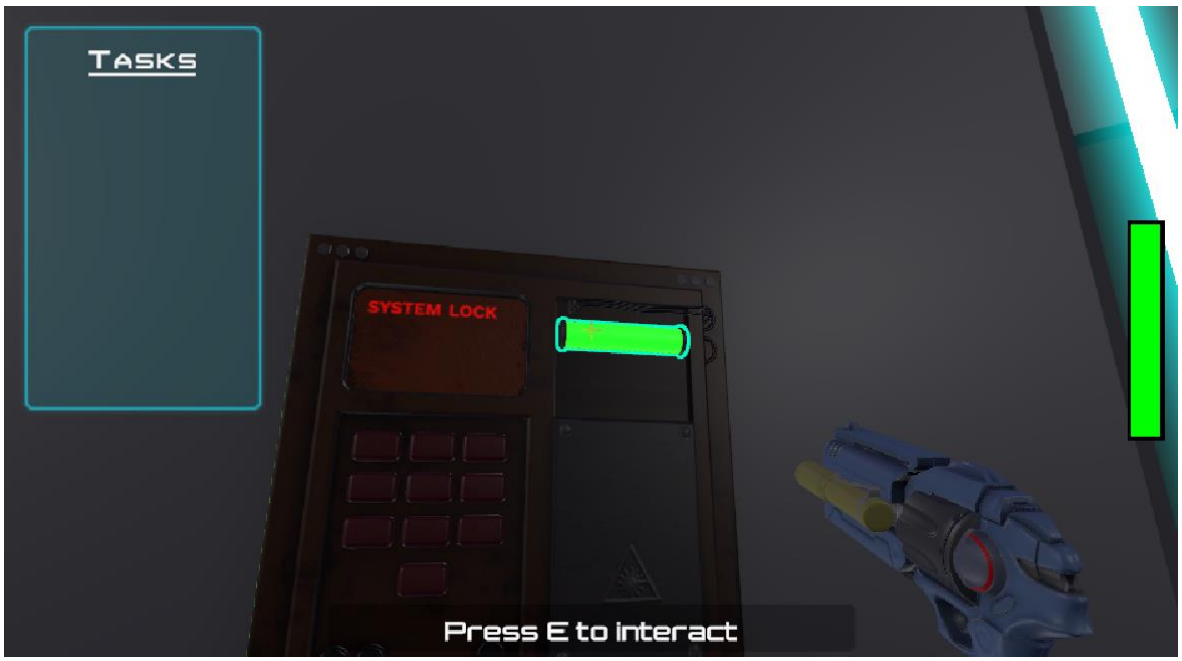


Ilustración 3.181; Interacción con batería de las salas

Todas estas funcionalidades se realizarán como se han comentado para todas las interacciones con elementos del entorno y con ítems, pero como excepción para el panel de control de la puerta de StarLord, todas estas funcionalidades se realizarán si el jugador tiene activo el ítem de batería en sus manos y el panel no tiene las tres baterías colocadas.

Ilustración 3.182; Interacción con panel de control de la sala de StarLord

Ilustración 3.183; No se puede interactuar con panel de control

Como se hacía con los NPCs, si el jugador pulsa la tecla de interacción que le muestra en el panel de interacción, la tecla “e”, se recogerán todos los componentes del elemento que sobrescriban el método “Activate()” de la clase interfaz “IAction” para ejecutarlo y realizar la acción de la interacción. Además, se desactiva el panel de interacción para que el jugador sepa que no se puede interactuar con el NPC.

3.52.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Aparición del panel de interacción
2. Acción de interacción

Test_09_03	
Nombre de la prueba o test	Aparición del efecto “outline”
Objetivo	El contorno del elemento con el que se puede interactuar se debe marcar
Resultado	Cuando el jugador mira hacia el elemento interactuable, se marca su contorno
Comentarios	Todo OK.

Tabla 3.155; Test_09_03: aparición del efecto “outline”

3.2.10 NPCs aliados

Para que los NPCs aliados ayuden al jugador en su partida, es necesario implementar la lógica de cada uno de ellos.

3.2.10.1 Kyle

Kyle es el primer NPC que el jugador se encuentra y es el que aporta el tutorial para que el jugador aprenda a jugar. Este, por tanto, es el que tiene la lógica más compleja de los NPCs.

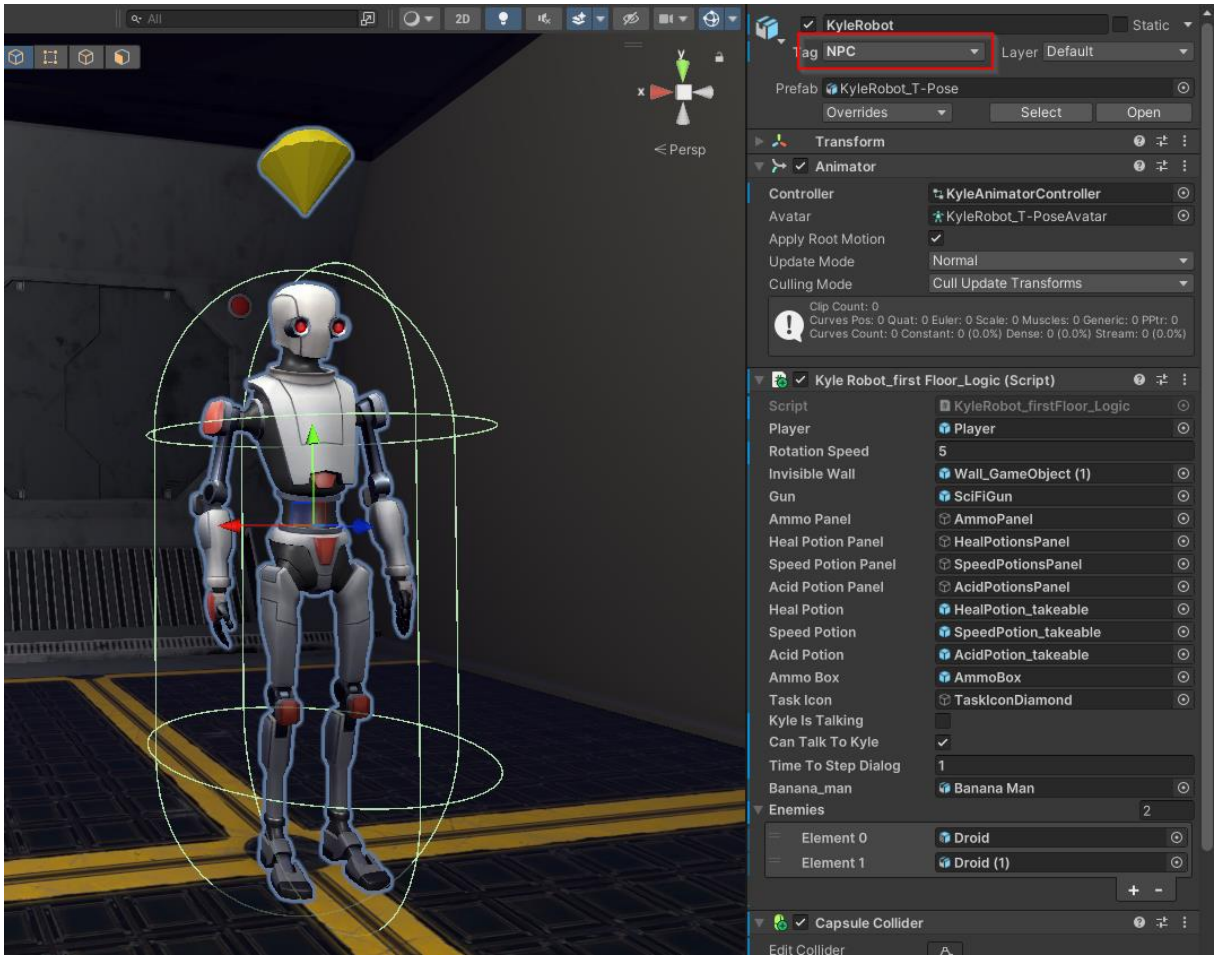


Ilustración 3.184; Componentes de Kyle

Primero de todo, se ha de comentar que este objeto tiene un componente “Animator” asociado, el cual tiene un componente “KyleAnimatorController” asignado que controla el flujo de las animaciones del objeto. El componente “KyleAnimatorController” tiene la siguiente estructura:

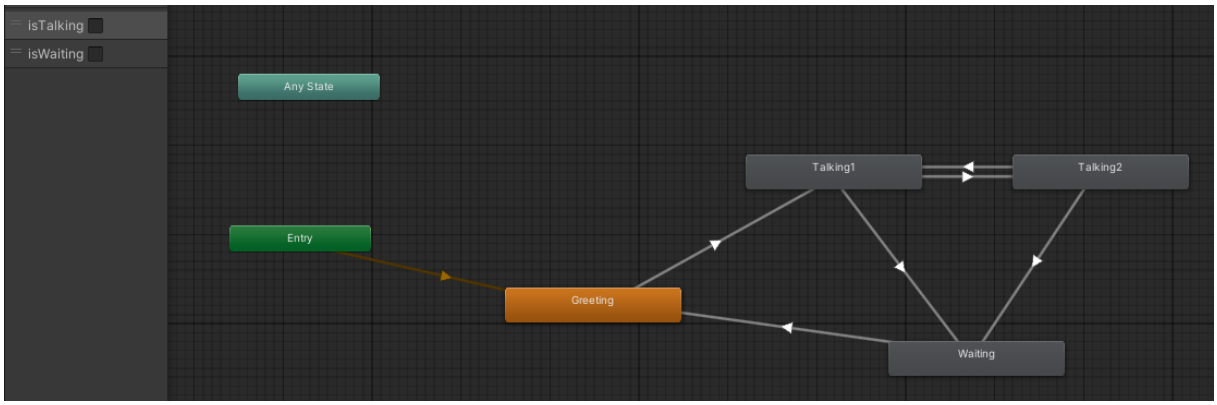


Ilustración 3.185; AnimatorController de Kyle

Esta comprende varias animaciones. La primera de ellas es “Greeting”, en la que Kyle saludará para indicarle al jugador que puede hablar con él. Si el parámetro “isTalking” se pone a true, pasará al ciclo de animaciones de hablar en el que, de forma sucesiva, pasará de una animación de hablar a otra hasta que el parámetro “isWaiting” se ponga a true y el parámetro “isTalking” se ponga a falso, momento en el cual Kyle realizará la animación “Waiting”, en la que se pondrá a esperar a que el jugador complete la tarea que se le ha encomendado.

Todos estos parámetros son controlados por el script asociado al objeto, el cual se llama “KyleRobot_firstFloor_Logic” y controla la lógica del NPC. Este script hereda y sobrescribe el método “Activate()” de la clase interfaz “IAction”, por lo que, cuando el jugador interactúe con el NPC, gracias a que este tiene asignada la etiqueta “NPC”, se llamará al método “Activate()”. Este método lo que hace es ver si se puede hablar con Kyle o no. Esto es controlado por la variable booleana “canTalkToKyle”, de forma que si tiene valor “true” el jugador puede hablar con Kyle, y si tiene valor “false” el jugador no puede hablar con Kyle. Si “canTalkToKyle” tiene valor “true”, lo que se hace es que, si no está hablando Kyle, controlado de la misma manera por la variable booleana “kyleIsTalking”, se empieza el dialogo, y si está hablando Kyle, se sigue el dialogo por donde se quedó.

Para empezar el dialogo, lo que se hace es que se le obliga al jugador a mirar a Kyle y se le prohíbe moverse, disparar y mover la cámara. Además, se activa el panel de dialogo en el que se mostrará el diálogo que el jugador está manteniendo con Kyle. Con estos prerrequisitos, se continúa el dialogo por el siguiente mensaje.

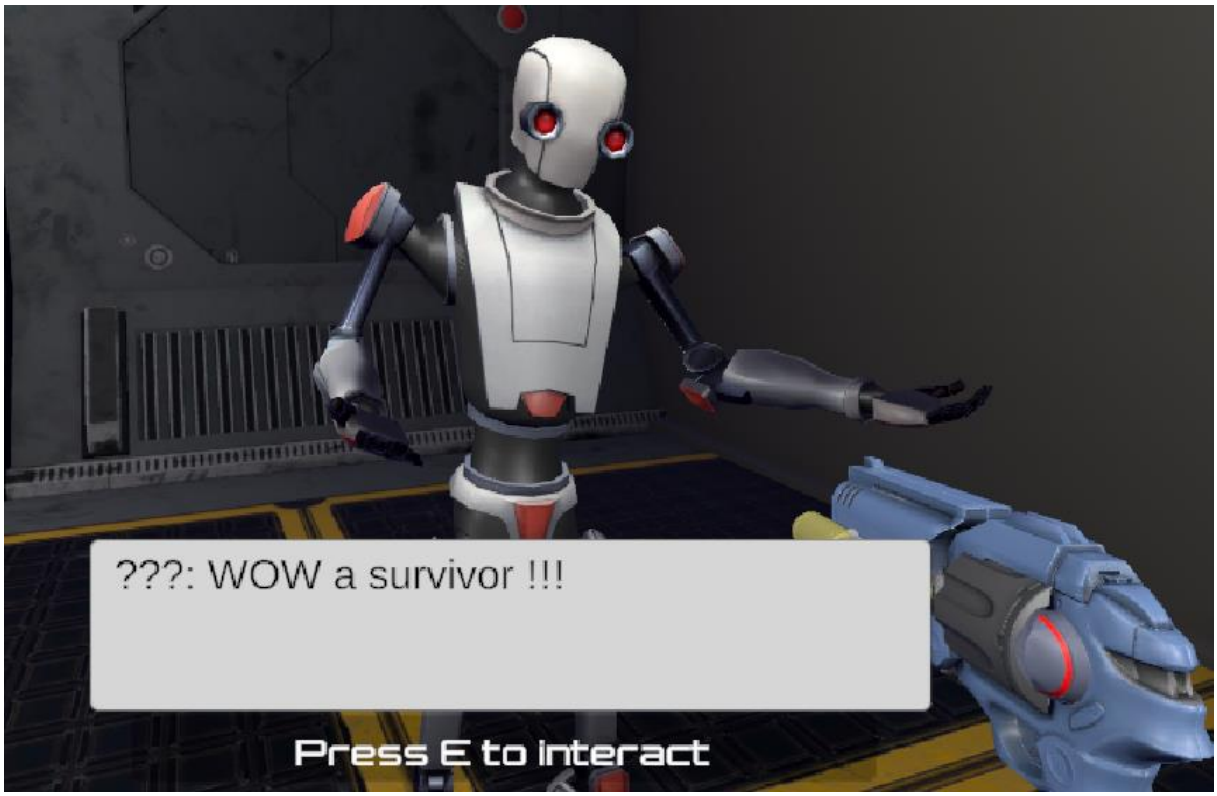


Ilustración 3.186; Dialogo con Kyle

Cuando el dialogo con Kyle llegue a ciertos puntos, como en el momento en el que Kyle le manda la tarea de correr o saltar al jugador, el dialogo se parará, desactivando el panel de diálogo y volviéndole a activar la movilidad del personaje y de la cámara del jugador, aunque no el poder disparar, a no ser que sea necesario el poder disparar para la tarea que le ha encomendado Kyle al jugador.

En estos puntos, lo que se hace es actualizar el texto del panel de tareas de la interfaz del jugador, para que este sepa qué tiene que hacer.

Cuando el jugador complete la tarea encomendada, se activará la animación “Greeting” de Kyle, así como un icono en forma de diamante que indicará al jugador que puede hablar con Kyle.

Ilustración 3.187; Kyle quiere hablar con el jugador

Si el jugador vuelve a hablar con Kyle, se desactivará el icono de encima de Kyle y seguirá el diálogo por donde se quedó.

Tras algunos diálogos con Kyle, el script instanciará un enemigo llamado “Banana-man”, del cual se habló en apartados anteriores. Este enemigo se instanciará para que el jugador aprenda la mecánica de disparar y así pueda completar la tarea de disparar. En el mismo momento en el que se le encarga la tarea de disparar al jugador, el script activará el panel de la munición del arma, para que así el jugador pueda ver que este panel está relacionado con el arma y con la mecánica de disparar, así como con la funcionalidad de munición del arma.

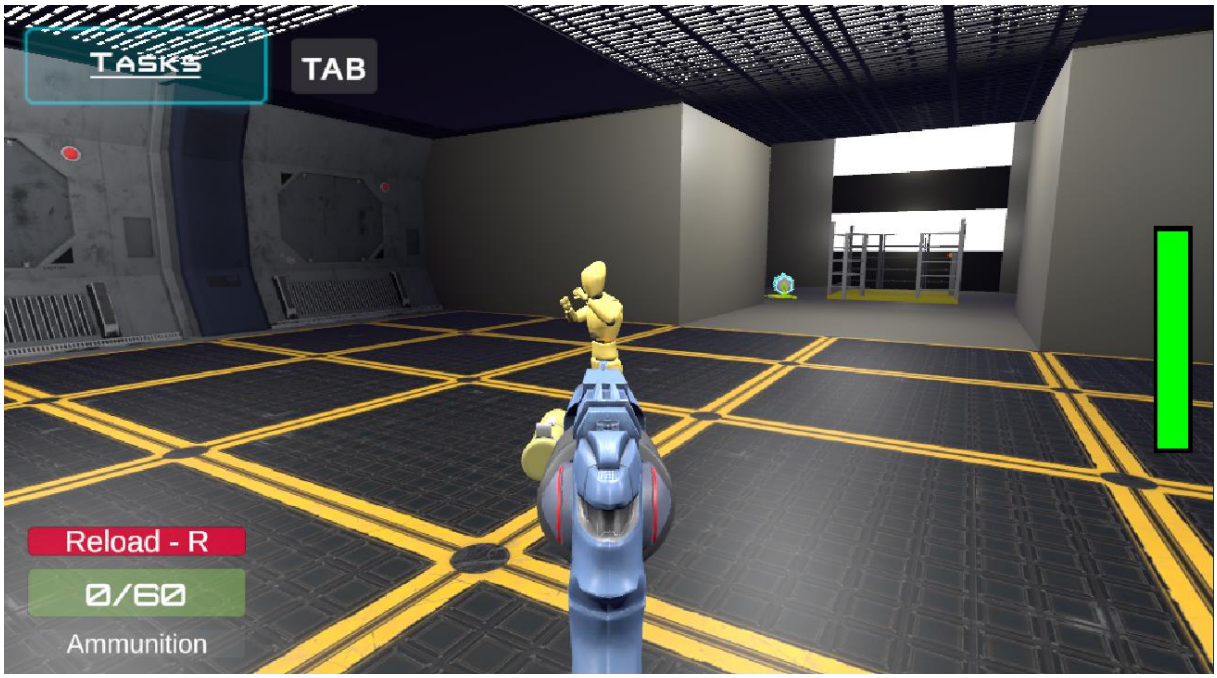


Ilustración 3.188; Tarea de disparar a banana-man

Tras completar esa tarea, Kyle felicitará al jugador por sus esfuerzos y aparecerán enemigos, los cuales Kyle le pide al jugador que elimine. Estos enemigos serán instanciados por el script en un punto predefinido.

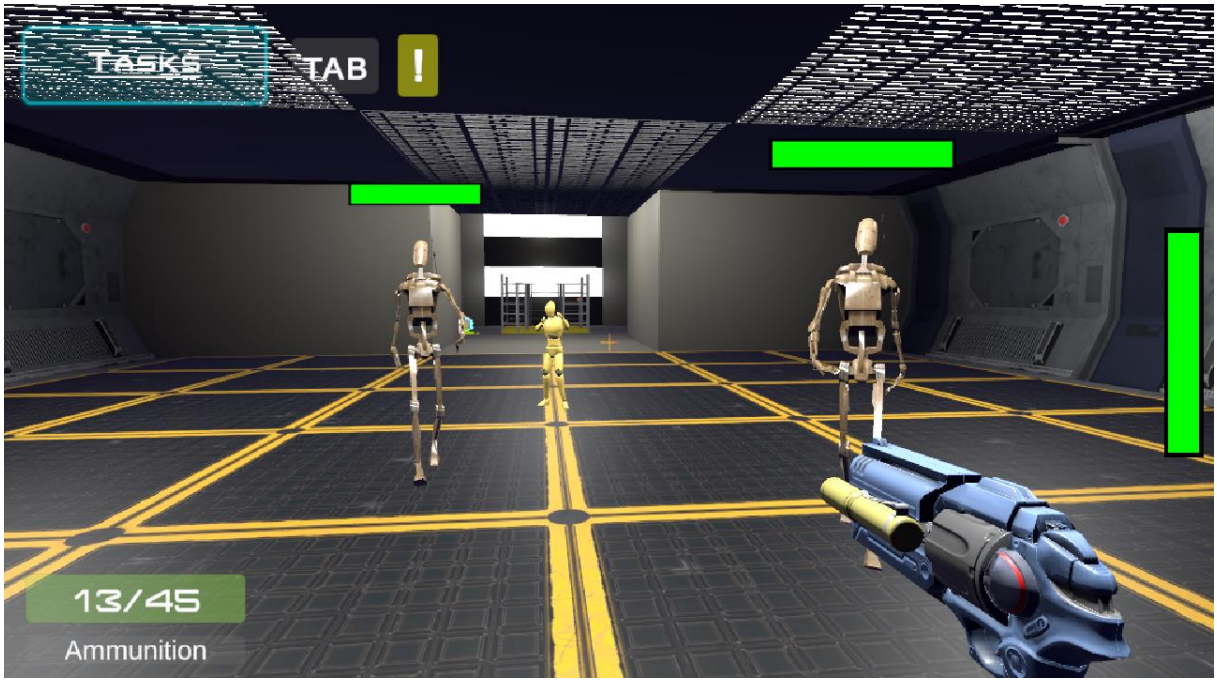


Ilustración 3.189; Tarea de eliminar enemigos

Llegado a un punto de la conversación, Kyle agradecerá el esfuerzo realizado por el jugador aportándole munición y pociones con las que empezará el juego. A su vez, se activará en la interfaz del jugador paneles en los que se le indican las distintas pociones y cantidades de ellas que el jugador tiene disponibles.

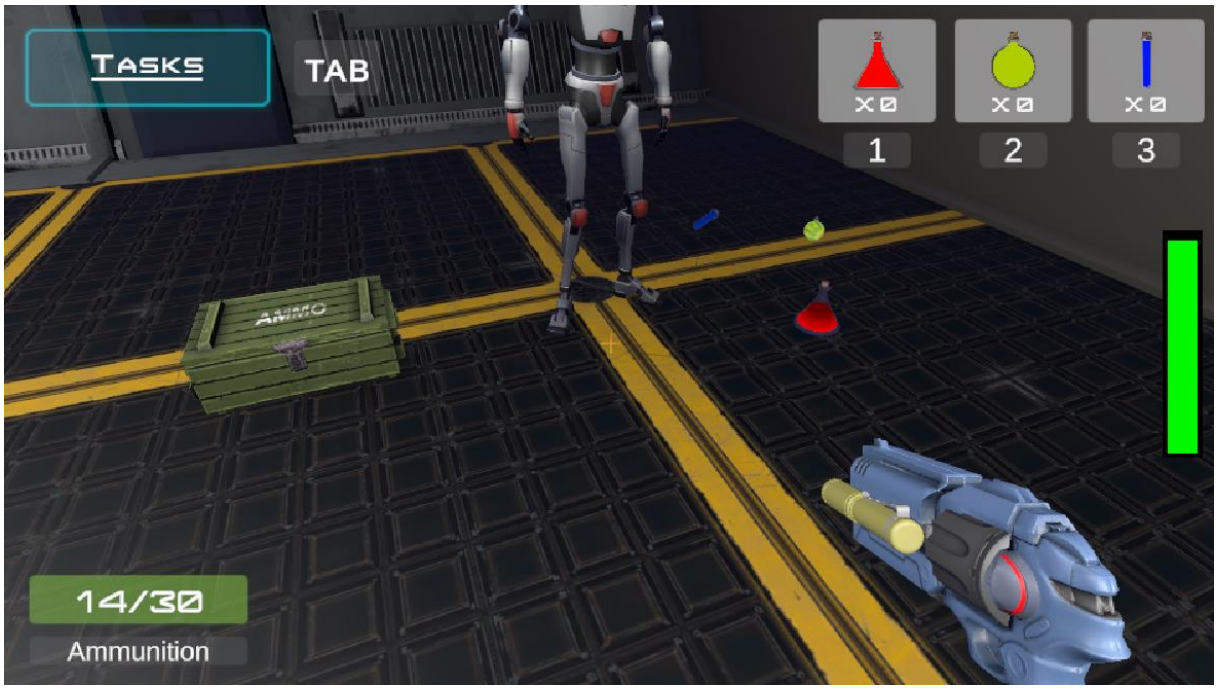


Ilustración 3.190; Recompensa del final del tutorial

Al final del diálogo con Kyle, el script eliminará la pared invisible colocada al final de la sala para que el jugador deba completar el tutorial antes de seguir explorando la nave.

Cabe comentar que, el icono encima de Kyle, el cual le indica al jugador que este puede hablar con él, tiene asociado un script llamado “TaskIconMovement” el cual hará rotar al diamante con una velocidad de rotación determinada mientras que este se desplaza verticalmente en sentido positivo y negativo constantemente. Todo esto se hace para simular que el diamante está flotando.

Por último, al empezar la partida, este script comprueba el valor de la variable “tutorialCompleted” del “GameManager”. Si este es “true”, entonces se desactivará el muro invisible del final de la sala del tutorial, ya que el jugador ha completado el tutorial y después ha muerto o ha reiniciado la partida, por lo que no es necesario “obligarle”

a que vuelva a pasar el tutorial. Además, por el mismo motivo, se le permitirá disparar y Kyle le dará pociones desde un primer momento, sin necesidad si quiera de tener que hablar con él y pasarse el tutorial. Por el contrario, si el valor de la variable "tutorialCompleted" es "false", el jugador aún no habrá completado el tutorial, por lo que deberá pasárselo para eliminar el muro invisible que le impide avanzar y el juego seguirá su transcurso normal.

3.54.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Flujo animaciones adecuado
2. Animaciones adecuadas

Test_10_01	
Nombre de la prueba o test	Completar el tutorial: guardar información
Objetivo	Cuando el jugador complete el tutorial, se debe guardar esa información
Resultado	Cuando el jugador complete el tutorial, si este muere o reaparece, no es necesario que vuelva a completar el tutorial
Comentarios	El jugador puede que no saber que no es necesario completar de nuevo el tutorial si lo ha completado antes de reaparecer o morir

Tabla 3.156; Test_10_01: completar el tutorial: guardar información

Test_10_02	
Nombre de la prueba o test	Completar el tutorial: consecuencia
Objetivo	Cuando el jugador complete el tutorial, se debe permitir el paso del jugador por el resto de la nave

Resultado	Cuando el jugador complete el tutorial, desaparece la pared invisible que le prohibía el paso hacia el resto de la nave.
Comentarios	El jugador puede que no saber que completando el tutorial, desaparece la pared invisible.

Tabla 3.157; Test_10_02: completar el tutorial: consecuencia

Test_10_03	
Nombre de la prueba o test	Aparición de los paneles de la interfaz del jugador
Objetivo	A medida que se vaya completando el tutorial, deben ir apareciendo paneles en la interfaz del jugador.
Resultado	Conforme el jugador va completando el tutorial, van apareciendo paneles en su interfaz.
Comentarios	El jugador puede no saber que significan algunos de los elementos de su interfaz.

Tabla 3.158; Test_10_03: aparición de los paneles de la interfaz del jugador

Test_10_04	
Nombre de la prueba o test	Momento de aparición de los paneles de la interfaz del jugador
Objetivo	Los paneles de la interfaz de usuario deben aparecer en el momento en el que se hace mención a ellos o cuando es relativo al dialogo que se está manteniendo.
Resultado	Los paneles de la interfaz del usuario aparecen en función de lo que el NPC le va diciendo. Por ejemplo, si el NPC está hablando de las pociones, aparecen los paneles de las pociones.
Comentarios	El jugador puede no saber que significan algunos de los elementos de su interfaz.

Tabla 3.159; Test_10_04: momento de aparición de los paneles de la interfaz del jugador

Test_10_05	
Nombre de la prueba o test	Aparición de ítems

Objetivo	Los ítems deben aparecer cuando el jugador llegue hasta ese punto del tutorial
Resultado	Cuando el jugador llega a la parte de la conversación con el NPC en el que este le habla acerca de los ítems, estos aparecen
Comentarios	Todo OK.

Tabla 3.160; Test_10_05: aparición de ítems

Test_10_06	
Nombre de la prueba o test	Activación de la funcionalidad de disparar
Objetivo	Se debe activar la función de disparo en el momento en el que el jugador deba disparar
Resultado	Cuando se le pide al jugador que dispare, se activa la función de disparar, no antes.
Comentarios	El jugador puede que no saber qué tiene que hacer para que la funcionalidad de disparo se active.

Tabla 3.161; Test_10_06: activación de la funcionalidad de disparar

Test_10_07	
Nombre de la prueba o test	Paso de los diálogos
Objetivo	El diálogo con el NPC debe ser progresivo
Resultado	Cada vez que el jugador interactúa con el NPC, este le dice una cosa distinta de la anterior.
Comentarios	El jugador puede no saber cuánto diálogo con el NPC le queda, ni cuánto lleva.

Tabla 3.162; Test_10_07: paso de los diálogos

Test_10_08	
Nombre de la prueba o test	Activación del icono de interacción
Objetivo	El icono de interacción con el NPC debe aparecer encima de este solo si le tiene que decir algo nuevo al jugador y este puede interactuar con él

Resultado	Cuando el jugador completa una tarea, aparece el icono de interacción encima del NPC
Comentarios	El jugador puede no saber que significa este icono.

Tabla 3.163; Test_10_08: activación del icono de interacción

Test_10_09	
Nombre de la prueba o test	Desactivación del icono de interacción
Objetivo	El icono de interacción con el NPC debe desaparecer de encima de este cuando el jugador tenga que completar una tarea
Resultado	Cuando se le asigna una tarea al jugador para que la complete, el icono desaparece
Comentarios	Todo OK.

Tabla 3.164; Test_10_09: desactivación del icono de interacción

Test_10_10	
Nombre de la prueba o test	Movimiento del icono de interacción
Objetivo	El icono de interacción debe moverse de arriba a abajo mientras rota.
Resultado	El icono, cuando está activo, rota sobre sí mismo a la vez de que realiza un movimiento vertical de vaivén.
Comentarios	Todo OK.

Tabla 3.165; Test_10_10: movimiento del icono de interacción

Test_10_11	
Nombre de la prueba o test	Desactivación del movimiento del jugador
Objetivo	Cuando el jugador esté hablando con el NPC, este no deberá poder moverse
Resultado	Mientras que el jugador está hablando con el NPC, este no se puede mover

Comentarios	El jugador puede no saber por qué no se puede mover y puede llegar a no saber qué tiene que hacer para volver a poder moverse.

Tabla 3.166; Test_10_11: desactivación del movimiento del jugador

Test_10_12	
Nombre de la prueba o test	Tareas aparecen en el panel de tareas
Objetivo	Las nuevas tareas encomendadas al jugador deben aparecer en el panel de tareas
Resultado	Las tareas que el NPC le encarga al jugador aparecen en el panel de tareas
Comentarios	Todo OK.

Tabla 3.167; Test_10_12: tareas aparecen en el panel de tareas

Test_10_13	
Nombre de la prueba o test	Momento en el que las tareas aparecen en panel
Objetivo	Las tareas que salgan en el panel deben aparecer cuando esa tarea se haya encomendado al jugador
Resultado	Las tareas que se le encomiendan al jugador aparecen en el panel en el momento en el que se las encomiendan.
Comentarios	Todo OK.

Tabla 3.168; Test_10_13: momento en el que las tareas aparecen en el panel de tareas

Test_10_14	
Nombre de la prueba o test	Cumplimiento de tareas
Objetivo	El jugador debe poder completar tareas.
Resultado	El jugador, si hace lo que le pide la tarea, la completa
Comentarios	El jugador puede anticipar erróneamente lo que va a pasar al completar la tarea.

Tabla 3.169; Test_10_14: cumplimiento de tareas

3.2.10.2 Freddy

Freddy es el otro NPC aliado con el que el jugador se encontrará en el juego. A diferencia de Kyle, con quien era indispensable hablar para poder avanzar en el juego puesto que este proporciona al jugador el tutorial del mismo, el cual es imprescindible superar, el jugador puede avanzar en la historia sin necesidad de hablar con Freddy. No obstante, esto es posible solo si este sabe lo que tiene que hacer para avanzar en el juego, cosa de la que Freddy informa.

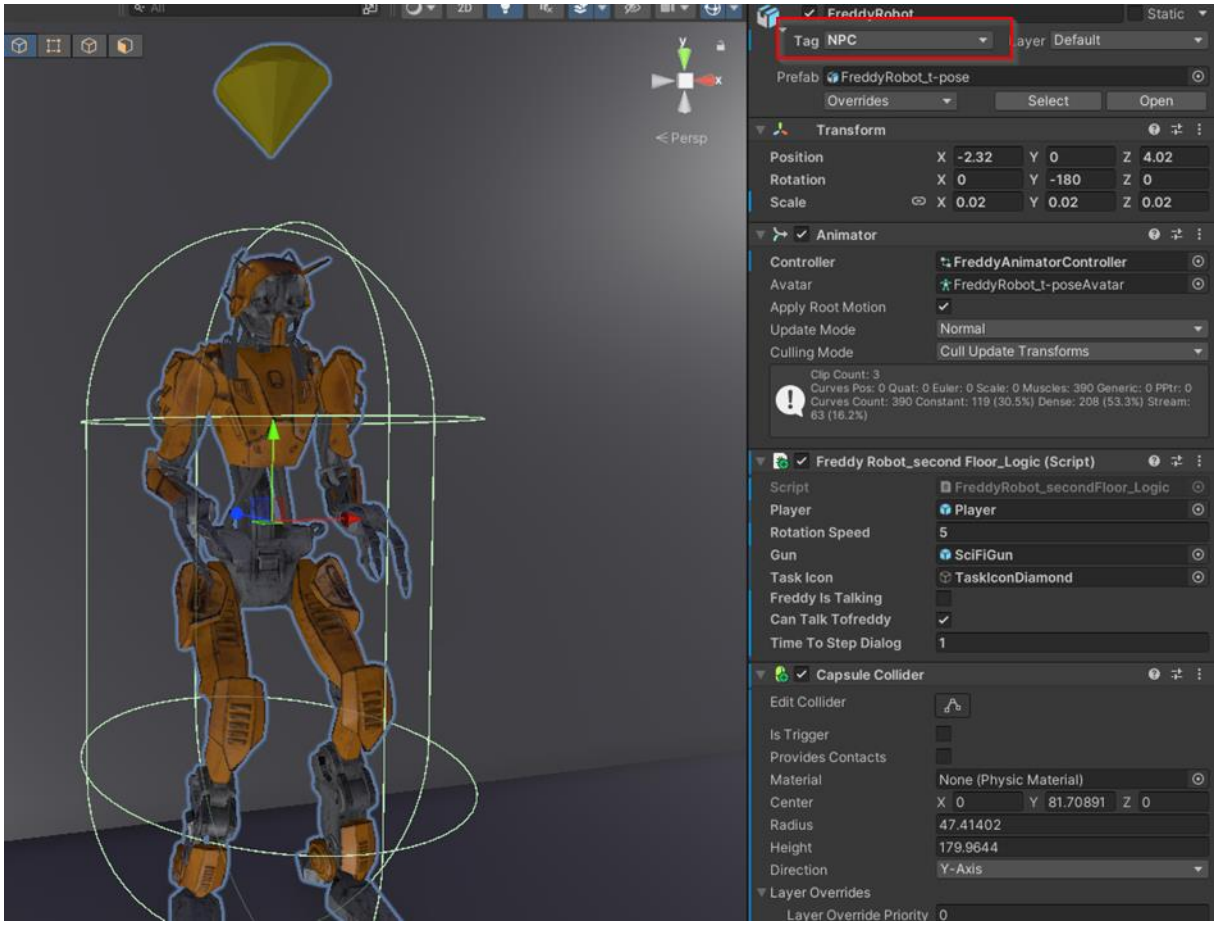


Ilustración 3.191; Componentes de Freddy

Freddy, al igual que Kyle, tiene un componente “Animator” asociado, el cual está regido por el componente “FreddyAnimatorController”, el cual controla el flujo de las animaciones del objeto. Este componente tiene la siguiente estructura:

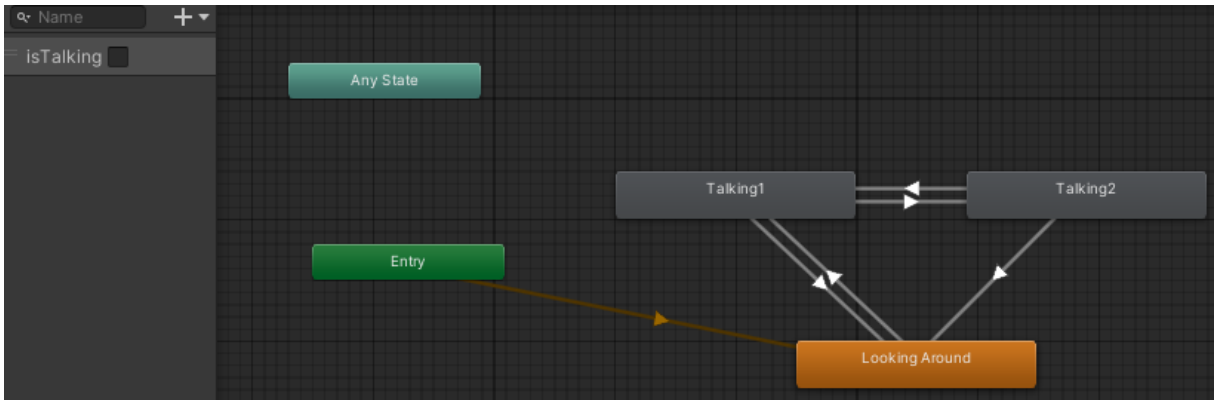


Ilustración 3.192; AnimatorController de Freddy

Este flujo de animaciones es mucho más sencillo que el de Kyle, puesto que Freddy solo tiene la misión de informar al jugador, a diferencia de Kyle que, además de informar al jugador, le encarga misiones.

Primeramente, Freddy estará realizando la animación “LookingAround”, la cual reflejará que Freddy tiene miedo y está asustado, según comenta al jugador, por la presencia de StarLord. En el momento en el que el parámetro booleano “isTalking” tenga valor “true”, Freddy pasará a realizar el ciclo de animaciones de hablar, al igual que lo hacía Kyle. Independientemente del momento en el que se encuentre Freddy, si este está hablando y el parámetro “isTalking” pasa a tener valor “false”, Freddy volverá a realizar la animación “LookingAround”.

Además de un componente “Animator”, Freddy tiene asociado un script llamado “FreddyRobot_secondFloor_Logic”, el cual controlará toda la conversación con el jugador.

Al igual que Kyle, Freddy tiene asociada la etiqueta “NPC”, por lo que, si el jugador interactúa con él, se activará el método “Activate()” de la clase interfaz “IAction”, método el cual hereda y sobrescribe el script “FreddyRobot_secondFloor_Logic”, asociado a Freddy.

Este método lo que hará es que, cada vez que el jugador interactúe con el NPC, se comprobará si el jugador puede hablar con Freddy o no. Esto es controlado por la variable booleana “canTalkToFreddy”, de forma que si tiene valor “true” el jugador

podrá hablar con Freddy, y si tiene valor “false” el jugador no podrá hablar con él. Si “canTalkToFreddy” tiene valor “true”, lo que se hace es que, si no está hablando Freddy, controlado de la misma manera por la variable booleana “freddyIsTalking”, se empieza el dialogo, y si está hablando Freddy, se sigue el dialogo por donde se quedó.

Para empezar el dialogo, lo que se hace es que se le obliga al jugador a mirar a Freddy y se le prohíbe moverse, disparar y mover la cámara. Además, se activa el panel de dialogo en el que se mostrará el diálogo que el jugador está manteniendo con Freddy. Con estos prerrequisitos, se continúa el dialogo por el siguiente mensaje.

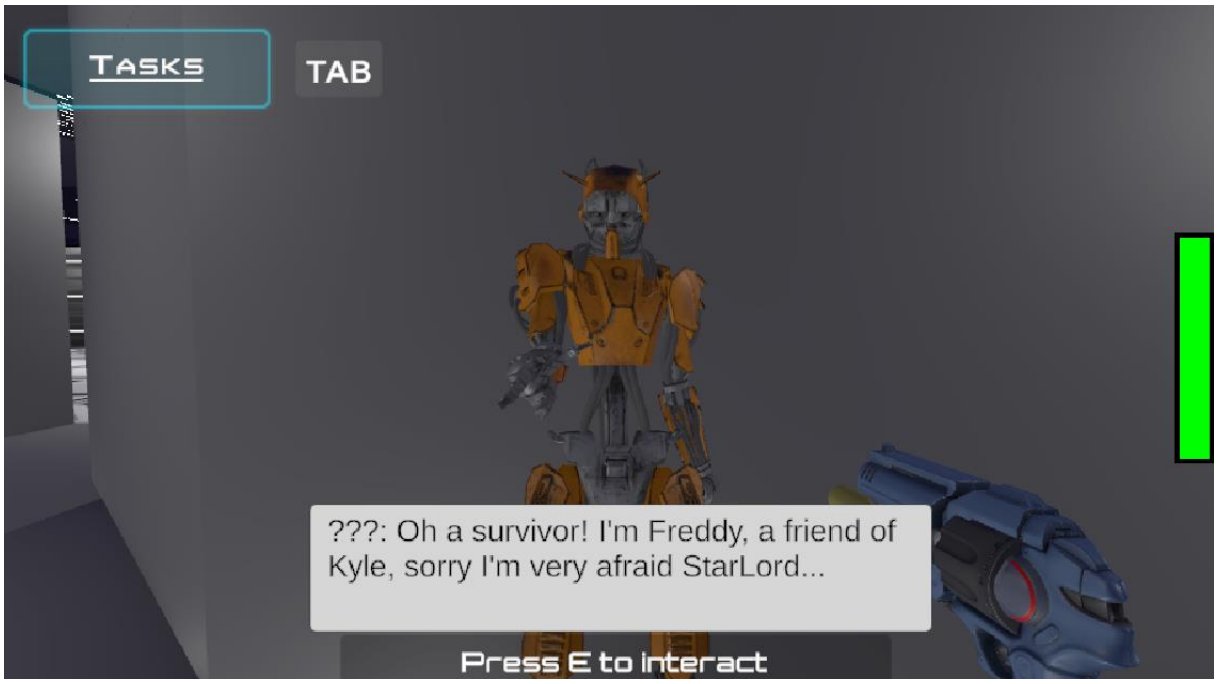


Ilustración 3.193; Diálogo con Freddy

Tras toda la conversación con Freddy, el jugador volverá a poder moverse y disparar para seguir con su partida. No obstante, no desaparecerá el icono de interacción que tiene en su cabeza al igual que Kyle, y esto es porque el jugador podrá seguir hablando con Freddy. Freddy, en esta situación, le explicará al jugador qué es la cascada de ácido que tiene él en frente.

En cuanto al icono de interacción encima de Freddy, este tendrá la misma lógica que el icono de encima de Kyle.

Cabe destacar que ambos NPCs, en sus diálogos, al principio de ellos no se verá el nombre del NPC con el que el jugador está hablando. A partir del momento en el que el NPC le dice su nombre al jugador en alguno de sus diálogos, este aparecerá en el resto.

3.56.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Activación de funcionalidad de disparar
2. Paso de diálogos
3. Activación del icono de interacción
4. Desactivación del icono de interacción
5. Movimiento del icono de interacción
6. Desactivación del movimiento del jugador
7. Tareas aparecen en el panel
8. Momento en el que las tareas aparecen en el panel

3.2.11 Menús y acciones

En este apartado se van a ver las distintas funcionalidades de los botones y los componentes interactuables por el jugador desde los distintos menús.

Empezando por el menú de inicio, en este menú se pueden apreciar dos botones.

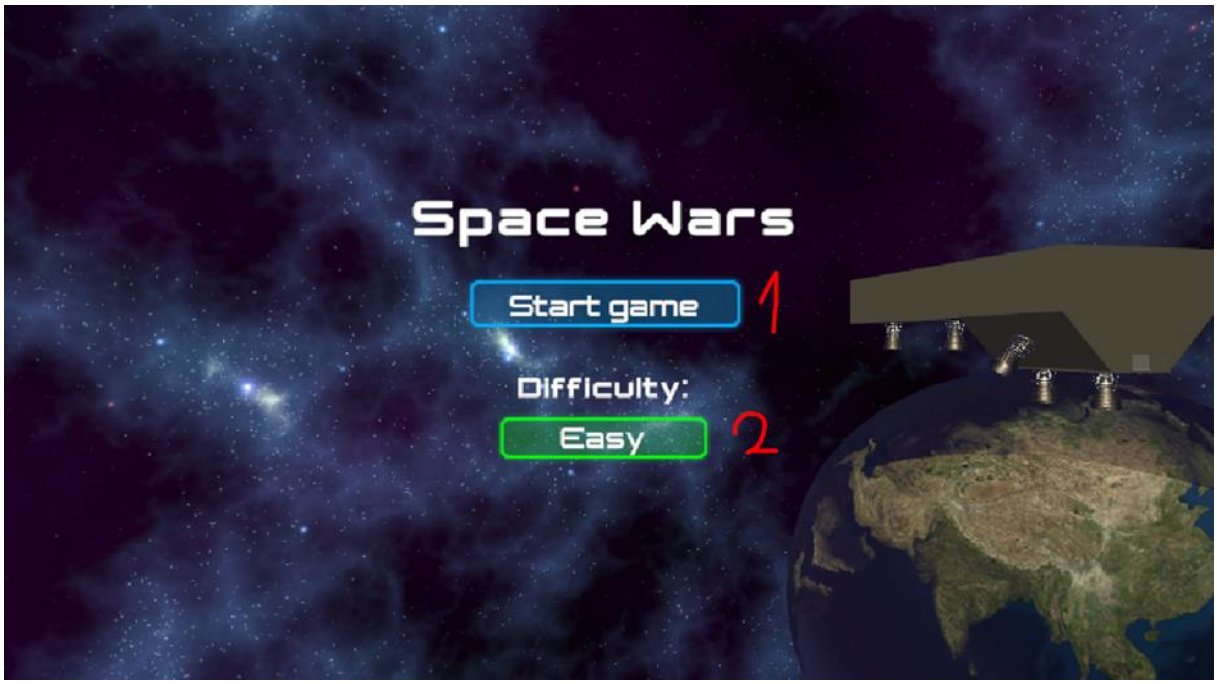


Ilustración 3.194; Botones del menú principal

El primer botón llamará al método “StartGame()” del “GameLogic”. Mientras que el segundo botón llamará al método “ToggleState()” del “GameLogic”.

En cuanto al menú de opciones dentro del juego, tendremos tres botones y dos barras de desplazamiento.

Ilustración 3.195; Botones y sliders del menú de opciones

El primer botón llamará al método “Resume()” del “GameLogic”. El segundo, llamará al método “MenuPrincipal()” del “GameLogic”. Por último, el tercer botón llamará al método “ExitGame()” del “GameLogic”. En cuanto a los “Sliders”, los dos primeros llamarán al método “ChangeVolume()” del “SoundManager”, mientras que el segundo llamará al método “ChangeSensibility()” del “GameManager”. No obstante, el primero de los sliders variará el volumen de los sonidos, mientras que el segundo variará el volumen de la música.

Por último, hablando a cerca de la pantalla de muerte, el jugador tendrá dos opciones.

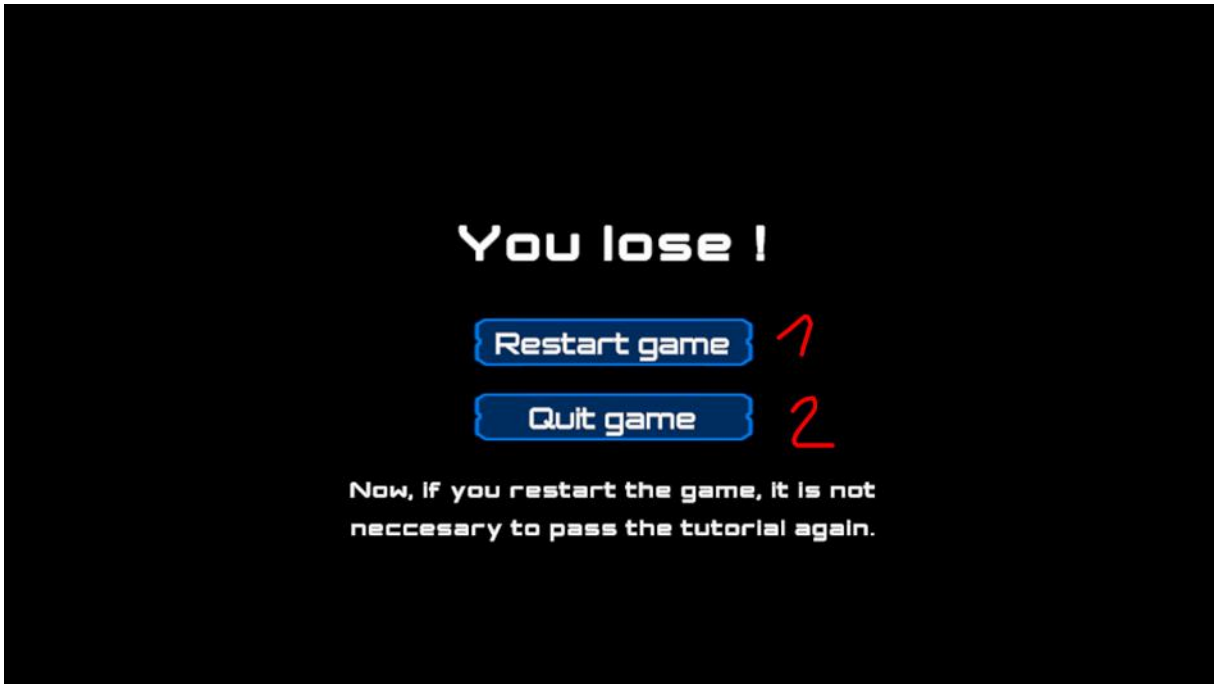


Ilustración 3.196; Botones de la pantalla de muerte

El primer botón llamará al método “MenuPrincipal()” del “GameLogic”. Mientras que el segundo ejecutará el método “ExitGame()” del “GameLogic”.

3.58.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_11_01	
Nombre de la prueba o test	Elementos interactuables
Objetivo	Los distintos elementos interactuables del menú deben poder ser interactuables
Resultado	El jugador puede interactuar con los elementos interactuables de los menús, como botones, sliders, …
Comentarios	El jugador puede que no saber que se puede interactuar con algún elemento

Tabla 3.170; Test_11_01: elementos interactuables

Test_11_02	
Nombre de la prueba o test	Función de los elementos interactuables
Objetivo	Los distintos elementos interactuables del menú deben tener algún efecto en el juego
Resultado	El jugador puede interactuar con los elementos interactuables de los menús y estos tienen un efecto en el juego. Por ejemplo, si el jugador interactúa con el botón de reaparición, el jugador volverá al menú principal.
Comentarios	El jugador puede que no saber qué hace el interactuar con el elemento.

Tabla 3.171; Test_11_02: función de los elementos interactuables

3.2.12 Cambio de dificultad

En el comienzo del juego habrá una opción para cambiar la dificultad de este antes de empezar.

Ilustración 3.197; Botón de cambio de dificultad del menú principal

Los atributos de las diferentes lógicas del juego variarán en función de la dificultad del juego elegida:

1. Si es facil:
    1. Cantidad de enemigos instanciados por spawn entre 1 y 3
    2. Todos los enemigos hacen 10 unidades de daño menos de lo normal
    3. El arma del jugador hace 10 unidades más de daño
    4. Se instancia la colección fácil de ítems

2. Si es media:
    1. Cantidad de enemigos instanciados por spawn entre 1 y 5
    2. Todos los enemigos hacen 5 unidades de daño menos de lo normal

3. El arma del jugador hace 5 unidades más de daño
4. Se instancia la colección media de ítems

3. Si es difícil:
    1. Cantidad de enemigos instanciados por spawn entre 2 y 5
    2. Todos los enemigos hacen el daño normal
    3. El arma del jugador hace el daño normal
    4. Se instancia la colección difícil de ítems

3.60.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_12_01	
Nombre de la prueba o test	Efecto del cambio de dificultad
Objetivo	El cambiar la dificultad repercute en el juego.
Resultado	En función de la dificultad del juego, este cambia de una forma u otra.
Comentarios	El jugador puede no saber que varía entre una dificultad y otra.

Tabla 3.172; Test_12_01: efecto del cambio de dificultad

3.2.13 Ítems

Los ítems son uno de las cosas más importantes para el jugador, puesto que estos le ayudarán durante el transcurso del juego. Algunos de estos se usarán para restablecer la vida del jugador o proporcionarle mejoras en sus habilidades físicas, otros se lanzarán para hacer daño al enemigo.

3.2.13.1 Poción de curación

La poción de curación es indispensable para el jugador, puesto que le cura ante un daño producido por el enemigo. Para ello, la poción tiene asignado un script llamado “HealPotion”. Este script controlará si, cuando el jugador pulse la tecla de uso de pociones, la tecla ‘q’, tiene la poción de curación seleccionada y tiene pociones suficientes. En tal caso, el script llamará a los métodos “UpdatePlayerLife()” y “UsedItem()” del “GameManager”. Concretamente, al método “UpdatePlayerLife()” se le pasará como parámetro la cantidad de vida que la poción le cura al jugador, definida previamente.

Cabe destacar que el script solo le permitirá al jugador curarse si este no tiene la vida al máximo. Esto se hace para que el jugador no desperdicie la poción, ya que, al tener la vida al máximo, la poción se gastaría, pero no le surtiría efecto.

3.62.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_13_01	
Nombre de la prueba o test	Uso de una poción
Objetivo	El jugador debe poder usar pociones
Resultado	El jugador puede usar pociones si este selecciona la poción a usar
Comentarios	El jugador puede no saber cómo usar pociones y cómo usar la poción deseada.

Tabla 3.173; Test_13_01: uso de una poción

Test_13_02	
Nombre de la prueba o test	No uso de una poción
Objetivo	El jugador no debe poder usar pociones en ciertas situaciones

Resultado	El jugador no puede usar una determinada poción si no tiene suficientes pociones de este tipo en su inventario
Comentarios	El jugador puede no saber si tiene o no pociones de ese tipo en su inventario.

Tabla 3.174; Test_13_02: no uso de una poción

Test_13_03	
Nombre de la prueba o test	Efecto del uso de una poción
Objetivo	El usar una poción debe producir una acción
Resultado	El jugador, al usar una poción, produce una acción en el juego, ya sea lanzando una poción, curándose o mejorando sus habilidades físicas
Comentarios	El jugador puede no saber qué pasará al usar la poción.

Tabla 3.175; Test_13_03: efecto del uso de una poción

Test_13_04	
Nombre de la prueba o test	Gastar una poción
Objetivo	El usar una poción debe conllevar gastarla
Resultado	Al usar una poción, se reduce en uno, la cantidad de pociones que el jugador tiene de ese tipo
Comentarios	El jugador puede no saber que al usar la poción se gasta y puede incluso no saber cuántas se gastan.

Tabla 3.176; Test_13_04: gastar una poción

3.2.13.2 Poción de velocidad

En cuanto a la poción de velocidad, esta le aporta al jugador mayor velocidad de movilidad durante un periodo de tiempo definido. Para ello, la poción está ligada a un script llamado “SpeedPotionEffect”, el cual, al igual que el script “HealPotionEffect”, comprobará que el jugador tenga la poción de velocidad y que tenga suficientes pociones de velocidad en el momento en el que el jugador pulse el botón de usar la poción.

Para poder aportar este efecto al jugador, el script lo que hace es duplicar la velocidad de movimiento (tanto andando como corriendo) del jugador. Cuando varíen estos valores, el script llamará al método “ResetPlayerSpeed()” del “GameManager()” para que estos valores vuelvan a la normalidad tras un tiempo definido el cual se le pasa al método por parámetro. Además, el script también llama al método “UsedItem()” del “GameManager” para decrementar la cantidad de pociones de velocidad que el jugador tiene en el inventario una vez que este haya usado la poción y se le haya aportado el efecto de la misma.

3.64.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Uso de la poción
2. No uso de la poción
3. Gastar una poción
4. Efecto del uso de la poción

Test_13_05	
Nombre de la prueba o test	Duración del efecto de una poción
Objetivo	El efecto de la poción, si no es de curación, debe desaparecer tras un tiempo
Resultado	Cuando el jugador usa la poción, el efecto desaparece tras un tiempo
Comentarios	El jugador puede no saber cuánto tiempo dura el efecto ni cuánto le queda de tiempo antes de que el efecto desaparezca.

Tabla 3.177; Test_13_05: duración del efecto de una poción

3.2.13.3 Poción de ácido

Esta poción es la única que se puede lanzar de todas las pociones. Se lanza para crear una piscina o charco de ácido al colisionar con el suelo, la cual infringirá daño a la entidad, ya sea enemigo o jugador, que entre en contacto con ella.

Esta poción cuenta por tanto con dos funcionalidades, el propio efecto de infligir daño del ácido y la funcionalidad de poder lanzar la poción.

3.2.13.3.1 PotionThrower

Este es un elemento que controla la trayectoria del lanzamiento de pociones del jugador.

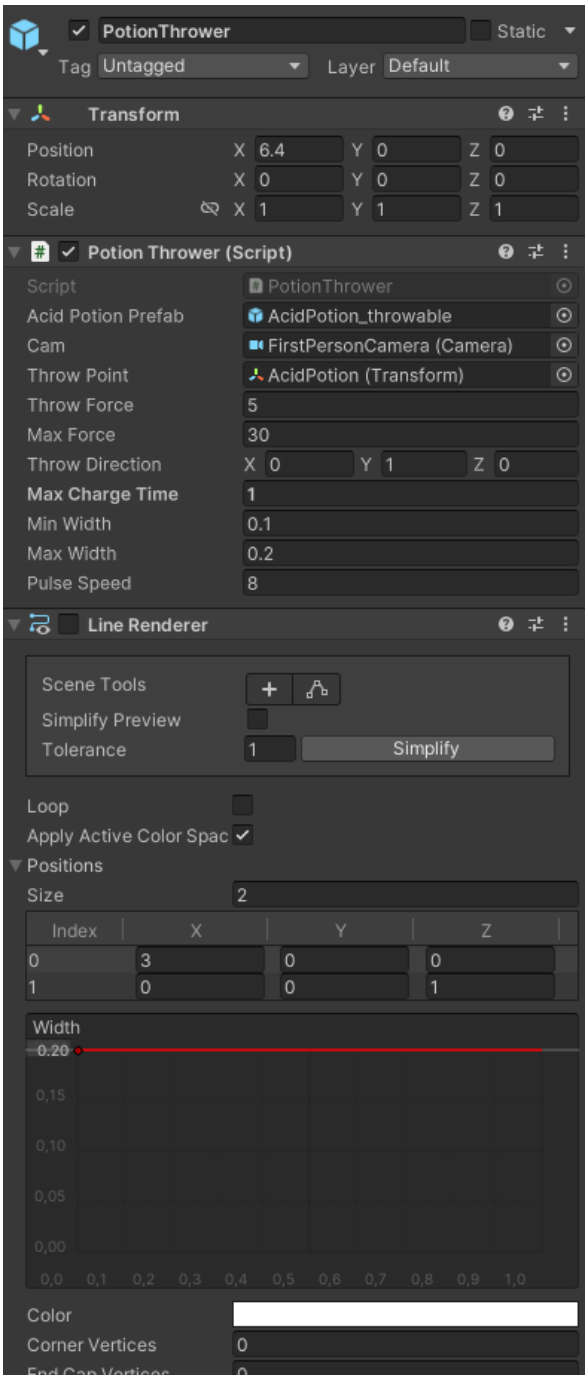


Ilustración 3.198; Componentes del PotionThrower

Tiene un componente “LineRenderer” [100] asignado, con el que crea y renderiza una línea curva que indica la trayectoria del lanzamiento de la poción.

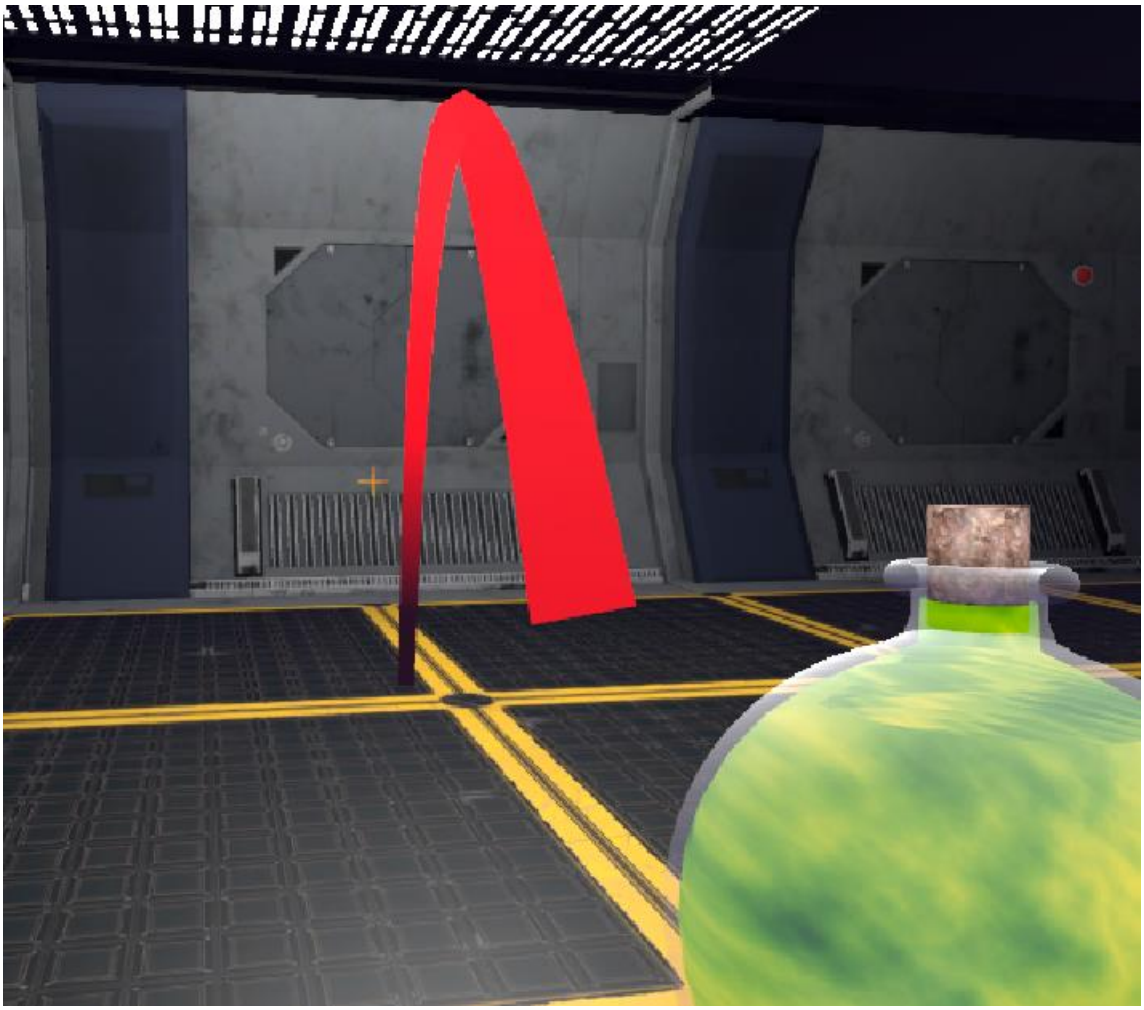


Ilustración 3.199; Trayectoria de lanzamiento de la poción de ácido

Esta línea está controlada por el script “PotionThrower”, asignado al objeto. Para controlar la línea, el script lo que hace es que, si el jugador pulsa la tecla de lanzar pociones, se inicializan las variables para comenzar el lanzamiento y se activa la línea. Si la tecla de lanzamiento se mantiene pulsada, se incrementa el valor de la variable “chargeTime”, la cual controla la fuerza de lanzamiento en base al tiempo de carga del lanzamiento. A la vez de aumentar el valor de “chargeTime”, se va se recalcula la dirección y velocidad de lanzamiento de la poción, y por tanto, los valores de las posiciones de la línea de trayectoria.

Al soltar la tecla de lanzamiento, se instancia la poción y se lanza con la fuerza acumulada en base al tiempo de carga y oculta la línea de lanzamiento.

Todo esto se puede realizar si la poción de ácido está activa y hay suficiente cantidad de pociones de ácido en el inventario del jugador.

También destacar que el tiempo de carga está limitado por el parámetro “maxChargeTime” para evitar que la línea de trayectoria y fuerza del lanzamiento aumente hasta el infinito.

A continuación, se aporta el pseudocódigo del script:

```
Update()
  Si se presiona la tecla "Q" y la pocion "AcidPotion" está activa y hay suficiente cantidad
    StartThrowing()

  Si isCharging es verdadero
    ChargeThrow()

  Si se libera la tecla "Q" y la pocion "AcidPotion" está activa y hay suficiente cantidad
    ReleaseThrow()
    GameManager.UsedItem(acidPotionPrefab)

StartThrowing()
  isCharging <- verdadero
  chargeTime <- 0
  Activa componente LineRenderer: linea de trayectoria de lanzamiento

ChargeThrow()
  Si chargeTime < maxChargeTime
    Incrementa chargeTime por Time.deltaTime

  Si no
    chargeTime <- maxChargeTime

  newThrowDirection <- normaliza cam.transform.forward + throwDirection
  potionVelocity <- newThrowDirection * mínimo entre chargeTime * throwForce y maxForce
  ShowTrajectory(throwPoint.position + throwPoint.forward, potionVelocity)

ReleaseThrow()
  ThrowPotion(mínimo entre chargeTime * throwForce * 1.5  y  maxForce * 1.5)
  isCharging <- falso
  Desactiva componente LineRenderer: línea de trayectoria de lanzamiento

ThrowPotion(force)
  spawnPosition <- throwPoint.position + cam.transform.forward
  potion <- instancia acidPotionPrefab en spawnPosition con rotación de cam.transform
  rb <- componente Rigidbody de potion

  Si rb no es nulo
    newThrowDirection <- normaliza cam.transform.forward + throwDirection
    Aplica fuerza newThrowDirection * force en rb con modo VelocityChange

ShowTrajectory(origin, speed)
  puntos <- vector de Vector3 de tamaño 100
  Cantidad de posiciones de trajectoryLine <- 100

  Para cada punto
    time <- i * 0.1
    puntos[i] <- origin + speed * time + 0.2 * Physics.gravity * time * time

  Posiciones de trajectoryLine <- puntos
```

Ilustración 3.200; Pseudocódigo del script PotionThrower

Al soltar la tecla de uso de poción ‘q’, como se ha comentado previamente, se instancia la poción de ácido con una fuerza y una dirección relativa a la línea de trayectoria dibujada.

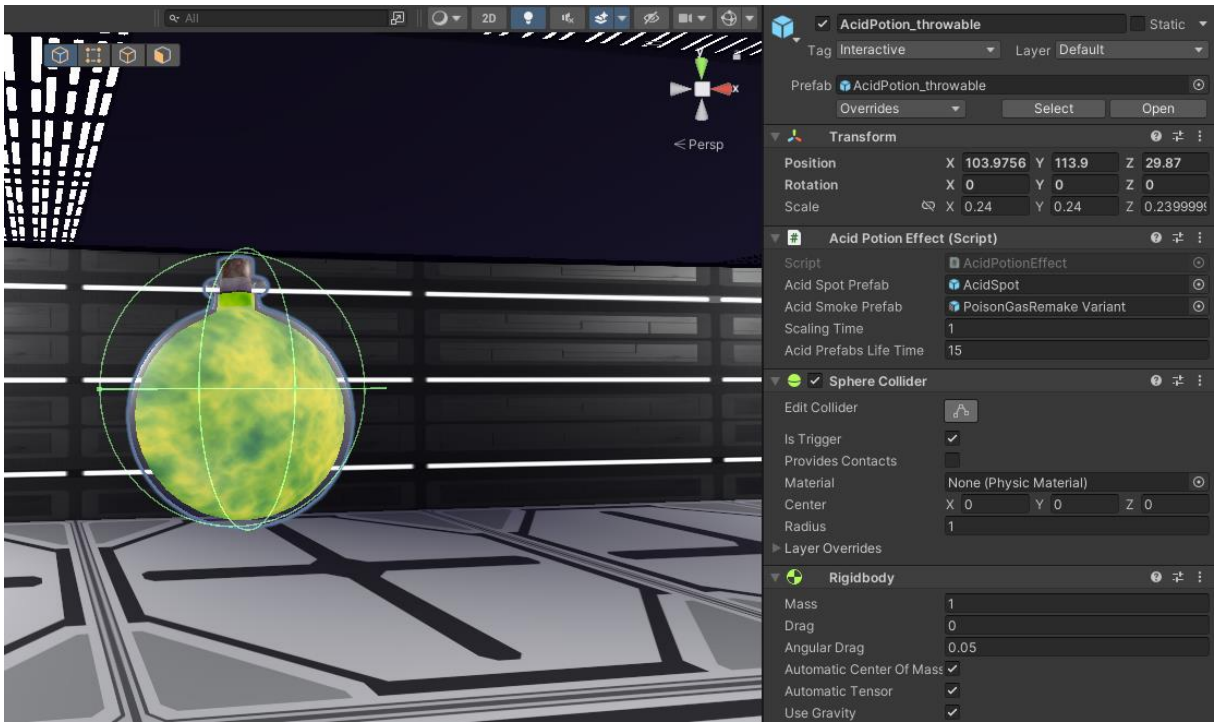


Ilustración 3.201; Componentes de la poción de ácido lanzable

Esta poción instanciada tiene un componente “SphereCollider” [101] con la opción “isTrigger” marcada. Además, tiene un script asociado llamado “AcidPotionEffect”. Este script controla si el componente “SphereCollider” entra en contacto con cualquier elemento del mapa. Si el elemento con el cual entra en contacto tiene una etiqueta “Wall” o “Floor”, el script instanciará un charco de ácido, y un gas verde cuya función es simplemente estética y sirve para hacer entender al jugador que es un charco de ácido y que no le conviene tocarlo.

Este script instanciará los prefabs con un tamaño muy reducido y llamará a la corrutina “ScaleOverTime()”, con la cual ambas instancias aumentarán poco a poco de tamaño hasta un tamaño final definido. Esto se hace para dar la sensación de que el charco de ácido se está expandiendo.

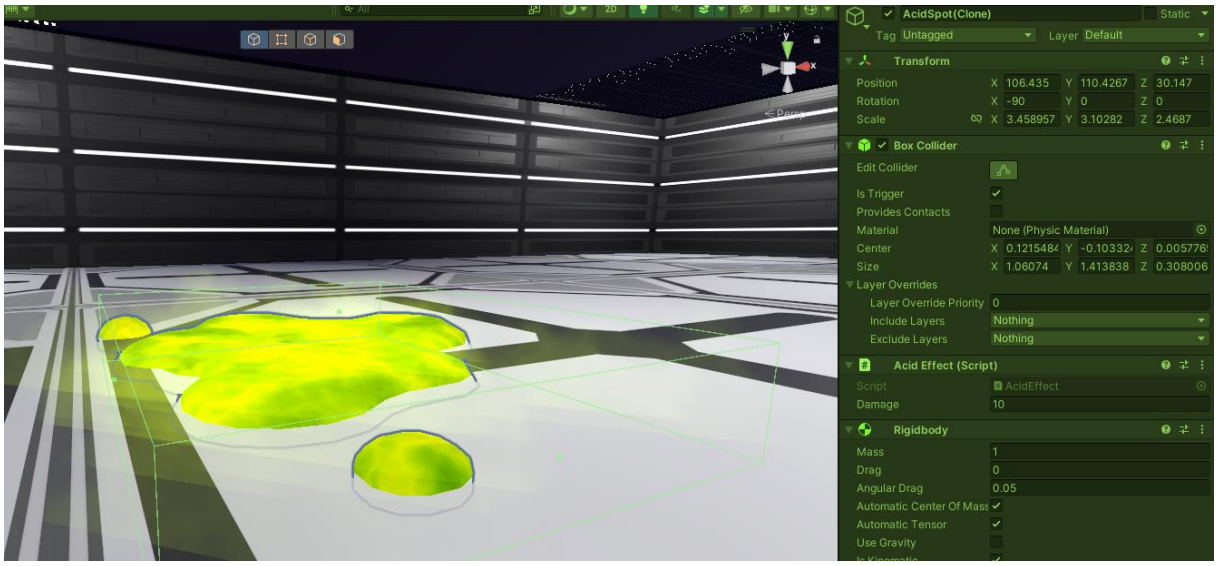


Ilustración 3.202; Componentes de la piscina de ácido creada

Esta piscina de ácido instanciada tiene asociado un componente “BoxCollider” asignado con la opción “isTrigger” marcada, de forma que, el script llamado “AcidEffect”, el cual también tiene asociado este objeto, controlará si alguna entidad entra en contacto con el componente “BoxCollider”, momento en el cual se le empezará a aplicar daño a la vida de esa entidad con la que se hace contacto.

3.66.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Tests anteriormente vistos que también se aplican a este enemigo:

1. Uso de la poción
2. No uso de la poción
3. Gastar una poción
4. Efecto del uso de la poción

Test_13_06	
Nombre de la prueba o test	Lanzamiento de una poción

Objetivo	Se deben poder lanzar ciertas pociones
Resultado	El jugador puede lanzar la poción de ácido
Comentarios	El jugador puede no saber que la poción que tiene en la mano se puede lanzar o no.

Tabla 3.178; Test_13_06: lanzamiento de una poción

Test_13_07	
Nombre de la prueba o test	Trayectoria de lanzamiento de una poción
Objetivo	Mientras que se carga el lanzamiento, debe aparecer la trayectoria que tomará la poción al ser esta lanzada
Resultado	Mientras que el jugador carga el lanzamiento da la poción, se puede ver la trayectoria que tomaría la poción si esta es lanzada
Comentarios	El jugador puede no llegar a pensar que esa sea la trayectoria de la poción.

Tabla 3.179; Test_13_07: trayectoria de lanzamiento de una poción

Test_13_08	
Nombre de la prueba o test	Vibración de la trayectoria de lanzamiento de una poción
Objetivo	La trayectoria del lanzamiento de la poción debe vibrar mientras se carga el lanzamiento
Resultado	La trayectoria del lanzamiento de la poción de ácido vibra mientras que se carga el lanzamiento de la poción
Comentarios	Todo OK.

Tabla 3.180; Test_13_08: vibración de la trayectoria de lanzamiento de una poción

Test_13_09	
Nombre de la prueba o test	Carga del lanzamiento de una poción
Objetivo	El lanzamiento de la poción debe poder cargarse
Resultado	El jugador puede mantener pulsado el botón de lanzamiento para cargar el lanzamiento y que la poción llegue más lejos al ser lanzada.

Comentarios	El jugador puede no saber que, si mantiene pulsado el botón de lanzamiento, el lanzamiento se carga, haciendo que la poción llegue más lejos.

Tabla 3.181; Test_13_09: carga del lanzamiento de una poción

Test_13_10	
Nombre de la prueba o test	Poción rota
Objetivo	La poción lanzada, debe romperse al impactar con una superficie sólida
Resultado	La poción lanzada por el jugador se rompe al tocar una superficie sólida.
Comentarios	El jugador puede no saber si la poción se va a romper o no cuando esta impacta sombre una superficie sólida.

Tabla 3.182; Test_13_10: poción rota

Test_13_11	
Nombre de la prueba o test	Instanciación del charco de ácido
Objetivo	Se debe instanciar un charco de ácido al romperse la poción de ácido lanzada
Resultado	La poción de ácido crea un charco de ácido al romperse por impactar sobre una superficie sólida
Comentarios	El jugador puede no saber qué va a pasar cuando la poción se rompa al tocar alguna superficie sólida.

Tabla 3.183; Test_13_11: instanciación del charco de ácido

Test_13_12	
Nombre de la prueba o test	Instanciación del gas tóxico
Objetivo	Debe instanciarse un gas tóxico a la misma vez que el charco de ácido
Resultado	El gas tóxico aparece a la vez que el charco de ácido
Comentarios	Todo OK.

Tabla 3.184; Test_13_12: instanciación del gas tóxico

Test_13_13	
Nombre de la prueba o test	Aumento progresivo del tamaño del charco de ácido
Objetivo	El charco de ácido instanciado debe aumentar de tamaño progresivamente con el paso del tiempo
Resultado	El charco de ácido se instancia pequeño pero crece de tamaño a medida que el tiempo pasa hasta alcanzar un tamaño definitivo
Comentarios	El jugador puede no saber qué tamaño final va a alcanzar el charco de ácido.

Tabla 3.185; Test_13_13: aumento progresivo del tamaño del charco de ácido

Test_13_14	
Nombre de la prueba o test	Velocidad de aumento de tamaño del charco de ácido
Objetivo	El charco de ácido instanciado debe tener una velocidad de aumento de tamaño adecuada
Resultado	El charco crece rápidamente hasta un tamaño final
Comentarios	El jugador puede percibir esta velocidad como demasiado rápida.

Tabla 3.186; Test_13_14: velocidad de aumento de tamaño del charco de ácido

Test_13_15	
Nombre de la prueba o test	Daño del charco de ácido
Objetivo	El charco de ácido debe poder infligir daño a todo el que lo toca
Resultado	El charco de ácido infringe daño a toda entidad que lo toca, incluido el jugador
Comentarios	El jugador puede no saber que, si toca el charco de ácido que él ha creado, le puede hacer daño al igual que a un enemigo cualquiera.

Tabla 3.187; Test_13_15: daño del charco de ácido

3.2.13.4 Caja de munición

En cuanto a la caja de munición, también es uno de los objetos indispensables para el jugador, ya que le aporta al jugador munición para su arma con la cual el jugador elimina enemigos y avanza en el juego.

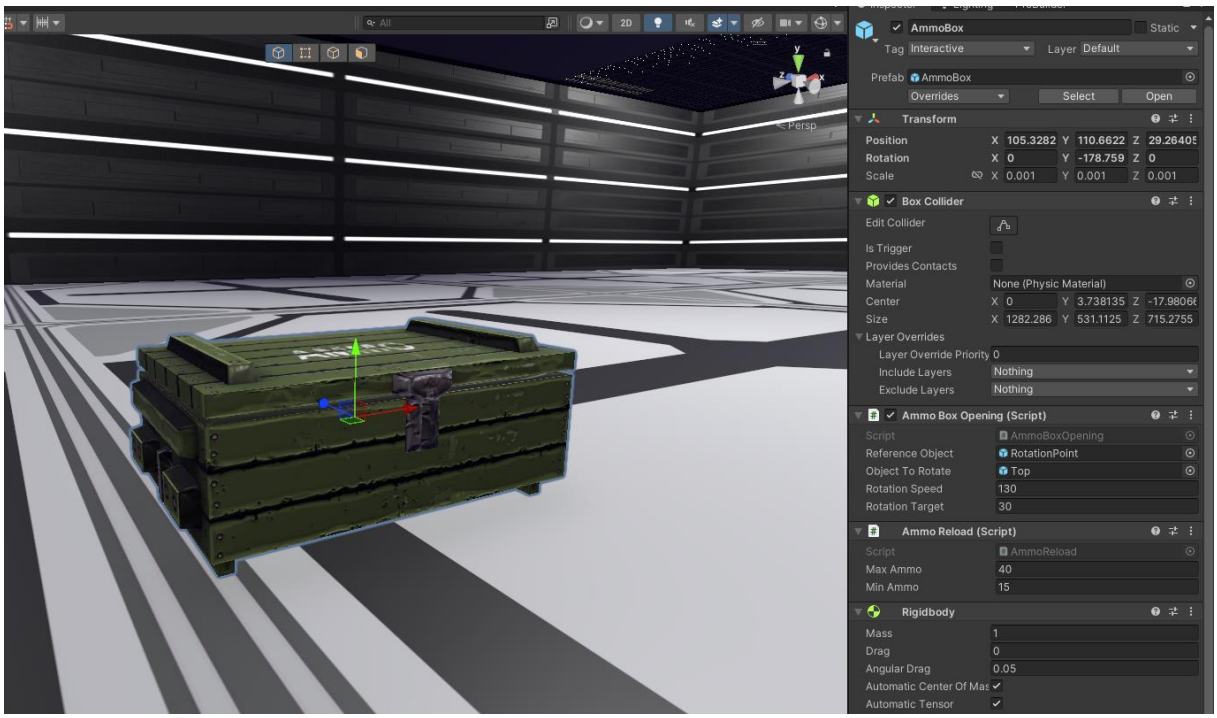


Ilustración 3.203; Componentes de la caja de munición

Las cajas de munición, las cuales el jugador podrá encontrarse durante su partida, tienen la etiqueta “Interactive” asociadas, gracias a esto es por lo que el jugador podrá interactuar con ellas. Estas tienen un script asociado llamado “AmmoReload”, el cual hereda y sobrescribe el método “Activate()” de la clase interfaz “IAction”, de forma que cuando el jugador interactúe con alguna de ellas, se ejecutará el método “Activate()” del script. Este método calculará una cantidad de munición aleatoria entre dos valores predefinidos y llamará al método “UpdateGunAmmo()” del “GameManager” pasándole este valor recién calculado por parámetro.

Cabe destacar que este objeto tiene otro script llamado “AmmoBoxOpening” el cual hace que la tapa de la caja con la que el jugador ha interactuado rote alrededor de un objeto de referencia. Esto se hace para que el jugador sepa que la caja ha sido abierta y que ya no se puede volver a interactuar con ella.

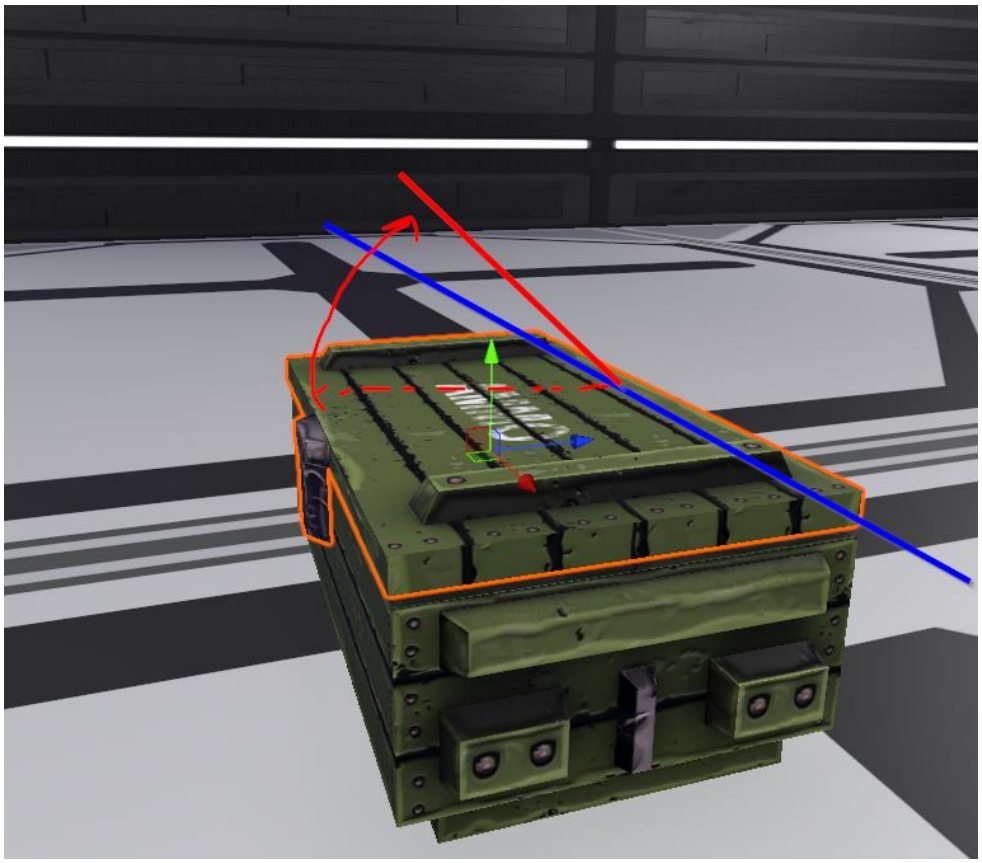


Ilustración 3.204; Apertura de la tapa de la caja de munición

3.68.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_13_16	
Nombre de la prueba o test	Rotación de la tapa de la caja de munición
Objetivo	La tapa de la caja de munición debe rotar adecuadamente
Resultado	La tapa de la caja de munición rota hasta un punto adecuado y en sentido correcto
Comentarios	Todo OK.

Tabla 3.188; Test_13_16: rotación de la tapa de la caja de munición

Test_13_17	
Nombre de la prueba o test	Velocidad de rotación de la tapa de la caja de munición

Objetivo	La velocidad de rotación de la tapa de la caja de munición debe ser adecuada.
Resultado	La tapa de la caja de munición rota a una velocidad adecuada, sin ser muy lenta, ni rápida
Comentarios	Todo OK.

Tabla 3.189; Test_13_17: velocidad de rotación de la tapa de la caja de munición

Test_13_18	
Nombre de la prueba o test	Momento de rotación de la tapa de la caja de munición
Objetivo	La tapa de la caja de munición debe rotar en el momento apropiado
Resultado	La tapa de la caja de munición empieza a rotar justo cuando el jugador interactúa con la caja de munición
Comentarios	Todo OK.

Tabla 3.190; Test_13_18: momento de rotación de la tapa de la caja de munición

Test_13_19	
Nombre de la prueba o test	No volver a poder interactuar con la caja de munición
Objetivo	El jugador no debe volver a poder interactuar con la caja de munición una vez abierta
Resultado	El jugador no puede volver a interactuar con la caja de munición si este la ha abierto antes
Comentarios	El jugador puede no saber que no puede volver a interactuar con la caja de munición una vez abierta, esperando poder interactuar con ella varias veces

Tabla 3.191; Test_13_19: no volver a poder interactuar con la caja de munición

Test_13_20	
Nombre de la prueba o test	Munición aumenta
Objetivo	La munición de reserva del jugador debe aumentar al interactuar con una caja de munición

Resultado	La munición de reserva del jugador aumenta al interactuar con la caja de munición
Comentarios	Todo OK.

Tabla 3.192; Test_13_20: munición aumenta

Test_13_21	
Nombre de la prueba o test	Cantidad de munición que aumenta
Objetivo	La munición de reserva del jugador debe aumentar una cantidad aleatoria al interactuar el jugador con una caja de munición
Resultado	La munición de reserva del jugador aumenta una cantidad aleatoria entre 25 y 60.
Comentarios	El jugador puede no saber cuánta munición le va a dar la caja de munición y esperar que le dé más de lo que realmente le ha dado.

Tabla 3.193; Test_13_21: cantidad de munición que aumenta

3.2.13.5 Líquidos

Cada uno de los líquidos de las pociones de curación y velocidad tienen un comportamiento que se adapta al movimiento y a la orientación del frasco de la poción.

En las siguientes ilustraciones se pueden ver los comportamientos de ambas pociones en varias circunstancias:

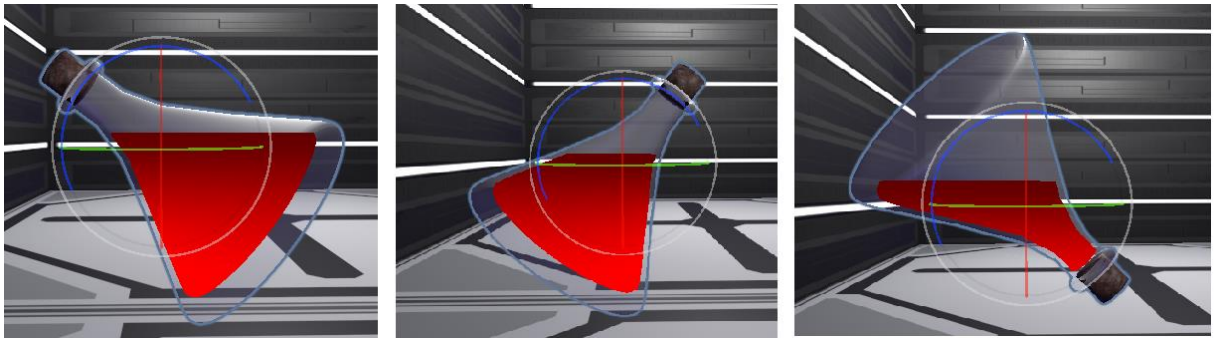


Ilustración 3.205; Casos de líquido de la poción de curación

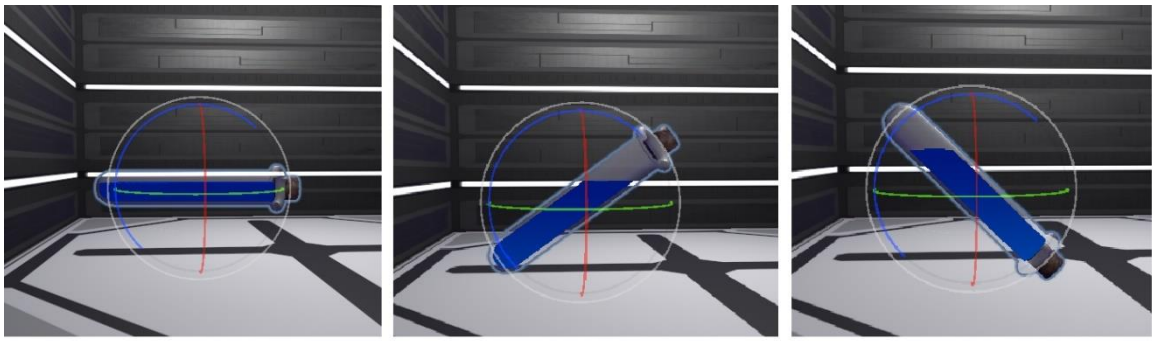


Ilustración 3.206; Casos de líquido de la poción de velocidad

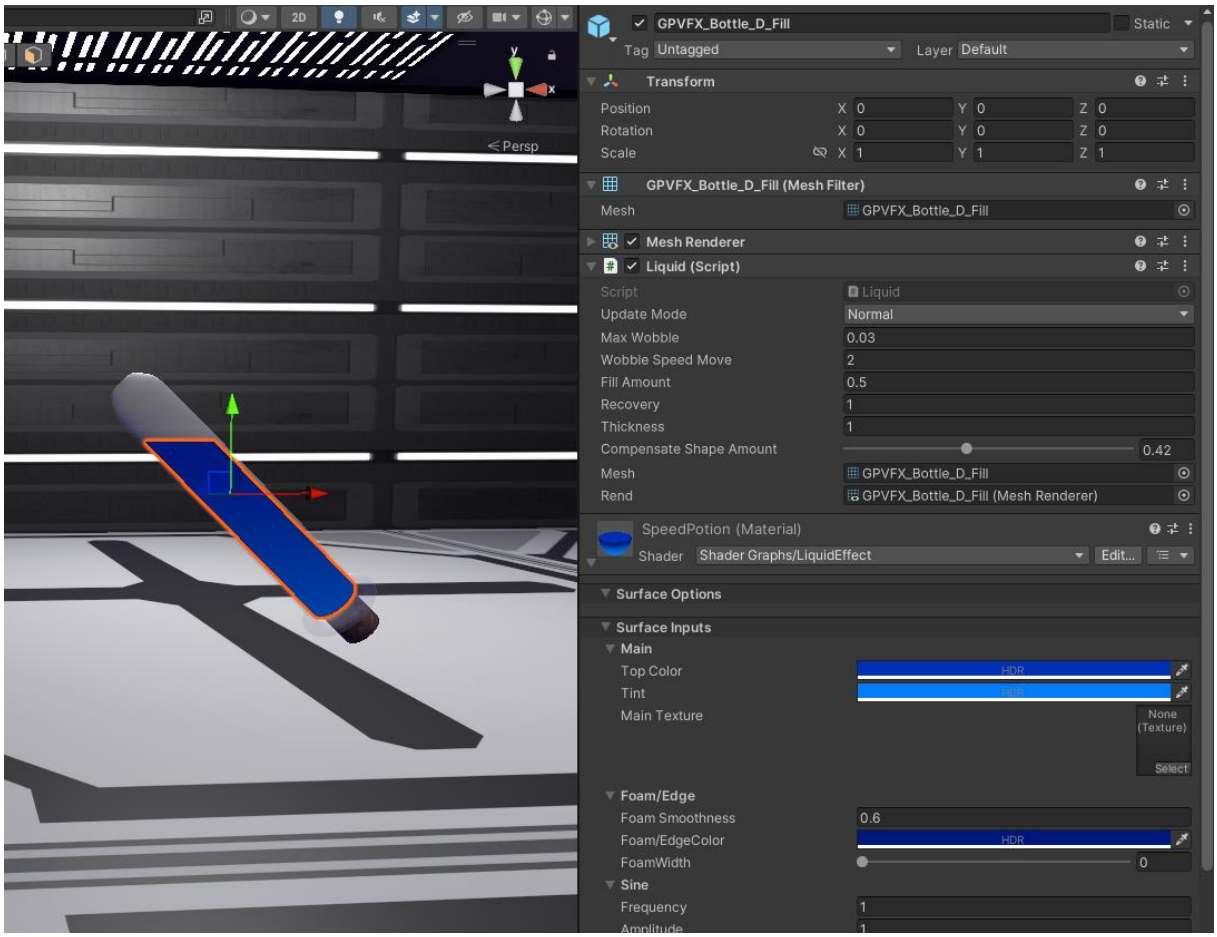


Ilustración 3.207; Componentes del líquido de la poción de velocidad

Para conseguir este efecto, lo que se ha hecho ha sido desarrollar un shader, el cual controla este efecto.

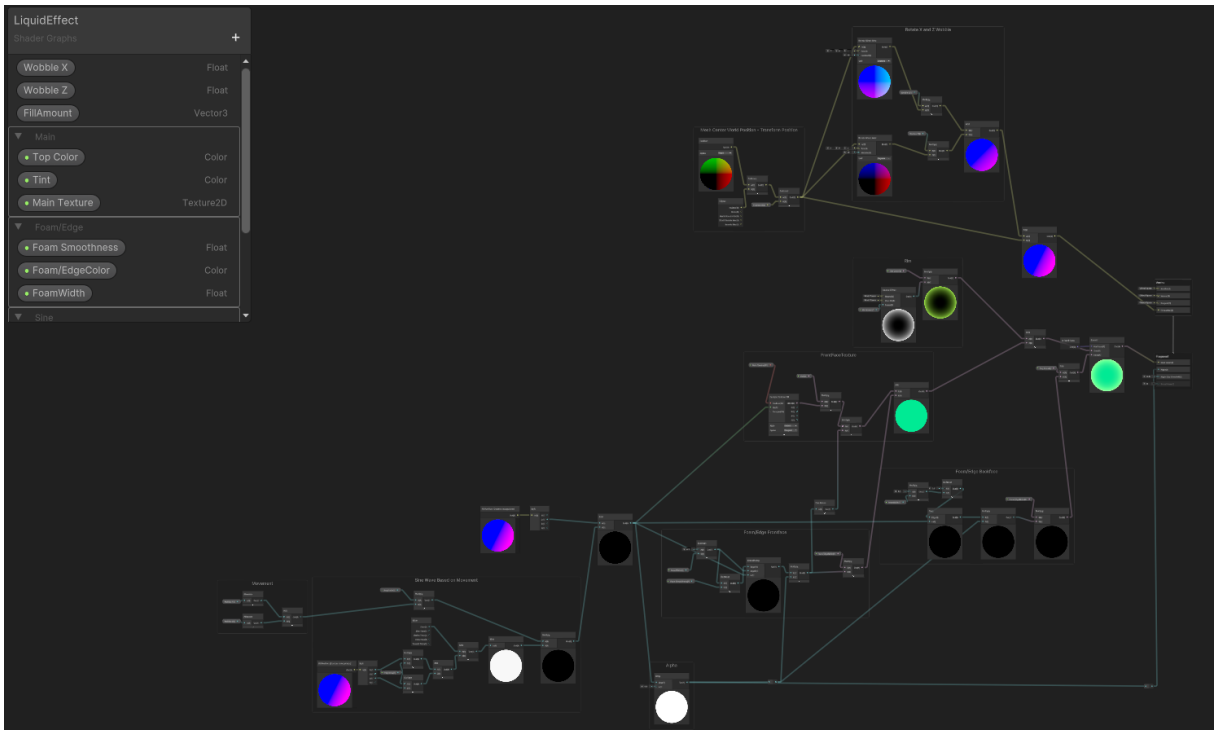


Ilustración 3.208; ShaderGraph del líquido de las pociones

Como se puede ver, es un shader bastante complejo, por lo que se va a desglosar para poder explicarlo de forma más clara.

En la primera parte del shader, se puede observar que usamos dos parámetros: “Wobble X” y “Wobble Z. Estos representan el movimiento de oscilación de la poción en los ejes X y Z. Para evitar errores, estos valores se meten en el nodo “Absolute” para convertir sus valores en valores absolutos y así evitar problemas con valores negativos. Una vez asegurado que se trabaja con valores positivos, se suman ambos valores con el nodo “Add”.

Después, se usa el nodo “Multiply” para multiplicar la suma por otro parámetro llamado “Amplitud”, con el cual se ajusta la magnitud del efecto de movimiento.

Ahora lo que se hace es hace es sumar la suma de la multiplicación del parámetro “Frequency”, el cual indica la frecuencia de la onda senoidal del líquido, con los canales rojo y verde del líquido. Esta suma se suma a su vez a una variable de tiempo para ajustar la frecuencia de la onda senoidal. El resultado se pasa al nodo “Sine”, con el que se consigue una onda sinusoidal en base al resultado obtenido

antes. Esta onda generada oscila entre -1 y 1 en función del tiempo y del movimiento del líquido.

El resultado de la suma se multiplica por una constante de Amplitud. Esto ajusta la magnitud del efecto del movimiento. El resultado de esta multiplicación se multiplica también por el seno del resultado de la suma del tiempo con la suma de la multiplicación del parámetro “Frequency”, el cual representa la frecuencia de la onda sensorial.

Por último, la onda generada por el nodo “Sine” se multiplica por el valor del movimiento calculado antes. Esto crea una onda senoidal, la cual se ajusta de forma dinámica con el movimiento del líquido.

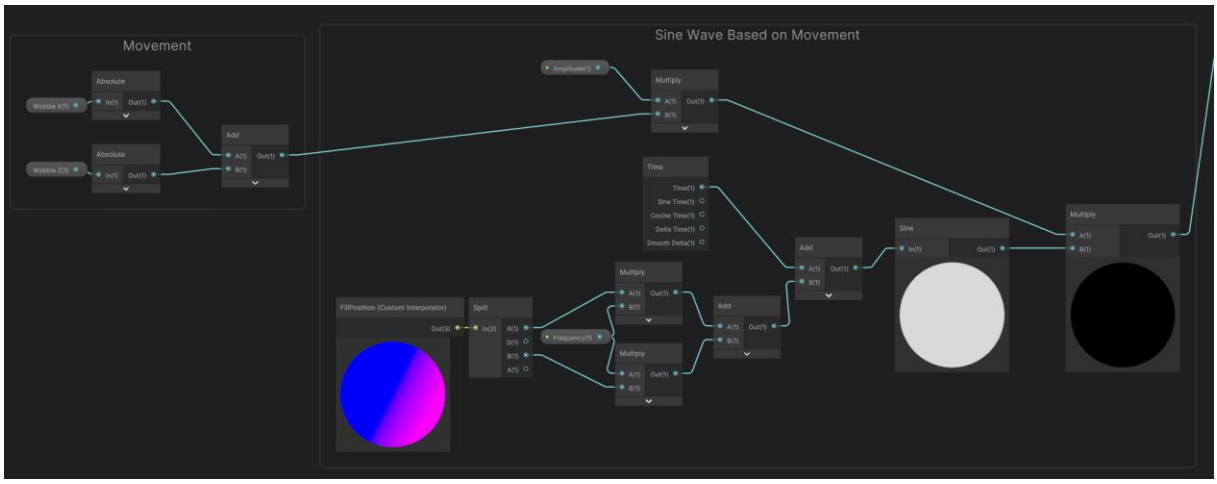


Ilustración 3.209; ShaderGraph del líquido de las pociones, primera parte

Una vez conseguido lo anterior, creamos y ajustamos la apariencia de la espuma en los bordes del líquido. Además, se ajusta su transparencia para aumentar el realismo del líquido.

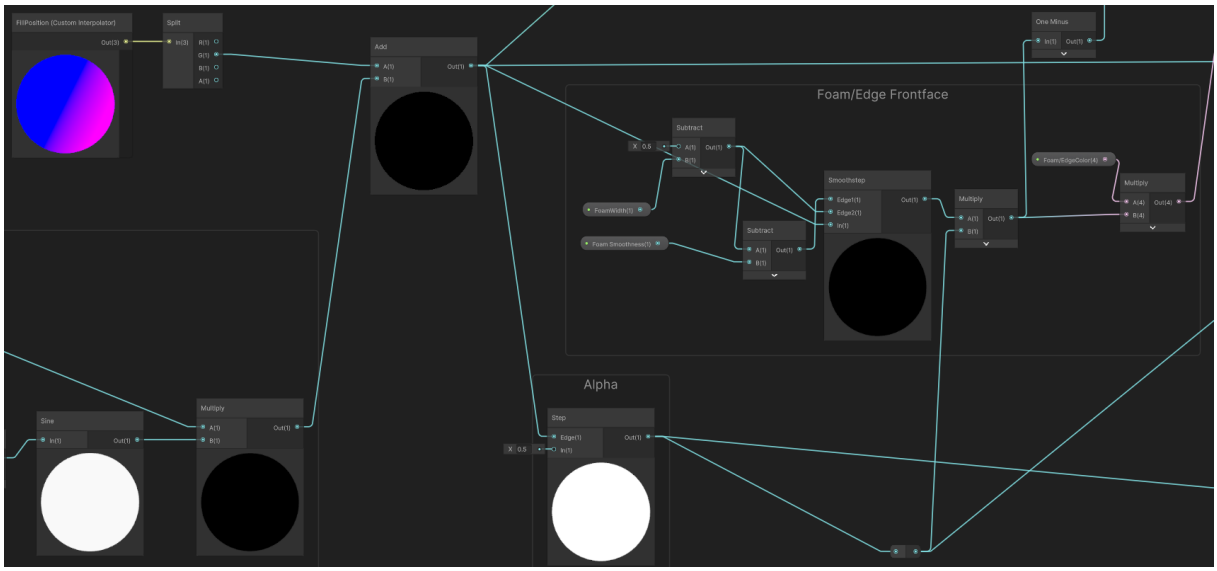


Ilustración 3.210; ShaderGraph del líquido de las pociones, segunda parte

Ahora, se muestrea la textura principal pasada como parámetro con la variable “MainTexture”. Además, se ajusta el color de la misma. La textura resultante se combina con la cara frontal de la espuma para obtener el color final de la textura frontal.

Por otro lado, en la parte trasera del líquido se ajusta el ancho y la suavidad de la espuma, al igual que el color de la espuma, que también se ajusta en esta parte

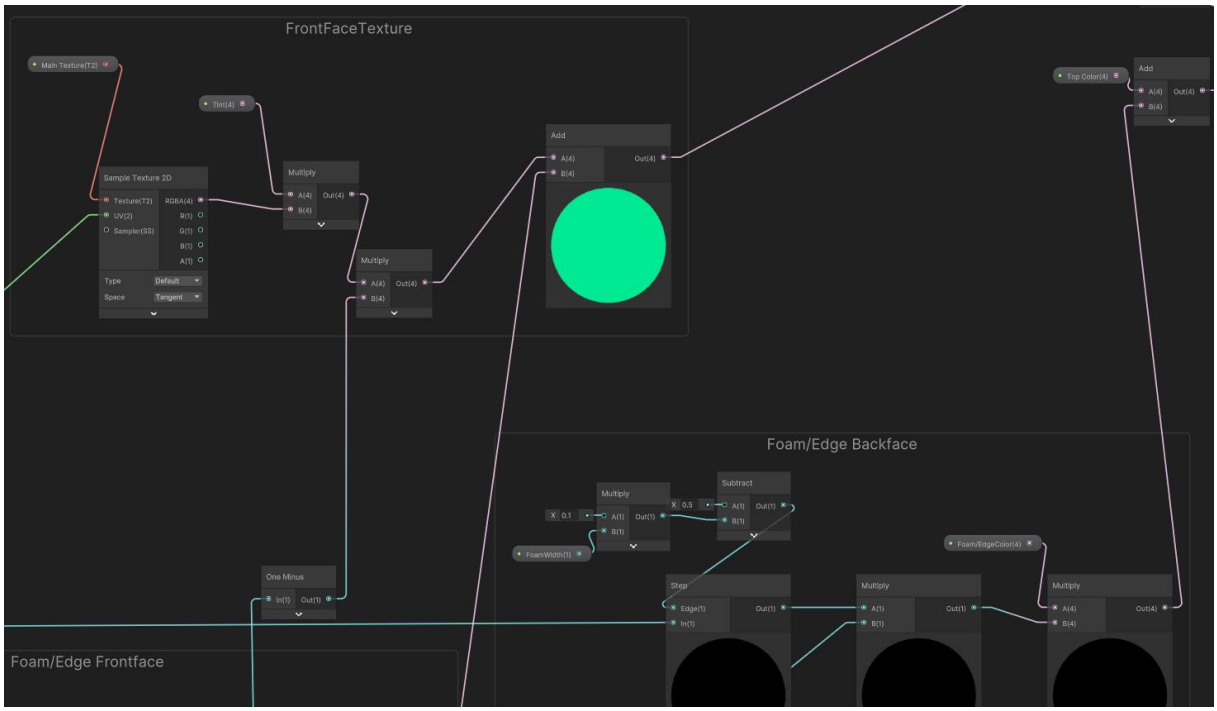


Ilustración 3.211; ShaderGraph del líquido de las pociones, tercera parte

Ahora se trata la malla del líquido. El centro de la misma se ajusta, proporcionando una posicion relativa en el espacio del mundo, con lo que se conseguirá que el líquido se mueva siempre siguiendo el centro de gravedad, el cual mira hacia el suelo.

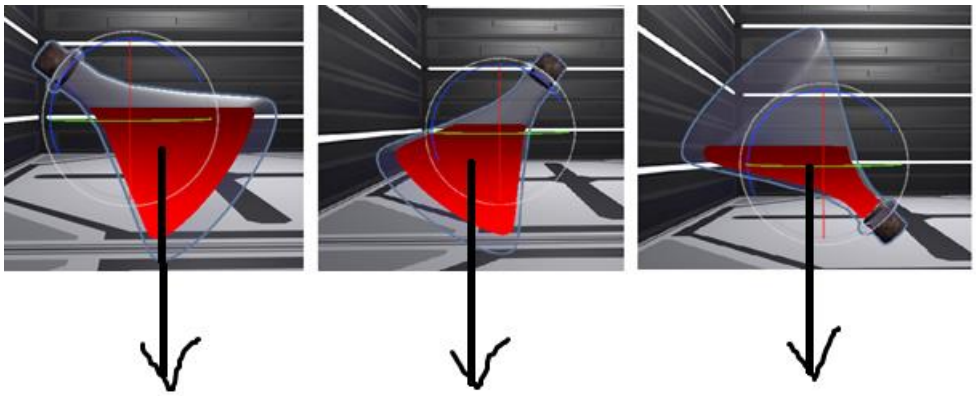


Ilustración 3.212; Gravedad en diferentes casos del líquido de la poción de curación

Para ello, se realizan rotaciones sobre los ejes X y Z utilizando los valores de las variables “WobbleX” y “WobbleZ”, los cuales se multiplican por los valores de rotación para ajustar la intensidad de la oscilación del líquido. Luego, estos resultados se suman para obtener el efecto final de oscilación en los ejes X y Z.

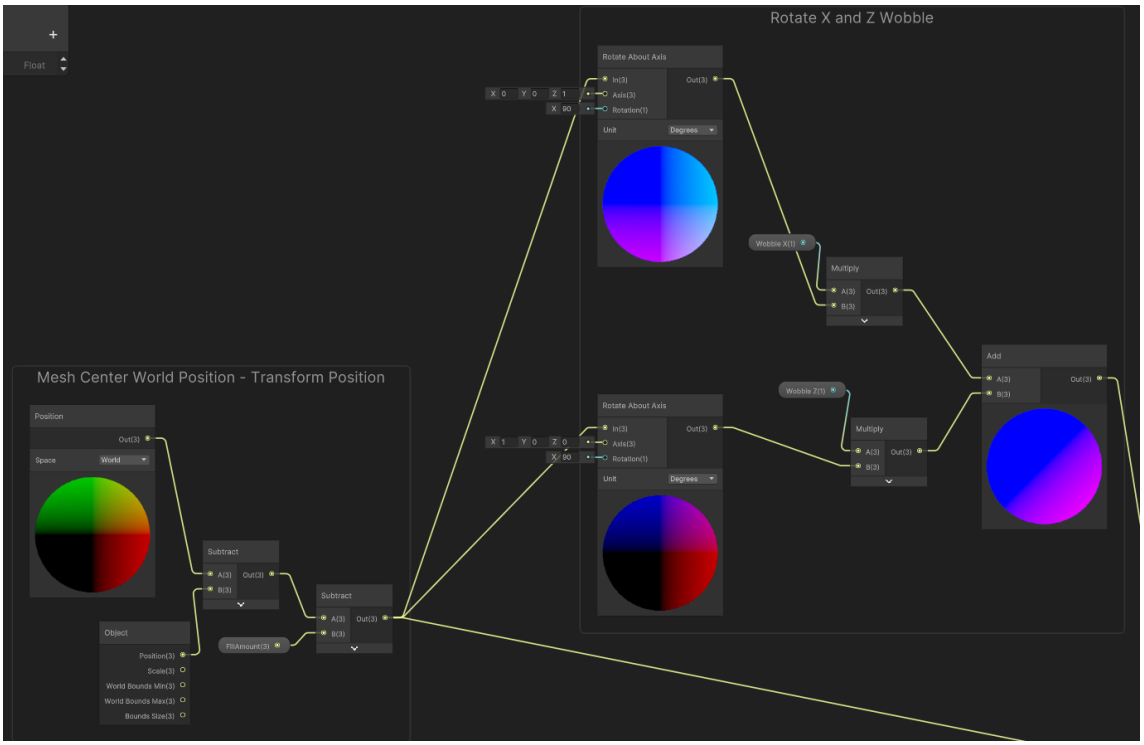


Ilustración 3.213; ShaderGraph del líquido de las pociones, cuarta parte

Por último, se calcula la intensidad del color de la espuma en base a la normal del líquido y de la direccion de la cámara, para que se pueda apreciar que esta siempre se ubica en la parte de arriba del líquido, independientemente de la orientación de este.

Esta intensidad calculada se suma con otros efectos, como el color del propio líquido, para crear el color final del objeto.

Para finalizar, los vértices, normales y tangentes del líquido se procesan de manera que se ajusten a la orientación y la vista del líquido desde la cámara.

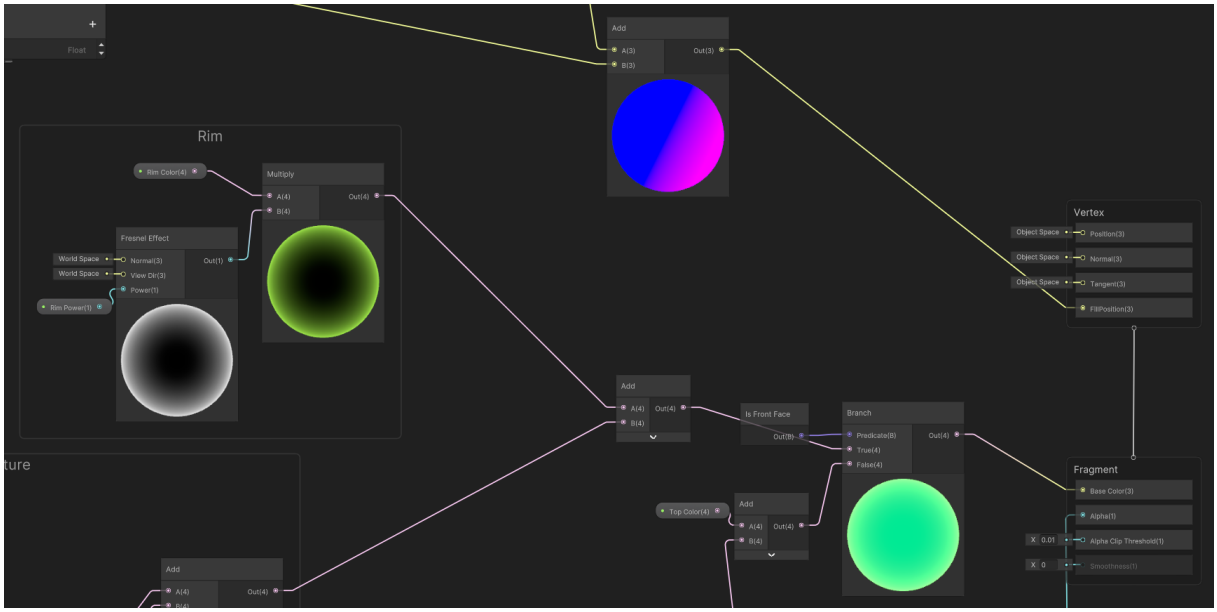


Ilustración 3.214; ShaderGraph del líquido de las pociones, quinta parte

Una vez explicado el shader desarrollado, este shader se asocia a un material y se configuran los parámetros del shader.

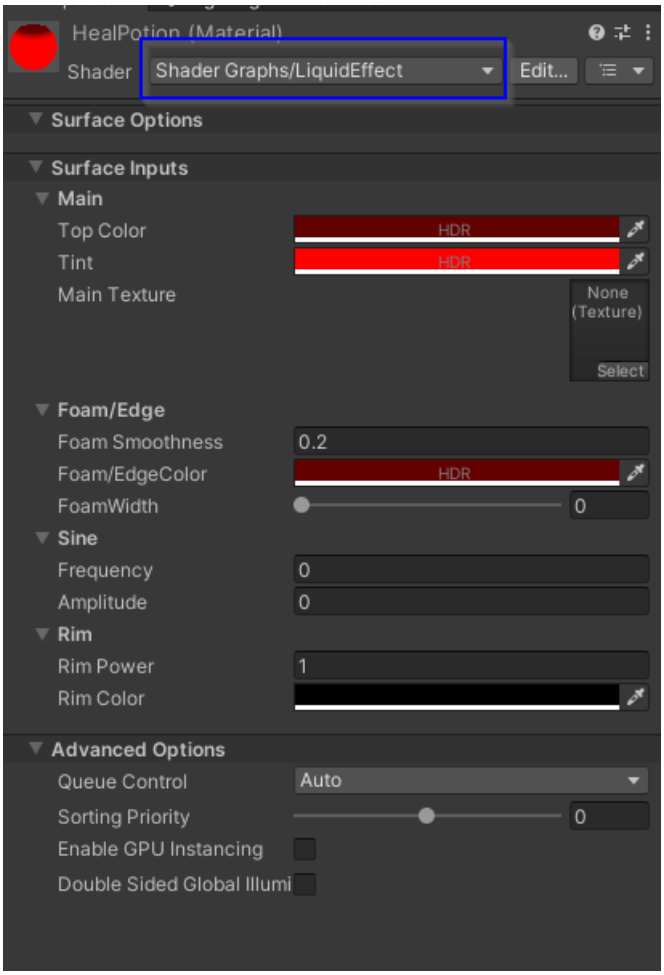


Ilustración 3.215; Componentes del líquido de la poción de curación

Para controlar este material, a los líquidos de las pociones se les asigna un script llamado “Liquid”, con el cual se maneja el movimiento y animación del líquido independientemente de la situación en la que se encuentre. Para ello, el script calcula constantemente la oscilación y el nivel de llenado en tiempo real y actualiza el shader “LiquidEffect” con estos valores, para que el líquido se vea como se tiene que ver en las circunstancias en las que esté.

3.70.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_13_22	
Nombre de la prueba o test	Movimiento del líquido

Objetivo	El líquido de las pociones debe moverse atendiendo a la fuerza de gravedad
Resultado	El líquido de las pociones se mueve dentro del frasco de la poción en función del movimiento del jugador si la poción está en su mano, y si la poción está en el suelo, su líquido se mueve si el jugador golpea la poción.
Comentarios	Si el movimiento es muy rápido, el líquido hace cosas raras.

Tabla 3.194; Test_13_22: movimiento del líquido

3.2.13.6 Cambio entre ítems y el arma

El jugador podrá elegir la poción que desea usar y el arma gracias al método "UpdateItemShowed()" del "GameManager" y el script llamado "PotionSwitching".

Este script, el cual está asignado al elemento "PotionPosition", hijo de la cámara principal como se ha comentado previamente en el **apartado 3.1.4.6.2.8 PotionPosition**, se encarga de visualizar la opción elegida, que no es otra cosa que activar la poción elegida y desactivar las otras, en función de las teclas que el jugador pulse:

1. Si el jugador pulsa la tecla 1: se activa poción de curación
2. Si el jugador pulsa la tecla 2: se activa poción de ácido
3. Si el jugador pulsa la tecla 3: se activa poción de velocidad
4. Si el jugador pulsa la tecla 4: se activa batería

Para ello, en función de la tecla que el jugador haya pulsado, se actualiza la variable "selectedPotion", la cual indica la poción que se selecciona. Si se ha seleccionado una poción que no estaba seleccionada antes, se llama al método "SelectPotion()" del mismo script, el cual revisa todas las pociones y desactiva la que no esté seleccionada y activa la que esté seleccionada, usando la variable "selectedPotion" como referencia.

Al activar o desactivar una poción, también se activa o desactiva el script “Liquid” asociado al líquido de la poción. Esto se hace para que no haya conflictos con el shader del líquido.

Se aporta el pseudocódigo del script:

```
Update()
  previousSelectedPotion <- selectedPotion

  Si se presiona la tecla "1"
    selectedPotion <- 0
    GameManager.UpdateItemShowed(ItemShowed.Potions)

  Si se presiona la tecla "2"
    selectedPotion <- 1
    GameManager.UpdateItemShowed(ItemShowed.Potions)

  Si se presiona la tecla "3"
    selectedPotion <- 2
    GameManager.UpdateItemShowed(ItemShowed.Potions)

  Si se presiona la tecla "4"
    selectedPotion <- 3
    GameManager.UpdateItemShowed(ItemShowed.Potions)

  Si previousSelectedPotion no es igual a selectedPotion
    GameManager.UpdateItemShowed(ItemShowed.Potions)
    SelectPotion(verdadero)

SelectPotion(selected)
  i <- 0
  Para cada potion
    fillObject <- se busca el elemento de relleno de la poción
    fillScript <- se obtiene el componente Liquid de fillObject

    Si i es igual a selectedPotion
      Si selected es verdadero y CheckPotionAmount(potion.name) es verdadero
        Se activa potion
        Se activa fillScript

      Si no
        Se desactiva potion
        Se desactiva fillScript

    Si no
      Se desactiva potion
      Se desactiva fillScript

    Se incrementa i

CheckPotionAmount(potionName)
  Si potionName es "HealPotion"
    Devuelve si getHealPotionAmount() > 0 en GameManager

  Si potionName es "SpeedPotion"
    Devuelve si getSpeedPotionAmount() > 0 en GameManager

  Si potionName es "AcidPotion"
    Devuelve si getAcidPotionAmount() > 0 en GameManager

  Si potionName es "Battery"
    Devuelve si getBatteriesAmount() > 0 en GameManager

  Si no es ninguno, se devuelve falso

checkIfPotionIsActive(potionName)
  Para cada potion
    Si nombre de la pocion es igual a potionName
      Se devuelve si pocion está activa en la escena o no

  Si no, devuelve falso
```

Ilustración 3.216; Pseudocódigo del script PotionSwitching

3.72.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_13_23	
Nombre de la prueba o test	Visualización de las pociones
Objetivo	Las pociones elegidas se deben poder ver
Resultado	La poción elegida se visualiza perfectamente en la cámara del jugador
Comentarios	Todo OK.

Tabla 3.195; Test_13_23: visualización de las pociones

Test_13_24	
Nombre de la prueba o test	Visualización del arma
Objetivo	El arma del jugador se debe poder ver
Resultado	El arma del jugador se visualiza perfectamente en su cámara
Comentarios	Todo OK.

Tabla 3.196; Test_13_24: visualización del arma

Test_13_25	
Nombre de la prueba o test	Cambio entre arma y pociones
Objetivo	El jugador debe poder cambiar entre arma y pociones sin problema alguno
Resultado	El jugador puede cambiar entre arma y pociones haciendo uso del dispositivos de entrada
Comentarios	El jugador puede no saber cómo cambiar entre arma y pociones

Tabla 3.197; Test_13_25: cambio entre arma y pociones

Test_13_26	
Nombre de la prueba o test	No visualizar pociones
Objetivo	La poción elegida no se deberá poder visualizar si no hay suficiente cantidad de esa poción
Resultado	El jugador no puede cambiar a la poción elegida si no tiene cantidades suficientes de esta poción
Comentarios	El jugador puede no saber cuántas pociones de ese tipo tiene y puede que no sepa por qué no se visualiza la poción elegida por más que pulse la tecla de cambiar a la poción.

Tabla 3.198; Test_13_26: no visualizar pociones

Test_13_27	
Nombre de la prueba o test	Solo visualizar poción elegida
Objetivo	Solo se debe poder ver la poción elegida, ninguna otra
Resultado	El jugador, al cambiar a una poción, si tiene cantidades suficientes de esta poción, aparece en su mano únicamente la poción elegida, no aparecen varias pociones en su mano.
Comentarios	Todo OK.

Tabla 3.199; Test_13_27: solo visualizar poción elegida

3.2.14 Aparición aleatoria de ítems

El jugador, como se ha comentado previamente, se puede encontrar ítems repartidos por el mundo durante su partida, concretamente, en toda la segunda planta, planta de principal actividad de combate del juego. No obstante, los ítems no están colocados de forma manual en el juego, si no que la ubicación e incluso la cantidad de estos varía en cada ejecución del juego o “respawn” del jugador. Para ello, lo que se ha hecho ha sido colocar puntos de aparición en varios puntos de la segunda planta, como podemos ver en la siguiente imagen (los cubos negros simplemente son para que se puedan apreciar las posiciones, no estarán en el juego final):

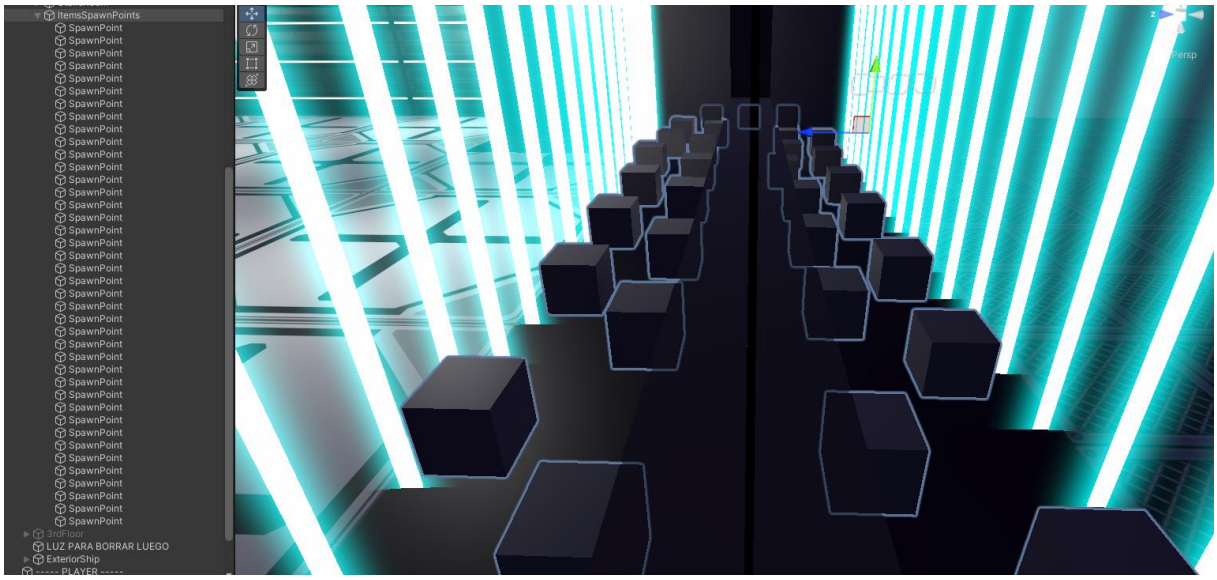


Ilustración 3.217; Posibles ubicaciones de reaparición de items en la segunda planta

Junto a todas estas posiciones, se ha incluido un script llamado “SpawnItems”, el cual, en función de la dificultad del juego seleccionada, hace aparecer un “ItemCollectionData” u otro.

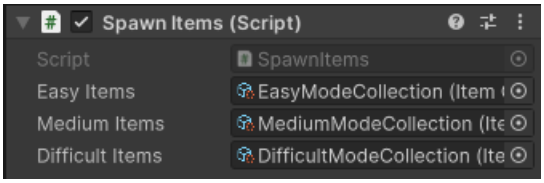


Ilustración 3.218; Componentes del script SpawnItems

Un “ItemCollectionData” no es más que un objeto “ScriptableObject” [102] personalizado. Un “ScriptableObject” es una clase especial en Unity que se puede usar para almacenar grandes cantidades de datos de una manera organizada y eficiente. A diferencia de los “MonoBehaviour”, los ScriptableObjects no se aplican a ningún “GameObject” de Unity, ni tampoco participan directamente en la lógica del juego en tiempo de ejecución.

Este “ItemCollectionData” tiene una EEDD compuesta por dos estructuras de datos estáticas. La primera de ella, llamada “requiredItems”, almacena los ítems indispensables de instanciar en el mundo. La segunda estructura de datos estática, llamada “randomItems”, almacena los ítems que pueden o no aparecer, es decir, almacena los ítems no relevantes o dispensables.

```
[CreateAssetMenu(fileName = "ItemCollectionData", menuName = "ScriptableObjects/ItemCollectionData", order = 3)]
4 referencias
public class ItemCollectionData : ScriptableObject
{
    2 referencias
    public GameObject[] requiredItems;
    3 referencias
    public GameObject[] randomItems;
}
```

Ilustración 3.219; Estructura de la clase ItemCollecionData

En base a estos “ScriptableObjects”, se han creado tres objetos, uno para cada dificultad del juego.

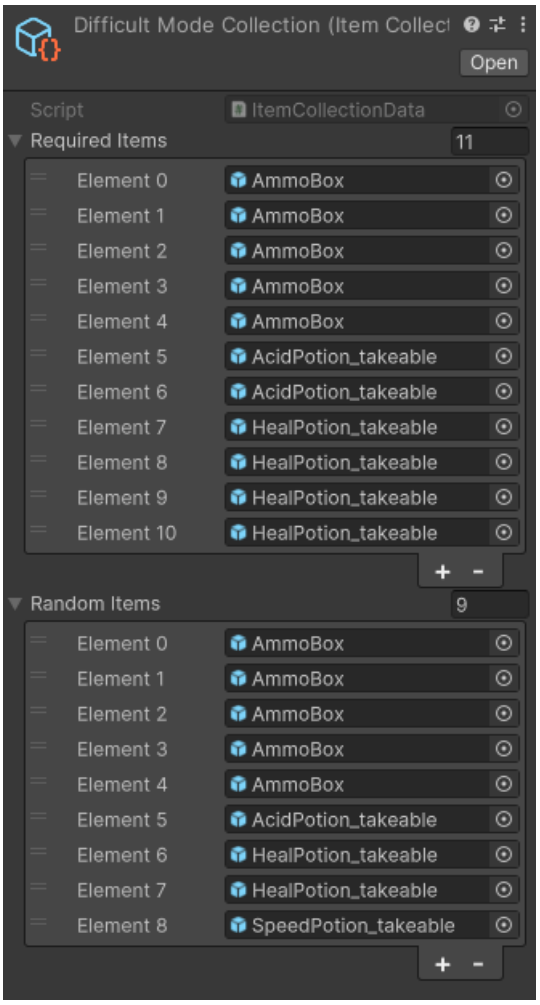


Ilustración 3.220; Ejemplo de instancia de ScriptableObject

Como se ha comentado previamente, el script “SpawnItem” elige uno de los “ItemCollectionData” que recibe por parámetro en base a la dificultad del juego. Después, coge todas las posiciones de aparición de ítems y las mezcla, de forma que siempre sean distintas en cada ejecución. Una vez mezcladas las posiciones de

aparición de ítems, el script instancia cada uno de los ítems imprescindibles en una de esas posiciones de forma aleatoria, sin repetir. Cuando instancien los ítems indispensables, se procede a instanciar los ítems prescindibles. Para aportar más aleatoriedad a esta funcionalidad, lo que se hace es que se le da a cada ítem una probabilidad del 80% (4/5) de aparecer.

Cabe mencionar que, independientemente de la dificultad del juego, se considera indispensable que haya: cinco cajas de munición, cuatro pociones de curación y dos pociones de ácido. Por este motivo, todas las colecciones de ítems van a tener esos ítems en la estructura de datos de “requiredItems”.

3.74.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_14_01	
Nombre de la prueba o test	Aparición de ítems
Objetivo	Los ítems deben aparecer
Resultado	Aparecen ítems repartidos en la segunda planta
Comentarios	Todo OK.

Tabla 3.200; Test_14_01: aparición de ítems

Test_14_02	
Nombre de la prueba o test	Aparición aleatoria de ítems
Objetivo	Los ítems deben aparecer, de forma aleatoria, en una de las posiciones definidas previamente
Resultado	La posición de los ítems es totalmente aleatoria
Comentarios	El jugador puede no saber que los ítems aparecen de forma aleatoria y la próxima vez que el jugador vaya a por un ítem concreto, sabiendo que ese ítem aparece en X posición, vaya y no esté ese ítem en esa posición concreta

Tabla 3.201; Test_14_02: aparición aleatoria de ítems

Test_14_03	
Nombre de la prueba o test	Ubicación de aparición de ítems
Objetivo	Los ítems deben aparecer en una de las ubicaciones posicionadas previamente
Resultado	Los ítems aparecen posicionados exactamente en una de las ubicaciones definidas previamente
Comentarios	Todo OK.

Tabla 3.202; Test_14_03: ubicación de aparición de ítems

Test_14_04	
Nombre de la prueba o test	Todos ítems indispensables aparecen
Objetivo	Todos los ítems indispensables deben aparecer
Resultado	Todos los ítems de la colección elegida aparecen en alguno de los lugares definidos
Comentarios	El jugador puede no saber cuáles son los ítems indispensables de aparecer.

Tabla 3.203; Test_14_04: todos los ítems indispensables aparecen

Test_14_05	
Nombre de la prueba o test	La colección de ítems depende la dificultad
Objetivo	En función de la dificultad elegida, se debe colocar una colección de ítems u otra.
Resultado	En función de la dificultad elegida, se coloca una colección de ítems diferente.
Comentarios	Todo OK.

Tabla 3.204; Test_14_05: la colección de ítems depende de la dificultad

Test_14_06	
Nombre de la prueba o test	Ítems aleatorios

Objetivo	Los ítems aleatorios deben tener una probabilidad de aparecer
Resultado	De los ítems aleatorios no todos aparecen, algunos si y otros no
Comentarios	El jugador puede no saber qué ítems van a aparecer y cuáles no.

Tabla 3.205; Test_14_06: ítems aleatorios

3.2.15 Iluminación

En el hueco del ascensor que llevará al jugador desde la primera planta hasta la segunda planta, se encontrarán unos bloques de luz

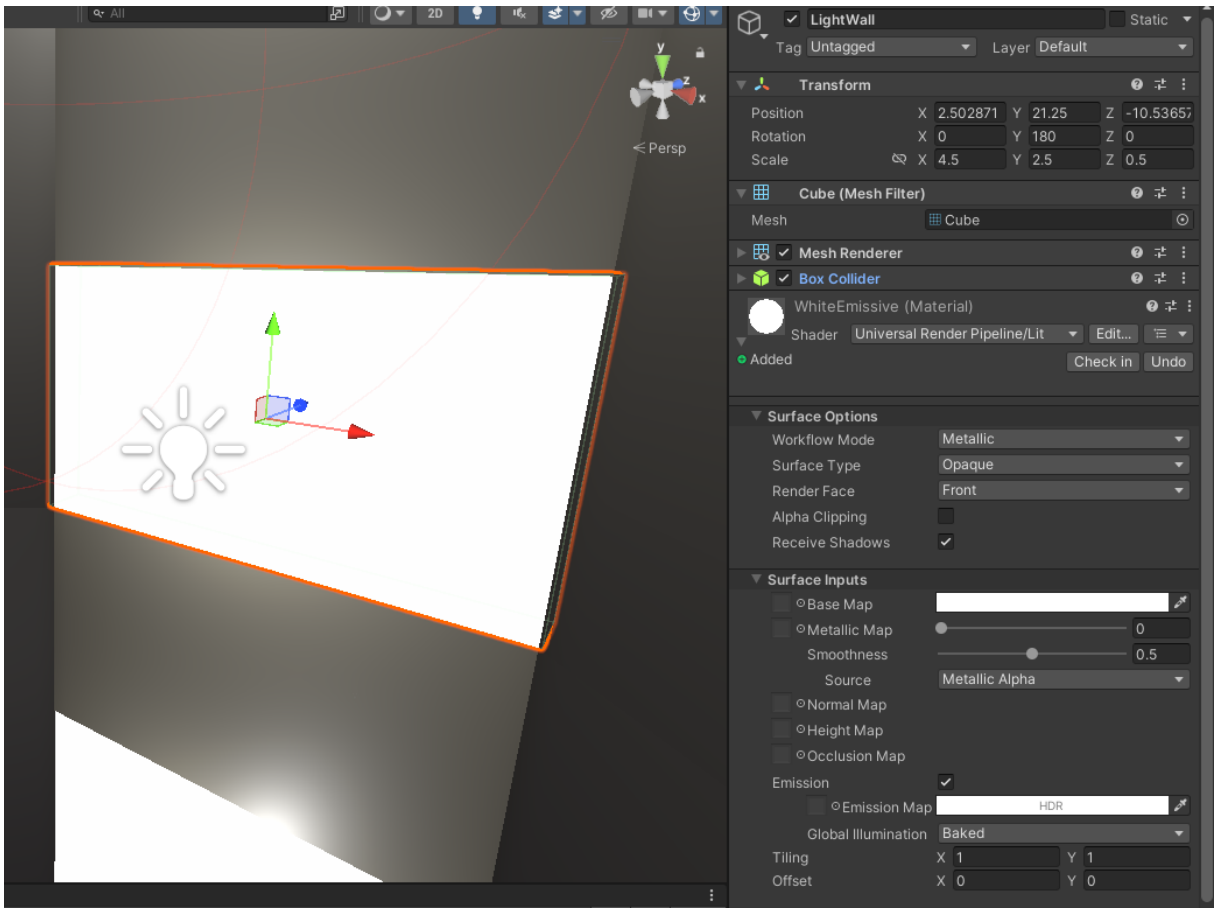


Ilustración 3.221; Componentes del bloque de luz del hueco del ascensor

Estos bloques de luz serán una composición de varios elementos. El primero, un objeto “Cube”, el cual tiene un material con un componente “Emissive” [103] de

color blanco para que dé la sensación de que aporta luz. El segundo, un objeto “Light” el cual es una luz puntual [104] que aportará luz al hueco del ascensor.

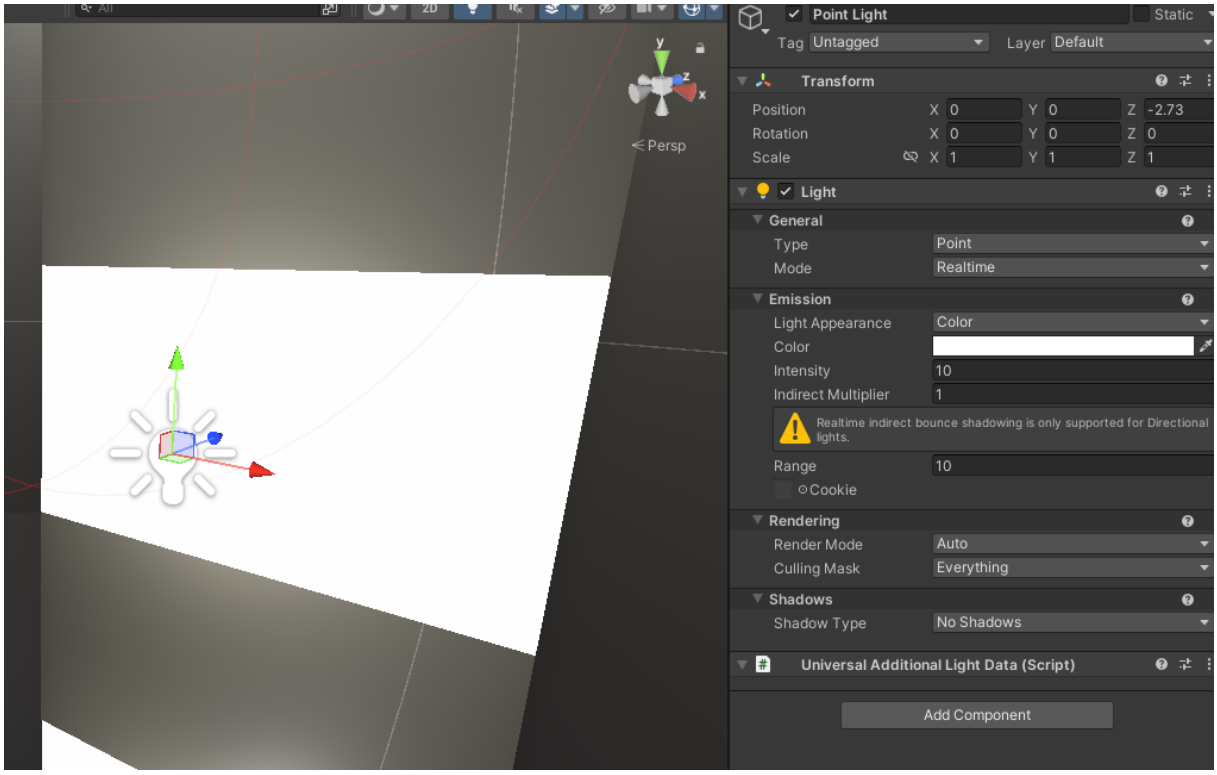


Ilustración 3.222; Componentes de la luz del bloque de luz del hueco del ascensor

También existirán los paneles de luz, los cuales se podrán encontrar como techo de algunas de las salas de la nave

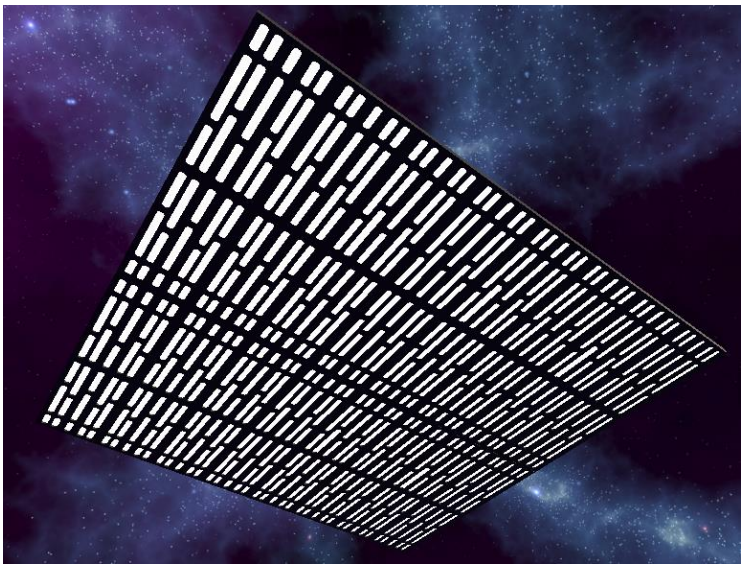


Ilustración 3.223; Panel de luz de las salas

Estos paneles estarán compuestos por ocho paneles individuales

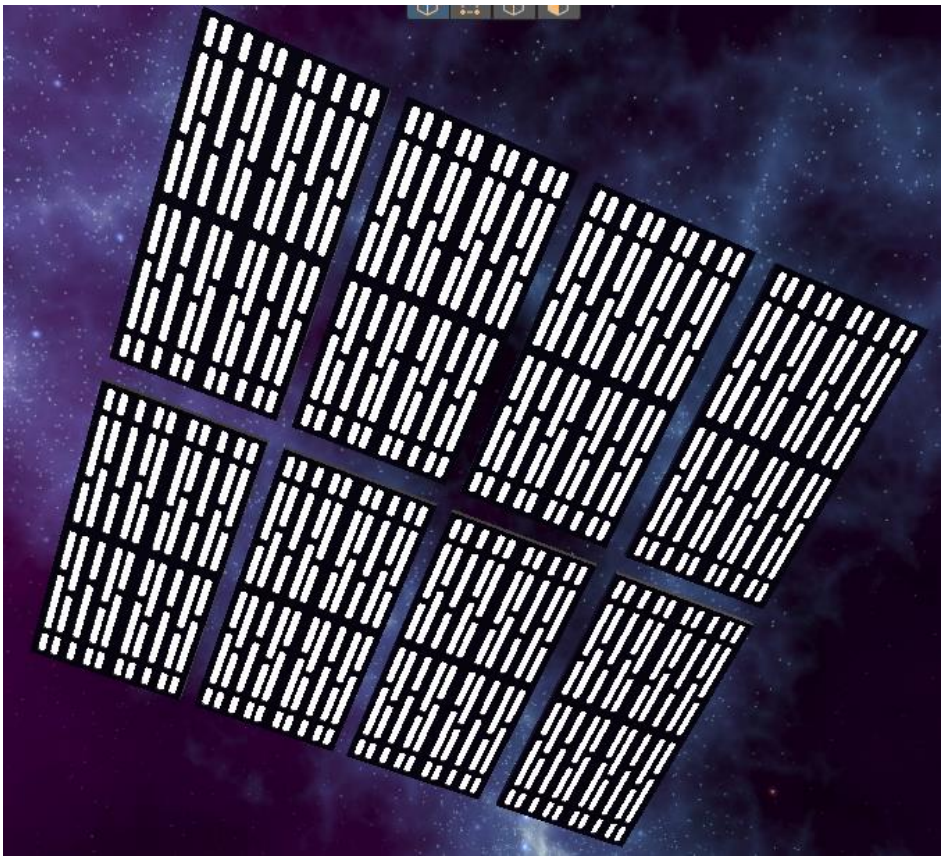


Ilustración 3.224; Panel de luz de las salas desglosado

Y por una luz focal [104] cuyos ángulos interno y externo son el mismo.

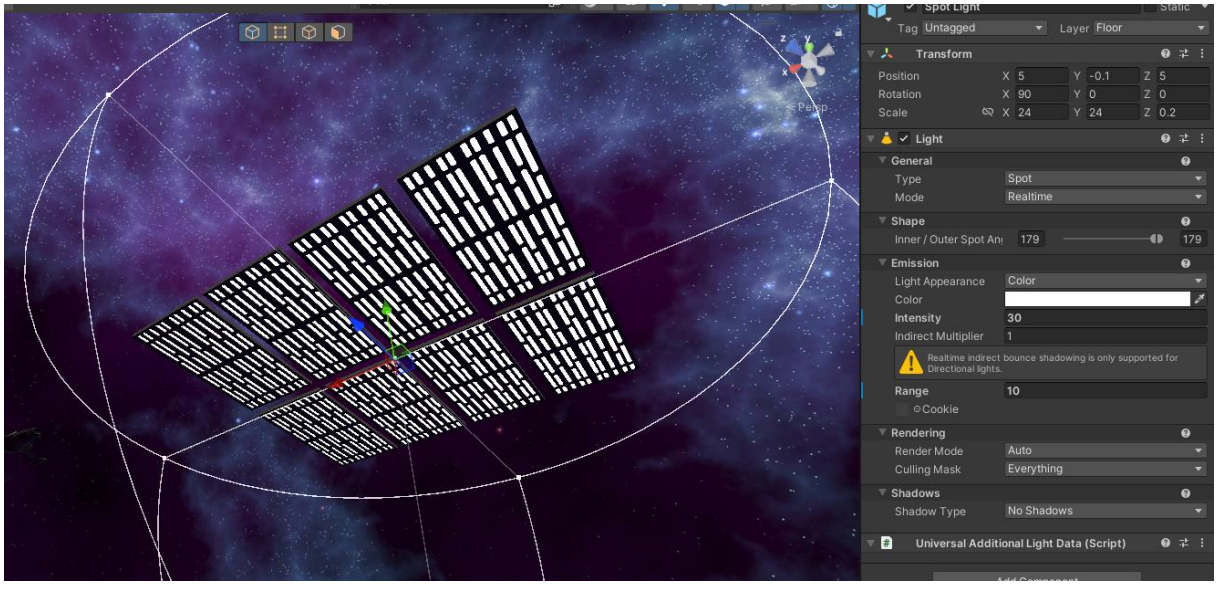


Ilustración 3.225; Componentes de la luz del panel de luz de las salas

En el juego, también se podrá encontrar una luz direccional. Esta se podrá encontrar en el Sol, el cual ha sido analizado en el **apartado 3.2.4.4.1.3 Sol**.

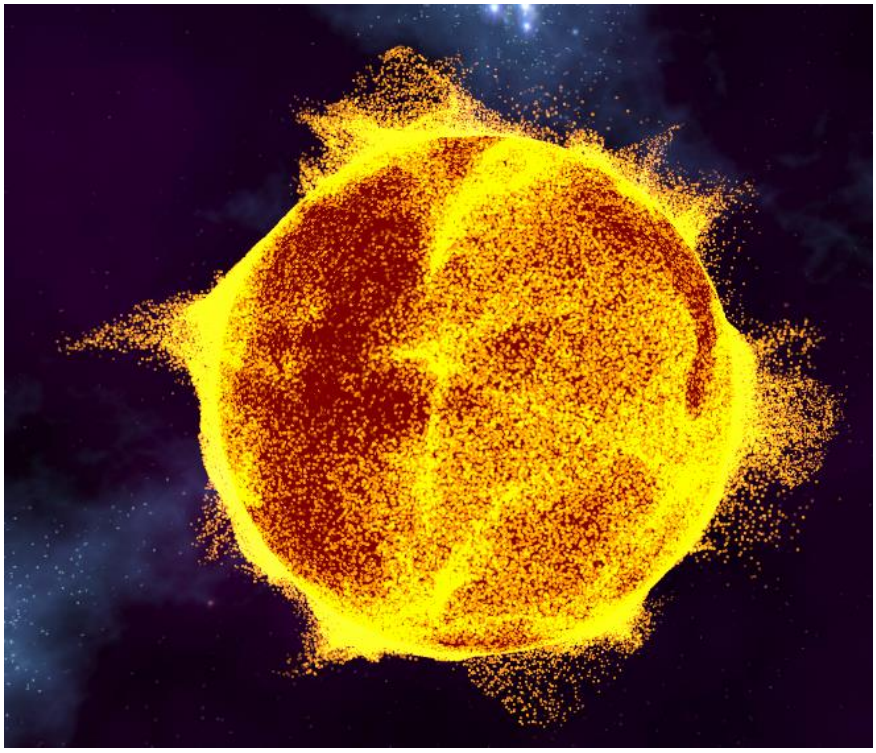


Ilustración 3.226; Sol

Esta fuente de luz direccional [104] “apuntará” al centro de la escena, la nave espacial y esta tendrá un color blanco con un poco de amarillo, simulando la luz del Sol.

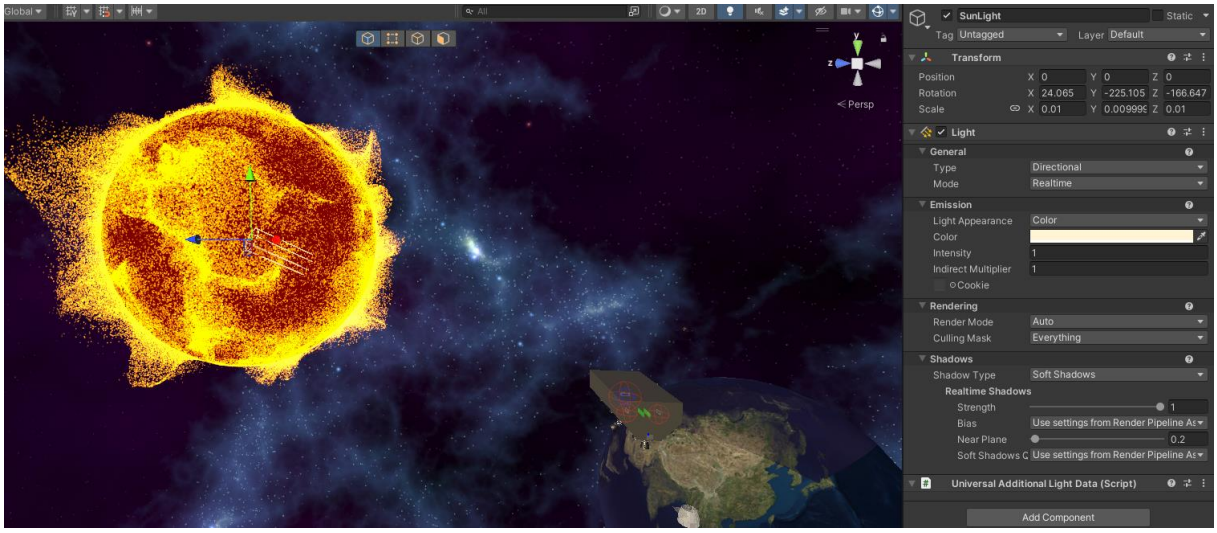


Ilustración 3.227; Componentes de la luz del Sol

3.76.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_15_01	
Nombre de la prueba o test	Iluminación adecuada
Objetivo	La iluminación debe dar luz a todas las partes de la zona en las que esté
Resultado	El jugador puede ver sin dificultad todas las partes de la zona en la que se encuentre
Comentarios	Todo OK.

Tabla 3.206; Test_15_01: iluminación adecuada

Test_15_02	
Nombre de la prueba o test	Intensidad de luz adecuada
Objetivo	La intensidad de luz debe ser adecuada
Resultado	No se aprecian fallos de iluminación o excesos de intensidad en las zonas alumbradas por la iluminación
Comentarios	Todo OK.

Tabla 3.207; Test_15_02: intensidad de luz adecuada

3.2.16 Créditos

Esta es una escena a parte a la que el jugador podrá llegar al interactuar con el portal, el cual ha sido comentado en el **apartado 3.2.4.3.3.2 Hangar**. Una vez llegados a esta escena, se podrán encontrar varios elementos que ayudarán al objetivo principal de esta escena, a contar el final y la conclusión de esta historia, acompañados de unos agradecimientos finales.

Para contar la conclusión de esta historia, se ha optado por usar el típico estilo de texto de StarWars, el “CrawlText” [105].

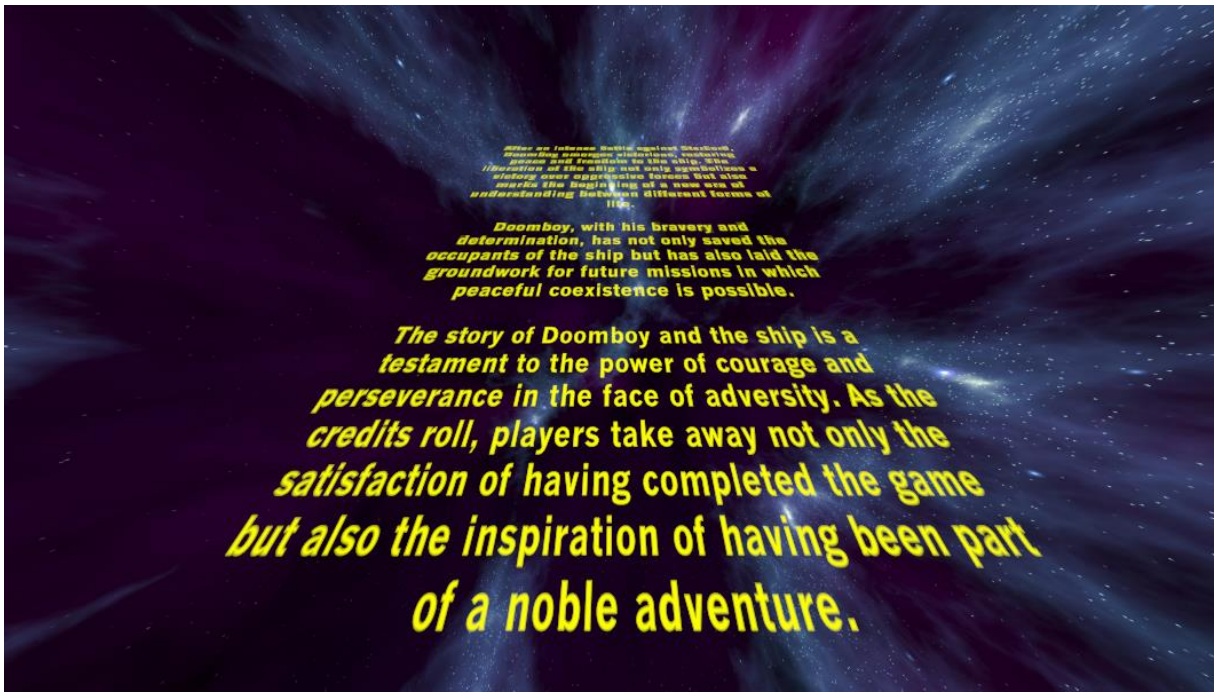


Ilustración 3.228; CrawlText de los créditos

Para conseguir este efecto, primeramente, se ha creado una cámara a la que se ha modificado el componente “FieldOfView”, “(FOV)” [106], de forma que se ha ampliado mucho ese componente para que los elementos que capten la cámara parezcan muy lejanos y pequeños, dando a la vez apertura y espacio a la visión de la cámara.

Ilustración 3.229; Componentes de la cámara de los créditos

Luego, se ha creado un componente “Canvas” al que se le ha incluido un componente de texto que almacenará el texto que se quiere mostrar. Este canvas,

está ligeramente inclinado para que el texto, junto con el efecto de la cámara, entre "grande" y salga "pequeño".

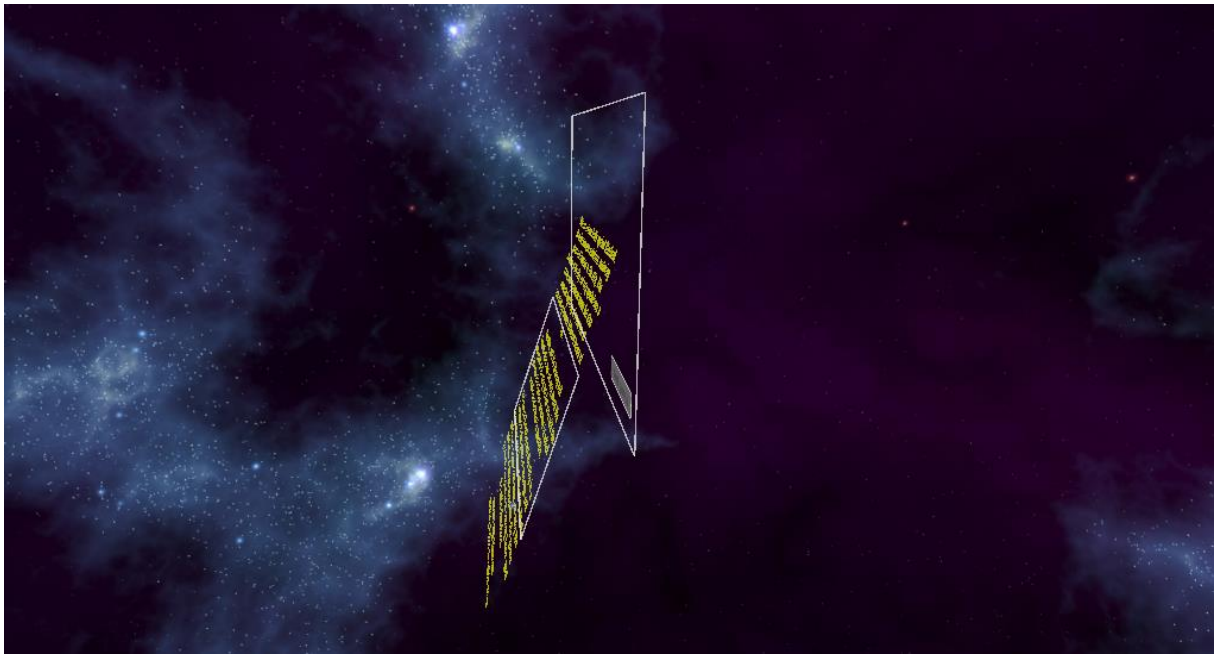


Ilustración 3.230; Vista de rotación del texto

Por último, para conseguir el efecto de "CrawlText", lo que se ha hecho es asignarle al canvas un script llamado "CrawlTextCanvasLogic". Este script lo que hace es coger el canvas y se desplaza para arriba a una velocidad determinada por parámetro.

Tambien comentar que, tras un determinado tiempo, aparecerá un botón el cual, si es pulsado por el jugador, llamará al método "ExitGame()" del "GameLogic" para salir de la aplicación. Para conseguir esta funcionalidad, se ha asignado al botón un script llamado "ButtonCanvasLogic", el cual se encargará de llamar a una corrutina llamada "ShowButton()" que realizará esta funcionalidad tras un tiempo determinado. Ese tiempo definido, será menor que el tiempo que dura todo el texto en aparecer. Esto se hace para que el jugador pueda cerrar el juego sin tener que pasar por todos los créditos.

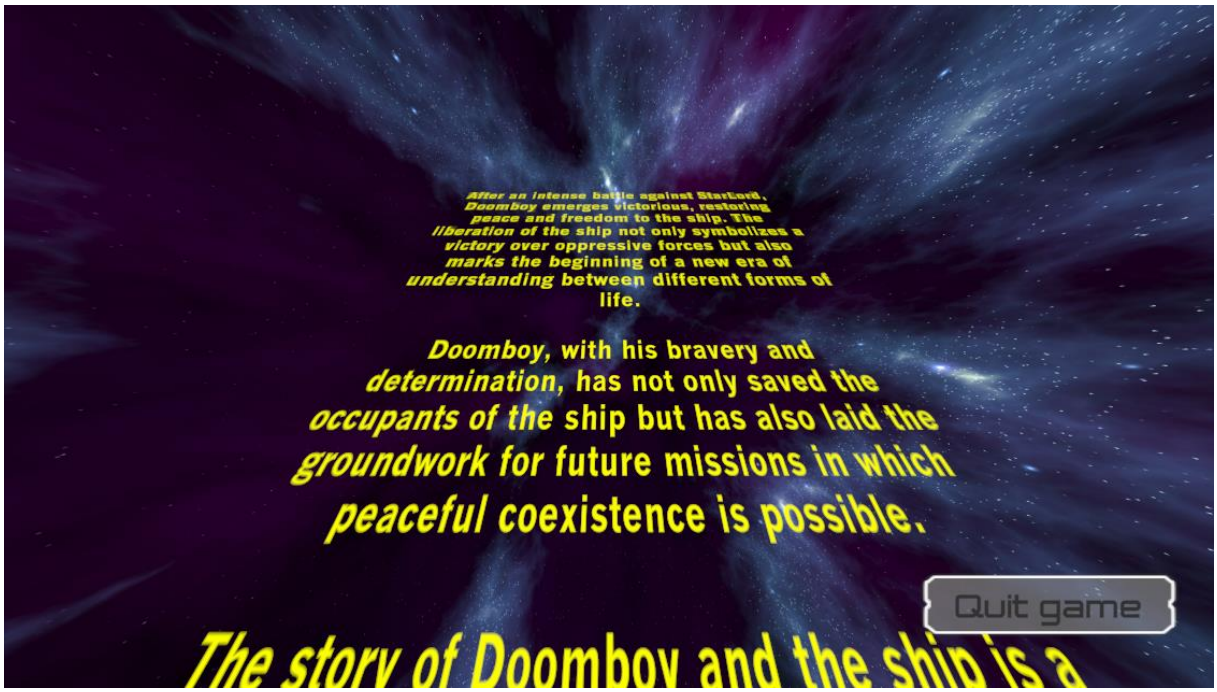


Ilustración 3.231; Botón de salir del juego

3.78.4.3.3.1.1 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el funcionamiento del arma del jugador.

Test_16_01	
Nombre de la prueba o test	Efecto CrawlText
Objetivo	El texto de los créditos debe tener el efecto de “CrawlText”.
Resultado	El texto de los créditos tiene el efecto de “CrawlText”
Comentarios	El jugador puede no leer bien el texto debido al efecto aplicado

Tabla 3.208; Test_16_01: efecto CrawlText

Test_16_02	
Nombre de la prueba o test	Distorsión del fondo
Objetivo	El fondo de la escena de los créditos debe estar distorsionado u hundido.

Resultado	El fondo de la escena de los créditos está hundido de forma que parece que el texto va hacia el centro de la pantalla
Comentarios	Todo OK.

Tabla 3.209; Test_16_02: distorsión del fondo

Test_16_03	
Nombre de la prueba o test	Aparición del botón
Objetivo	El botón de salir de la aplicación o programa debe aparecer tras cierto tiempo.
Resultado	Tras cierto tiempo viendo los créditos, aparece el botón para salir de la aplicación.
Comentarios	El jugador puede no prever que va a aparecer un botón con el que salir de la aplicación y puede llegar a desesperarse por querer salir de la aplicación, recurriendo a otros medios no convencionales.

Tabla 3.210; Test_16_03: aparición del botón

Test_16_04	
Nombre de la prueba o test	Utilidad del botón
Objetivo	El botón de los créditos debe funcionar
Resultado	Si el jugador pulsa el botón de los créditos, la aplicación se cierra
Comentarios	El jugador puede no saber qué hace el botón

Tabla 3.211; Test_16_04: utilidad del botón

Test_16_05	
Nombre de la prueba o test	No poder abrir menús
Objetivo	El jugador no deberá poder abrir menús en la escena de los créditos.
Resultado	El jugador, por más que pulse el botón de abrir los menús, no podrá abrir menús mientras esté en la escena de los créditos.

Comentarios	El jugador puede no saber que no puede abrir los menús en la escena de los créditos y puede llegar a pensar que está roto el sistema.

Tabla 3.212; Test_16_04: no poder abrir menús

3.3 Herramientas usadas

3.3.1 ProBuilder

"ProBuilder" [107] es una herramienta de Unity que permite a los desarrolladores crear y modificar geometría de los elementos de la escena directamente en el editor de Unity. Para poder realizar estas funcionalidades, "ProBuilder" aporta todas estas funciones y herramientas:

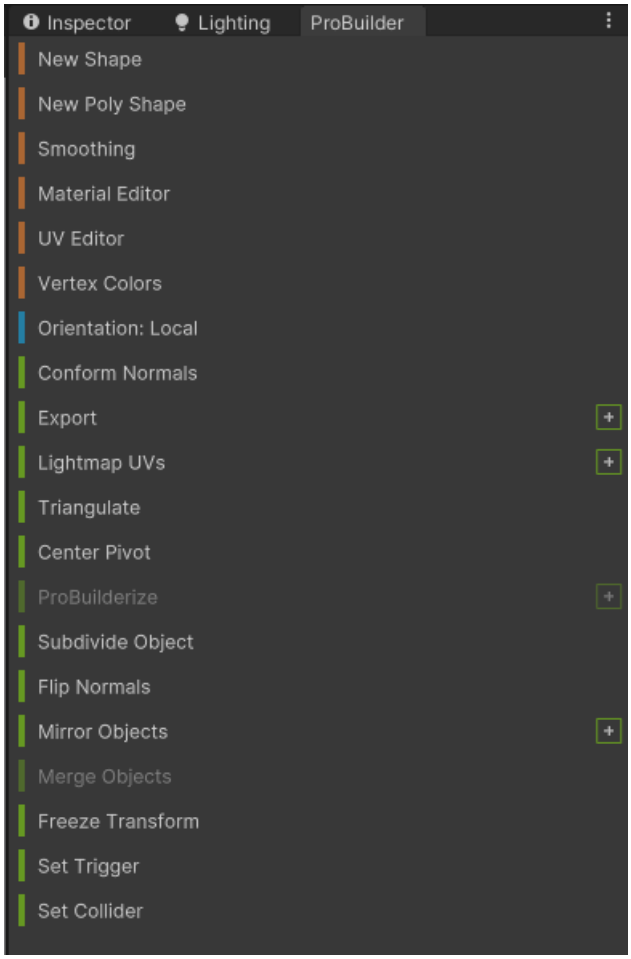


Ilustración 3.232; Funcionalidades de ProBuilder

Un ejemplo de su potencial es crear una figura con bordes “especiales”, la cual no es aportada por Unity. Por ejemplo, para el hangar del final del juego, se tenía esta situación en la que el modelo 3D del hangar no encajaba con las figuras y los objetos aportados por Unity.

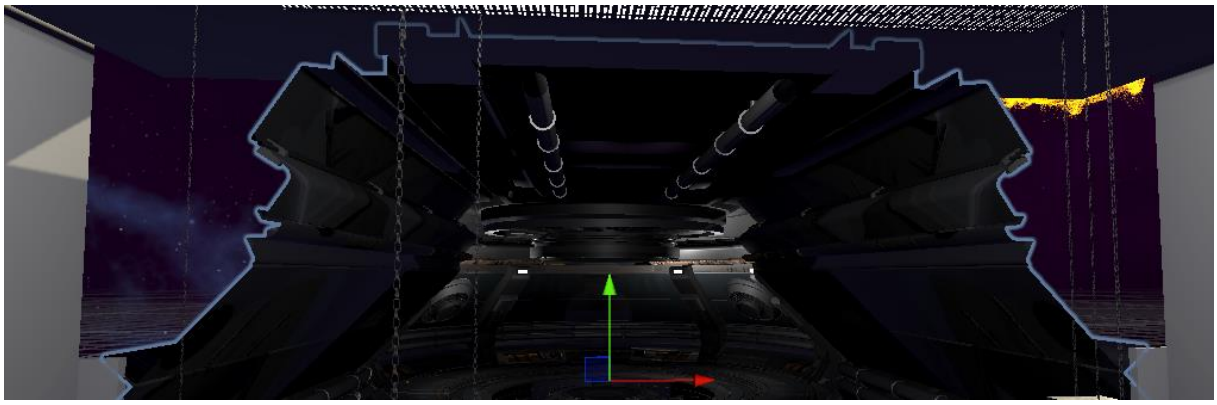


Ilustración 3.233; Huecos del hangar

Para solucionar esto, se usó la herramienta “NewPolyShape” de “ProBuilder”, con la que se pudo crear y modelar una geometría no proporcionada por Unity y específica para lo que se necesitaba.

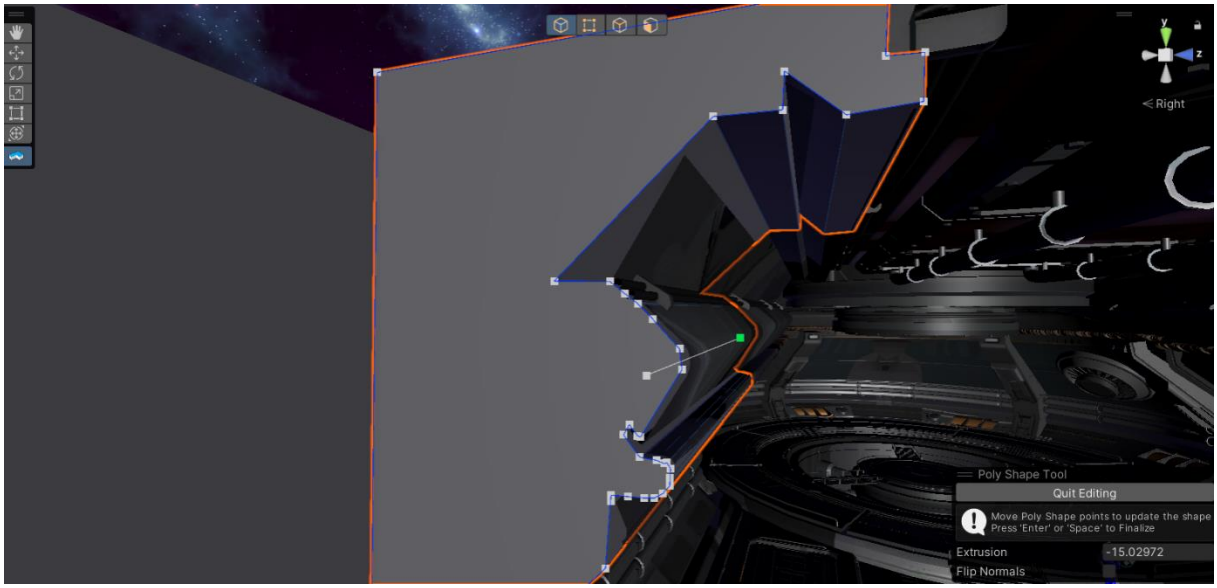


Ilustración 3.234; Proceso de creación de PolyShape hecha con ProBuilder

Hasta crear la figura deseada:

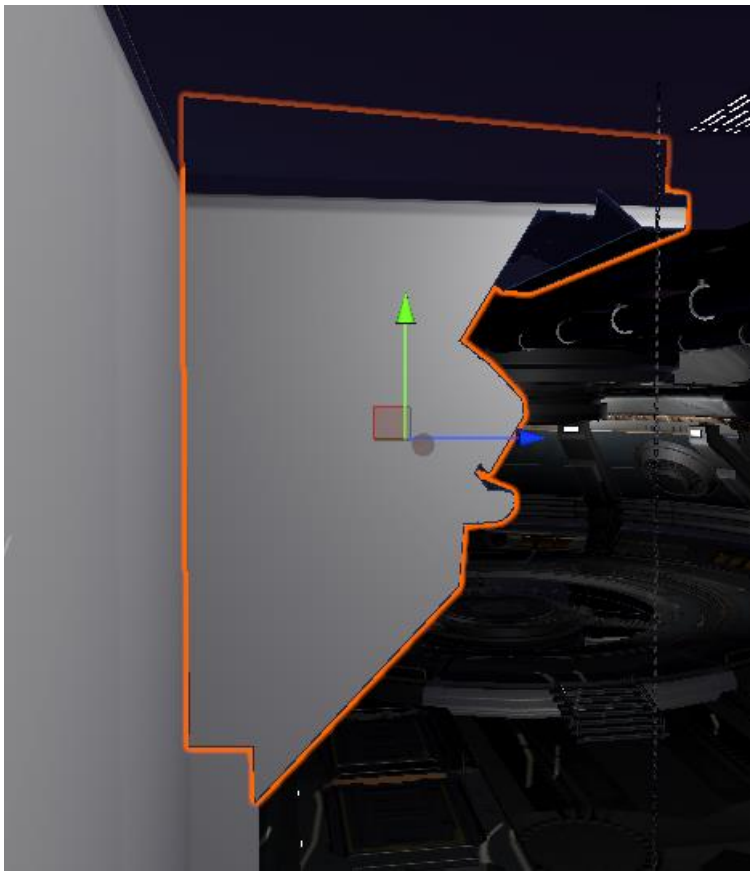


Ilustración 3.235; PolyShape hecha con ProBuilder

A partir de esta figura, se duplicó y se creó una figura simétrica para solucionar la otra parte del hangar.

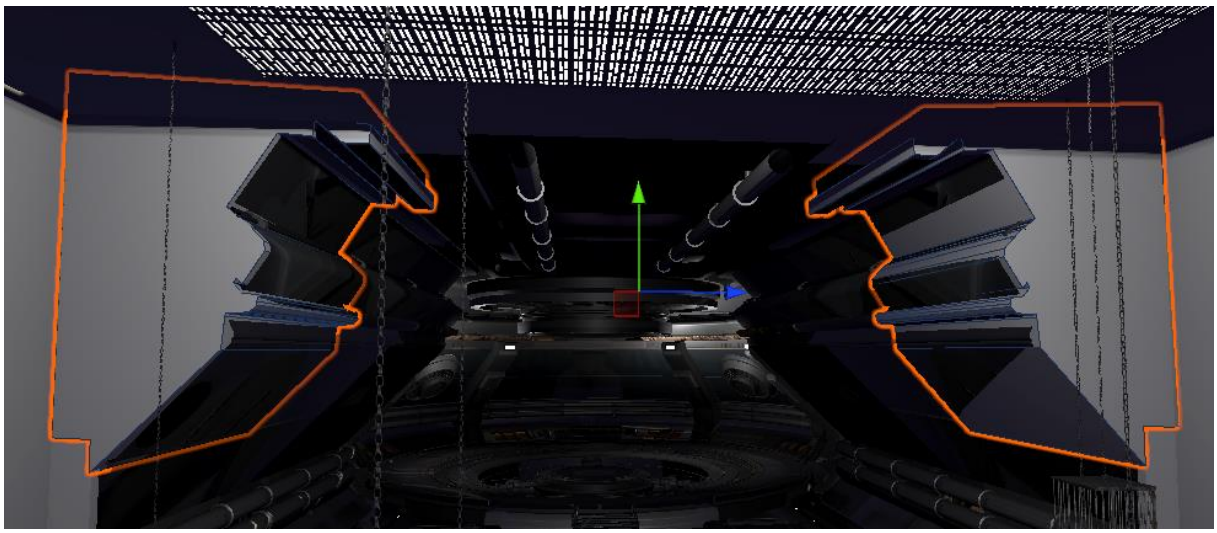


Ilustración 3.236; PolyShape hecha con ProBuilder rellena los huecos del hangar

3.3.2 Shader Graph

El “Shader Graph”, como se ha comentado en apartados anteriores, es una herramienta que busca proporcionar una interfaz sencilla e intuitiva a la hora de crear shaders, con el fin de que los desarrolladores no tengan que saber lenguaje HLSL o GLSL para programar shader, lo cual es bastante complejo.

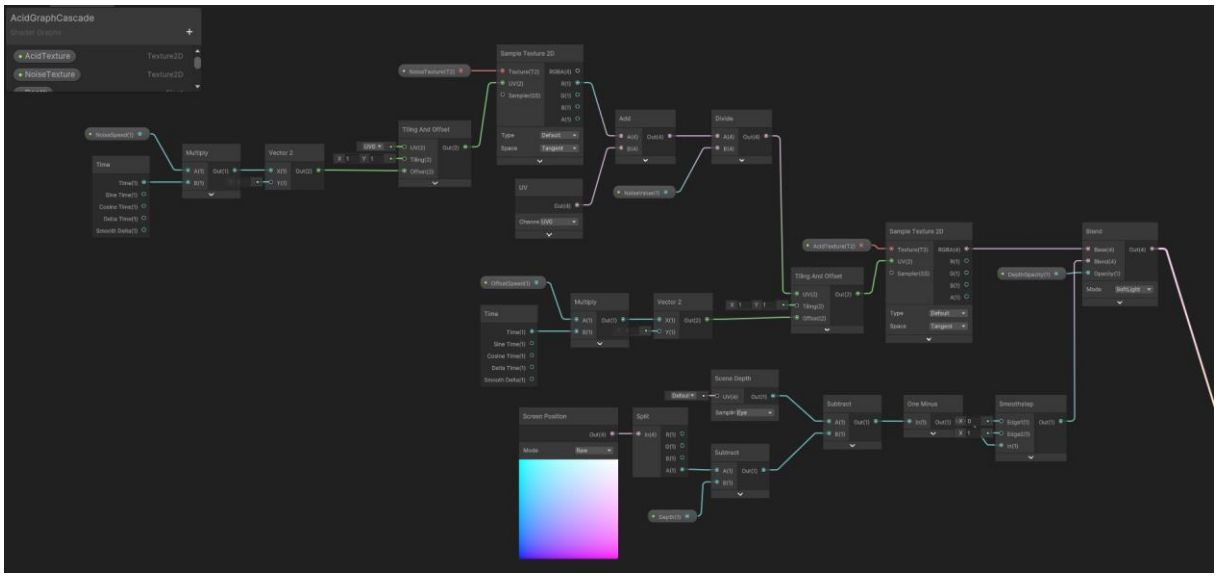


Ilustración 3.237; Ejemplo de ShaderGraph

3.3.3 Profiler

“Profiler” [108] es una herramienta de Unity que se utiliza para monitorear y registrar el rendimiento de la aplicación en tiempo real. Esta herramienta, proporciona datos detallados sobre el rendimiento, uso de la CPU, uso de la GPU, memoria, red, …

Ilustración 3.238; Ejemplo de rendimiento de una ejecución capturado por Profiler

Gracias a esta herramienta, se han podido detectar errores de bajo rendimiento en ciertos puntos de la ejecución de la aplicación o del programa. En la siguiente imagen se puede ver un ejemplo de los datos de rendimiento que la herramienta ha captado durante una ejecución de ejemplo, pudiendo ver qué elementos y componentes han consumido y afectado al rendimiento del juego.

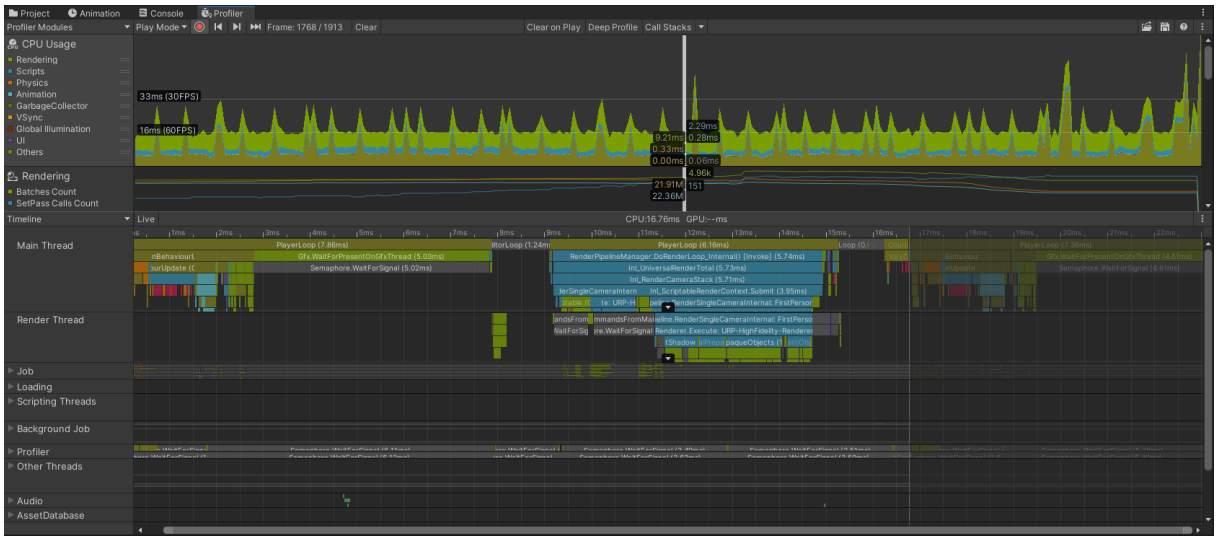


Ilustración 3.239; Detalles del rendimiento de una ejecución capturado por Profiler

4 CONCLUSIONES Y TRABAJOS FUTUROS

A lo largo de este prolongado y ambicioso proyecto, se ha conseguido desarrollar un videojuego FPS con algunas ideas y rasgos propios de sagas clásicas como Doom y StarWars, apoyado de algunas técnicas o mecánicas de los propios videojuegos de las sagas citadas anteriormente o incluso de juegos como Minecraft. Para conseguir desarrollar este proyecto, se ha tenido que realizar una gran investigación de las diferentes técnicas y mecánicas que se han querido implementar, lo que ha conllevado una gran colección de videos, fuentes y documentos, e incluso, asesoramiento profesional para poder conseguir ciertos aspectos del juego, tales como animación de los modelos 3D, curvas de Bèzier, optimización visual de sombras, y algunos otros detalles del videojuego. Además de las funcionalidades desarrolladas, también se ha aprendido en profundidad a cerca del uso de una herramienta de desarrollo de videojuegos y de algunas de sus herramientas, las cuales han sido complicadas de aprender, pero que, finalmente, han facilitado el trabajo. Para poder realizar todo este trabajo, se ha requerido del aprendizaje de algunos conceptos del desarrollo de videojuegos como por ejemplo el “ScriptableObject”, los “PlayerPrefs” [57], “Shaders”, “Colliders”, “Corrutinas”, “Time.Deltatime” [109] , … Por otro lado, también han sido necesarios conocimientos de gestión de proyectos, como el uso de control de versiones Git [110] y el desarrollo basado en metodologías de desarrollo ágiles y todo lo que ello conlleva.

Sumado a estos ámbitos, se ha de comentar que el desarrollo de este proyecto ha ayudado a propiciar la creatividad al poder hacer, aunque regido a una adecuación y a un ámbito, que ha sido el del juego FPS, lo que se ha querido y lo que ha ido saliendo a medida que se desarrollado el juego.

Sin duda alguna, por todo lo comentado previamente, se debe afirmar que, gracias al TFG, se ha podido adquirir una gran cantidad de conocimientos en el ámbito de videojuegos, desarrollo software y gestión de proyectos que me servirán en mi futuro profesional.

Para terminar, comentar que este proyecto, desde antes incluso de empezar a desarrollar, en las etapas en las que se pensaban las cosas que se iban a desarrollar

sin haber tocado nada del proyecto, hasta en esta propia conclusión, se ha cogido con muchas ganas, por lo que no han sido pesadas la gran cantidad de horas, esfuerzo y dedicación que se ha implantado en este proyecto, lo que podemos verificar con el gran resultado obtenido, mucho mejor frente a las expectativas iniciales.

No obstante, este proyecto tiene algunos aspectos mejorables. Como se comentó en diversos apartados de este documento, este proyecto tiene como uno de sus objetivos, ser la base para los desarrolladores que quieran mejorar o expandir el juego. Algunos de los aspectos a mejorar son:

1. Nuevas habilidades e ítems que el jugador pueda usar en su aventura, como un escudo de protección, pociones de mejoras de cadencia o daño de disparo del arma, …
2. Nuevos enemigos con diferentes habilidades
3. Nuevos controles con los que se pueda hacer más accesible e inclusivo para todas las personas del mundo con todo tipo de limitaciones físicas, como controles con gestos, sonidos, con diferentes dispositivos, …
4. Adaptar el juego a nuevas plataformas
5. Añadir más plantas a la nave, incluyendo nuevos niveles como puzles o tiendas en las que comprar ítems o habilidades
6. Mejorar el final, incluyendo una última misión como la de escapar de la nave en un tiempo determinado
7. Recompensa por disparar a un enemigo en la cabeza o en sus partes más débiles, en función del enemigo (por ejemplo: el dron aéreo, sus partes más débiles son sus alas o ventiladores).
8. Incluir jugabilidad multijugador online y/o local.
9. Diseño y estilo específico de los modelos 3D para el juego
10. Añadir una lógica e inteligencia artificial para los enemigos
11. Mejorar el diseño de la nave espacial enemiga, incluyéndole nuevos y mejores assets visuales que mejoren visualmente la nave

12. Adaptar el juego a realidad virtual u otros paradigmas.
13. Adaptar mejor las técnicas y efectos de post-procesamiento [111]
14. Incluir nuevas armas con las que el jugador pueda eliminar enemigos.
15. Incluir nuevos modos de juego, como una carrera, un contrarreloj o un modo supervivencia en el que el jugador deba aguantar lo máximo posible en oleadas de enemigos.

Estos son algunos de los aspectos mejorables del proyecto. No obstante, existen casi infinitas cosas mejorables, el límite, es la imaginación, y eso es lo que hace bonito a este tipo de proyectos.

5 APÉNDICES

En este apartado se tratará información adicional necesaria para entender ciertas partes del proyecto desarrollado.

5.1 Guía original del Trabajo Fin de Título

En este subapartado se incluirá información acerca de la guía original del TFG publicada en la web de la EPS en el momento de la convocatoria, así como el histórico de modificaciones que hayan podido ocurrir de la propuesta inicial.

Primeramente, se aporta información acerca de la propuesta inicial del TFG:

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

PROPUESTA DE TRABAJO DE FIN DE GRADO

GRADO EN:
Grado en Ingeniería Informática

ESPECIALIDAD:
Tecnologías de la Información

MENCIÓN:
Sistemas Gráficos

TÍTULO DEL TRABAJO:
Desarrollo de un videojuego FPS

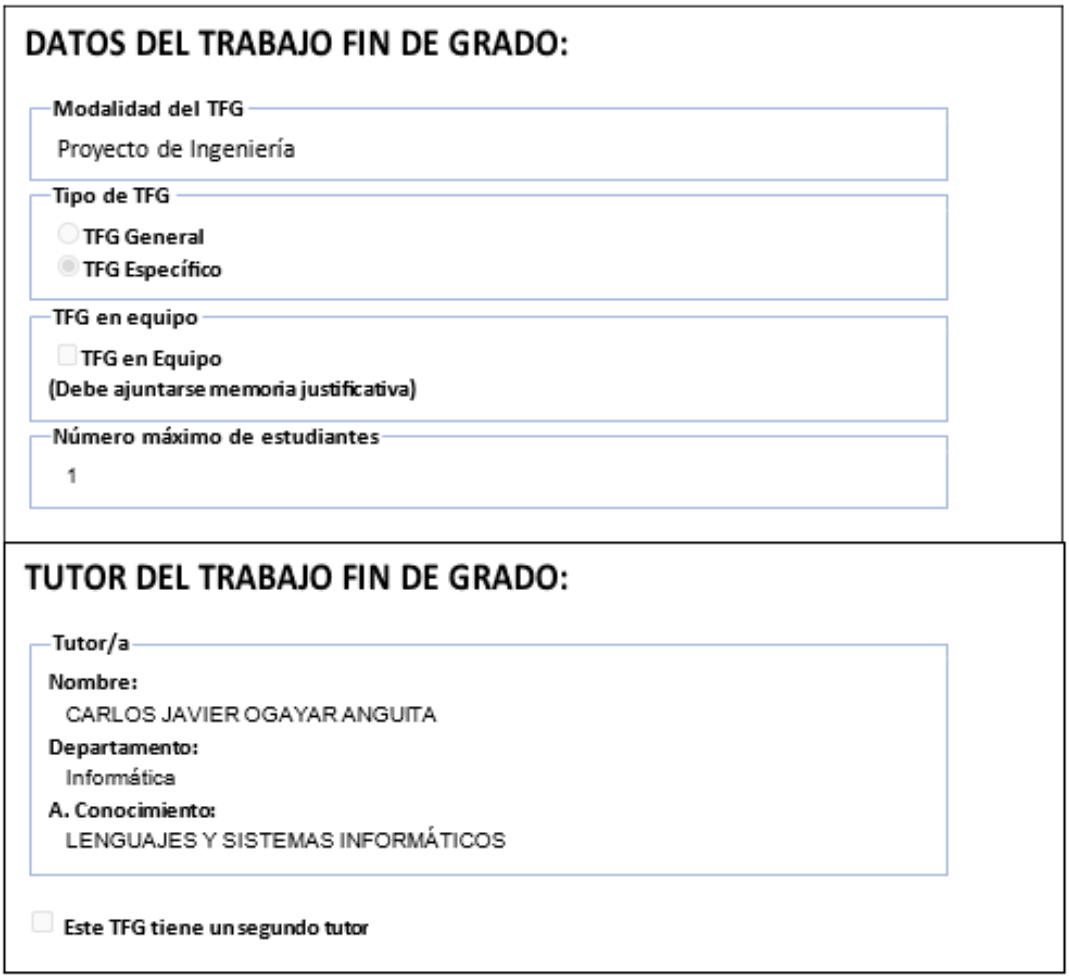
Idioma:
Castellano

DESCRIPCION CORTA DEL TFG:
Este trabajo consiste en la realización de un videojuego tipo FPS (First-Person Shooter) en 3D utilizando un motor de videojuegos actual, como Unreal Engine, CryEngine o Unity. El motor a utilizar queda a elección del alumnado.

Hay que describir las mecánicas de juego y la experiencia de usuario perseguida, incluyendo el sistema de recompensas, los mecanismos de progreso del usuario, los recursos gráficos y sonoros empleados, etc. Para la elaboración de escenas, personajes, NPCs, objetos, animaciones y todo tipo de material gráfico y sonoro, se podrán utilizar elementos descargables de la red, incluyendo sprites originales o cualquier otro recurso.

Ilustración 5.1; apéndice: propuesta del TFG

Ahora, se detallan datos del TFT y del tutor asignado:



DATOS DEL TRABAJO FIN DE GRADO:

Modalidad del TFG

Proyecto de Ingeniería

Tipo de TFG

○ TFG General
◉ TFG Específico

TFG en equipo

☐ TFG en Equipo
(Debe ajuntarse memoria justificativa)

Número máximo de estudiantes

1

TUTOR DEL TRABAJO FIN DE GRADO:

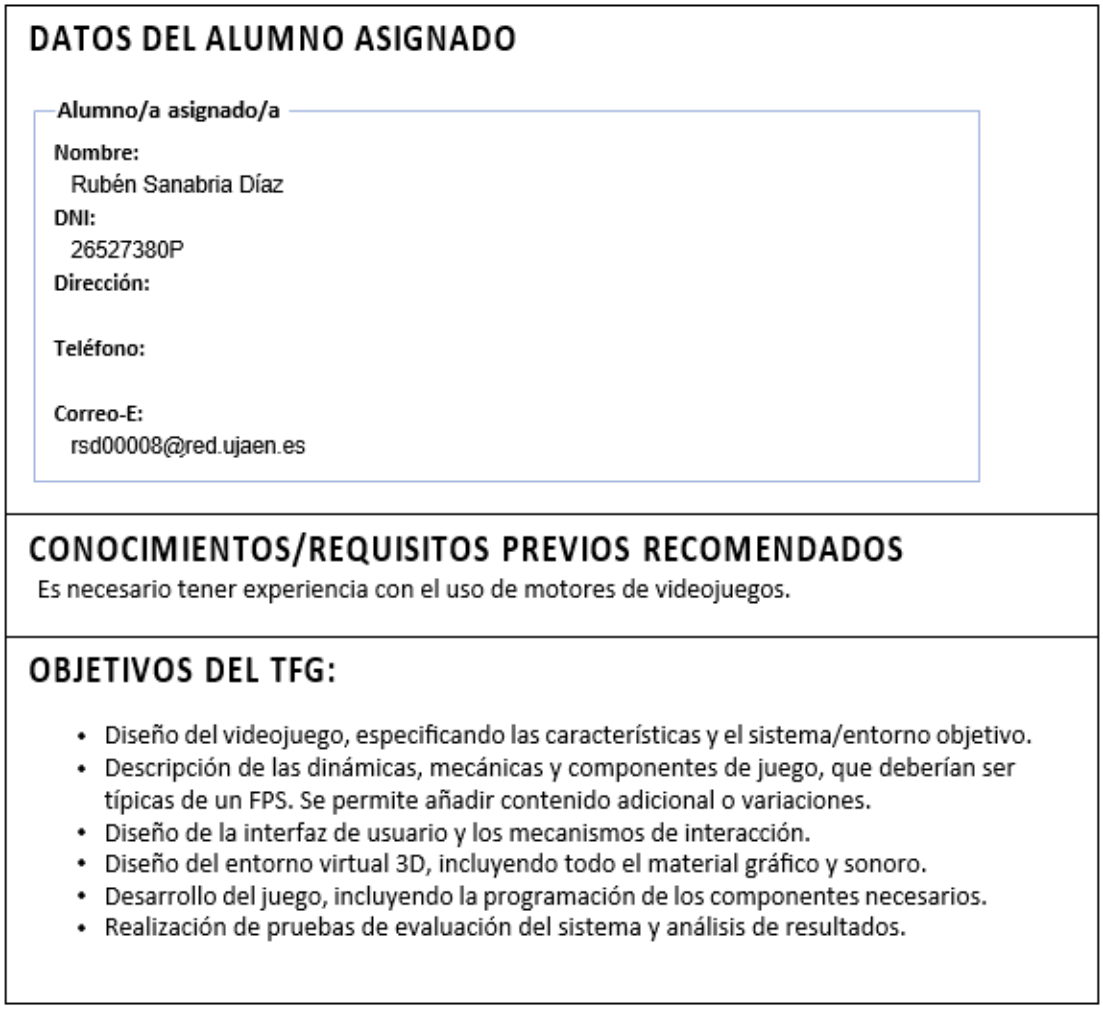
Tutor/a

Nombre:
CARLOS JAVIER OGAYAR ANGUITA
Departamento:
Informática
A. Conocimiento:
LENGUAJES Y SISTEMAS INFORMÁTICOS

☐ Este TFG tiene un segundo tutor

Ilustración 5.2; apéndice: datos del TFG y tutor asignado

Seguidamente, se aporta información acerca del alumno y desarrollador asignado para la realización del TFG, además de conocimientos previos y objetivos del TFG:



DATOS DEL ALUMNO ASIGNADO

Alumno/a asignado/a

Nombre:
Rubén Sanabria Díaz

DNI:
26527380P

Dirección:

Teléfono:

Correo-E:
rsd00008@red.ujaen.es

CONOCIMIENTOS/REQUISITOS PREVIOS RECOMENDADOS

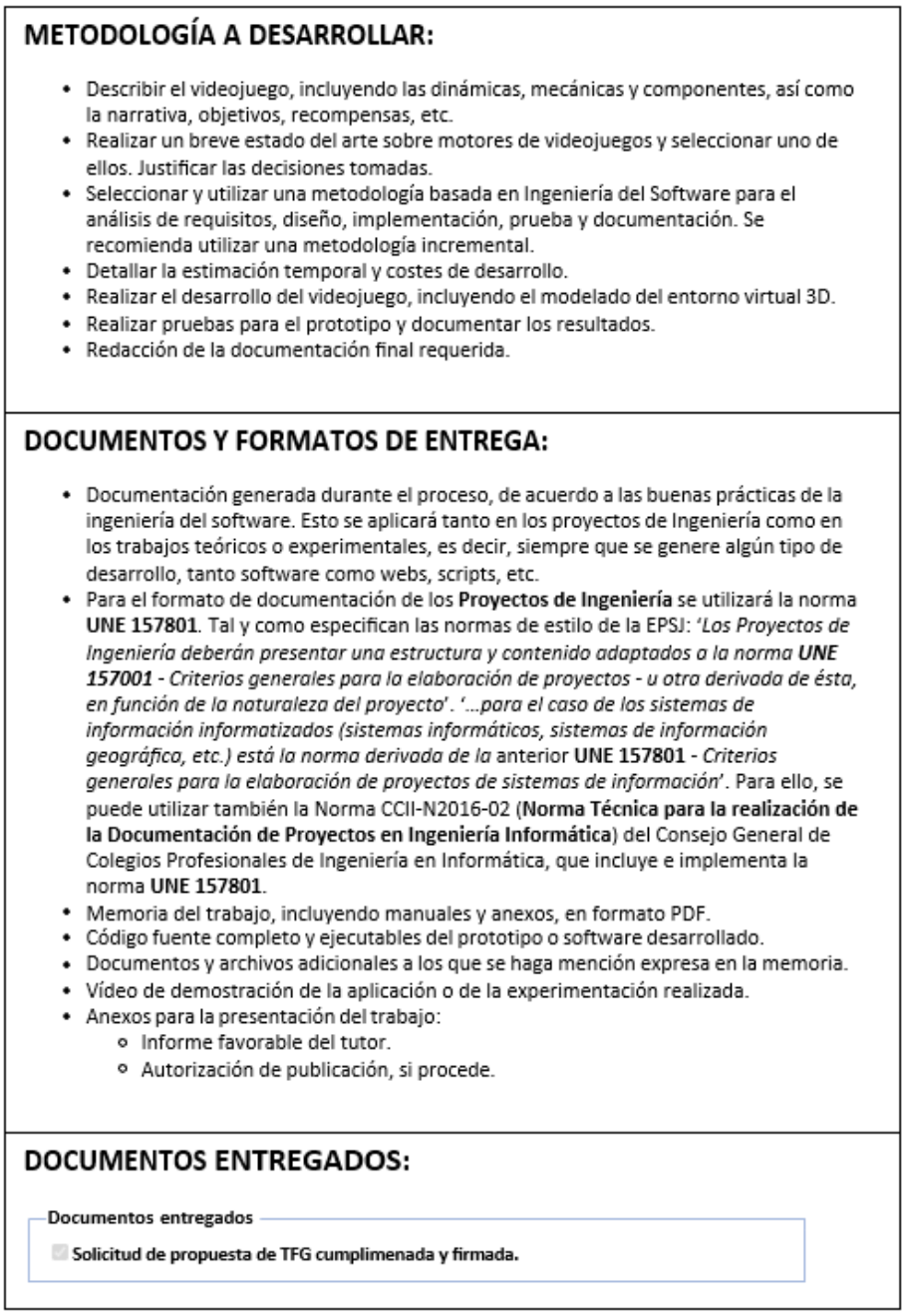
Es necesario tener experiencia con el uso de motores de videojuegos.

OBJETIVOS DEL TFG:

- Diseño del videojuego, especificando las características y el sistema/entorno objetivo.
- Descripción de las dinámicas, mecánicas y componentes de juego, que deberían ser típicas de un FPS. Se permite añadir contenido adicional o variaciones.
- Diseño de la interfaz de usuario y los mecanismos de interacción.
- Diseño del entorno virtual 3D, incluyendo todo el material gráfico y sonoro.
- Desarrollo del juego, incluyendo la programación de los componentes necesarios.
- Realización de pruebas de evaluación del sistema y análisis de resultados.

Ilustración 5.3; apéndice: objetivos del TFG y alumno asignado

Por último, se detalla la metodología a desarrollar y los documentos y formatos de entrega:



METODOLOGÍA A DESARROLLAR:

- Describir el videojuego, incluyendo las dinámicas, mecánicas y componentes, así como la narrativa, objetivos, recompensas, etc.
- Realizar un breve estado del arte sobre motores de videojuegos y seleccionar uno de ellos. Justificar las decisiones tomadas.
- Seleccionar y utilizar una metodología basada en Ingeniería del Software para el análisis de requisitos, diseño, implementación, prueba y documentación. Se recomienda utilizar una metodología incremental.
- Detallar la estimación temporal y costes de desarrollo.
- Realizar el desarrollo del videojuego, incluyendo el modelado del entorno virtual 3D.
- Realizar pruebas para el prototipo y documentar los resultados.
- Redacción de la documentación final requerida.

DOCUMENTOS Y FORMATOS DE ENTREGA:

- Documentación generada durante el proceso, de acuerdo a las buenas prácticas de la ingeniería del software. Esto se aplicará tanto en los proyectos de Ingeniería como en los trabajos teóricos o experimentales, es decir, siempre que se genere algún tipo de desarrollo, tanto software como webs, scripts, etc.
- Para el formato de documentación de los **Proyectos de Ingeniería** se utilizará la norma **UNE 157801**. Tal y como especifican las normas de estilo de la EPSJ: '*Los Proyectos de Ingeniería deberán presentar una estructura y contenido adaptados a la norma **UNE 157001** - Criterios generales para la elaboración de proyectos - u otra derivada de ésta, en función de la naturaleza del proyecto*'. '*...para el caso de los sistemas de información informatizados (sistemas informáticos, sistemas de información geográfico, etc.) está la norma derivada de la* anterior **UNE 157801** - *Criterios generales para la elaboración de proyectos de sistemas de información*'. Para ello, se puede utilizar también la Norma CCII-N2016-02 (**Norma Técnica para la realización de la Documentación de Proyectos en Ingeniería Informática**) del Consejo General de Colegios Profesionales de Ingeniería en Informática, que incluye e implementa la norma **UNE 157801**.
- Memoria del trabajo, incluyendo manuales y anexos, en formato PDF.
- Código fuente completo y ejecutables del prototipo o software desarrollado.
- Documentos y archivos adicionales a los que se haga mención expresa en la memoria.
- Vídeo de demostración de la aplicación o de la experimentación realizada.
- Anexos para la presentación del trabajo:
  - Informe favorable del tutor.
  - Autorización de publicación, si procede.

DOCUMENTOS ENTREGADOS:

Documentos entregados

Solicitud de propuesta de TFG cumplimenada y firmada.

Ilustración 5.4; apéndice: metodología y documentos y formatos de entrega del TFG

Cabe mencionar que la propuesta inicial del TFG fue un proyecto con título "Remake de un videojuego clásico", pero que sufrió un cambio y pasó a titularse "Desarrollo de videojuego FPS".

5.2 Instalación y configuración del sistema

El videojuego desarrollado en este proyecto no requiere de ninguna instalación previa, sino que el motor Unity puede crear una carpeta con el ejecutable del proyecto.

Es importante mencionar que solo es posible ejecutar el archivo ejecutable del juego en un sistema operativo Windows [112]. Por este motivo, si se quiere ejecutar en otro sistema operativo, se ha de usar algún software para emular el sistema operativo Windows.

Ilustración 5.5; carpeta ejectuable del juego

Para ejecutar el juego, lo único que se debe hacer es ejecutar el archivo ".exe" [113]de la carpeta "Ejectuable_SpaceWars". Cabe destacar que el ejecutable no funciona si no está en la misma carpeta que los archivos necesarios para la ejecución, por lo que **se recomienda no sacar el archivo de la carpeta**.

Una vez ejecutado el archivo ".exe", se abrirá aplicación y aparecerá el menú principal del juego.

En cuanto a la configuración del sistema, esta se puede hacer una vez ejecutado el juego e iniciado una partida, desde el menú de opciones. Ahí se podrán encontrar diversos recursos para configurar tanto el volumen de los sonidos y música, como la sensibilidad de la cámara. La configuración que se recomienda es:

1. Volumen de sonido: al máximo
2. Volumen de música: a la mitad de lo máximo
3. Sensibilidad de la cámara: a la mitad de lo máximo

5.3 Manuales de usuario

El software desarrollado en este proyecto, como se ha comentado y detallado a lo largo de este documento, tiene diversas funcionalidades. Para que el jugador sepa cómo usar todas y cada una de esas funcionalidades a través de los dispositivos de entrada, se incluye un manual de instrucciones.

Para detallar el manual de instrucciones, se presenta una imagen en la que se ven las teclas a usar y cada efecto que tiene el pulsar una determinada tecla.

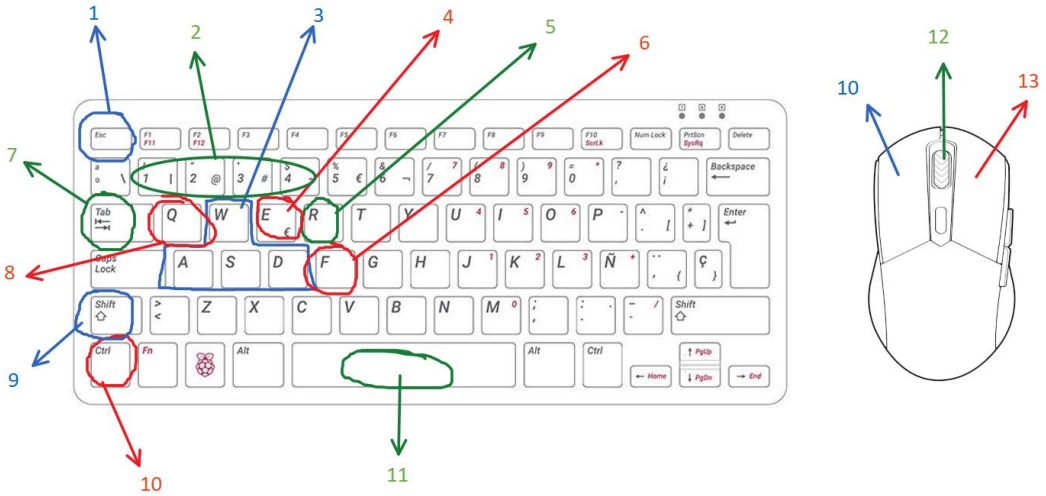


Ilustración 5.6; teclas de dispositivos de entrada

Como se ha comentado, cada tecla tiene su función dentro del juego. Se va a presentar una tabla para recoger la función de cada tabla:

Número de la imagen	Tecla/s	Función o efecto
1	Esc	Abrir menú de opciones
2	Números 1,2,3,4	Seleccionar pociones o batería
3	WASD	Movimiento del personaje
4	E	Interactuar
5	R	Recargar
6	F	Encender/apagar linterna
7	Tab	Abrir/cerrar panel de tareas
8	Q	Usar poción seleccionada

9	Shift	Correr mientras se mueve el personaje
10	Ctrl y click izq.	Disparar
11	Barra espacio	Saltar
12	Rueda del ratón	Cambiar arma
13	Click derecho	Apuntar con el arma

Tabla 5.1; tabla de tecla y su funcionamiento

6 DEFINICIONES Y ABREVIATURAS

Para facilitar la comprensión del texto de este documento, se procede a describir la terminología y conceptos empleados:

1. **Inventario**: esto hace referencia a un concepto virtual, imaginario y no definido ni implementado explícitamente del inventario del jugador, en el que se almacenan pociones, munición, …
2. **Jugador**: el jugador es el usuario que juega una partida
3. **Personaje**: el personaje es el protagonista de la historia, el cual está dentro del juego.
4. **StarLord**: es el jefe final del juego.
5. **EEDD**: siglas de Estructuras de Datos
6. **FPS**: en función del contexto, puede significar las siglas de First Person Shooter o Frame Per Second
7. **NPC**: siglas de la expresión inglesa “Non Playable Character”, la cual hace referencia a un personaje dentro del juego que el jugador no controla, sino que lo controla el sistema
8. **Prefab**: en Unity los prefabs son objetos de juego que se pueden crear para luego reutilizar en diferentes lugares del proyecto y que siguen el patrón peso ligero. Es importante su uso porque mejora la eficiencia de los objetos en una aplicación mediante la reducción de su uso de memoria.

9. **Asset**: es un recurso, es decir, elemento utilizado en la creación de un juego o aplicación. Estos pueden ser: modelos, texturas, efectos de sonido, scripts, ...
10. **Ítem**: es cualquier objeto utilizable por el jugador.
11. **ScriptableObject**: es una clase especial en Unity que se puede usar para almacenar grandes cantidades de datos de una manera organizada y eficiente. A diferencia de los “MonoBehaviour”, los ScriptableObjects no se aplican a ningún “GameObject” de Unity, ni tampoco participan directamente en la lógica del juego en tiempo de ejecución.
12. **Cliente:** es la persona que espera recibir el resultado del proyecto. Se hace referencia al tutor del mismo.
13. **StakeHolder:** son personas con interés en el proyecto y que pueden influir en el por este motivo. En este proyecto, se hace referencia a amigos, familiares o compañeros.
14. **Sprint:** es un ciclo de trabajo en una metodología ágil Scrum
15. **Backlog:** es una lista de tareas que quedan por realizar.
16. **Puntos de historia (Story points, SP):** son una unidad de medida utilizada en metodologías ágiles para estimar el esfuerzo de realizar una tarea de un sprint
17. **Diagrama de velocidad**: es una gráfica que muestra la cantidad de trabajo completado en un sprint.
18. **Diagrama Burndown**: es una gráfica que muestra la cantidad de trabajo restante en un sprint.
19. **Gameplay**: experiencia de una partida de un juego.

7 BIBLIOGRAFÍA

[1] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Videojuego_de_disparos_en_primera_persona.

[2] C. Rollié, «Taller de Diseño en Comunicación Visual B, UNLP | FBA,» [En línea]. Available: https://sied.ead.unlp.edu.ar/assets/files/sist%20graf%20.pdf.

[3] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Doom_(videojuego_de_1993).

[4] Software, id, «id Software,» [En línea]. Available: https://www.idsoftware.com/es.

[5] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/WAD.

[6] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Star_Wars.

[7] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/George_Lucas.

[8] PEGI, «PEGI Public Site,» [En línea]. Available: https://pegi.info/es.

[9] (ESRB), Entertainment Software Rating Board, «ESRB Ratings,» [En línea]. Available: https://www.esrb.org.

[10] (CERO), Computer Entertainment Rating Organization, «CERO,» [En línea]. Available: https://www.cero.gr.jp/en/smarts/index/17/.

[11] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Motor_de_videojuego.

[12] J. G. Bond, Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C#, Addison-Wesley Professional, 2023.

[13] Technologies, Unity, «Unity,» [En línea]. Available: https://unity.com/es.

[14] Foundation, Blender, «Blender.org,» [En línea]. Available: https://www.blender.org.

[15] O. Villar, Aprenda a crear personajes en Blender: El principal software 3D de código abierto, Marcombo, 2023.

[16] Wikipedia, «Wikipedia,» [En línea]. Available: https://nl.wikipedia.org/wiki/Ton_Roosendaal.

[17] Adobe, «Mixamo,» [En línea]. Available: https://www.mixamo.com.

[18] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://learn.unity.com/tutorial/working-with-animation-rigging.

[19] The Git Project, «Git SCM,» [En línea]. Available: https://www.git-scm.com.

[20] B. Moure, Git y GitHub desde cero: Guía de estudio teórico-práctica paso a paso más curso en vídeo, Independente, 2023.

[21] UXPin, «UXPin,» [En línea]. Available: https://www.uxpin.com.

[22] Visual Paradigm, «Visual Paradigm,» [En línea]. Available: https://www.visual-paradigm.com.

[23] GanttProject, «GanttProject,» [En línea]. Available: https://www.ganttproject.biz.

[24] Wikipedia, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Third-person_shooter.

[25] Wikipedia, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Real-time_strategy#:~:text=Real-time%20strategy%20(RTS),players%20take%20turns%20to%20play..

[26] Wikipedia, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Role-playing_game.

[27] Wikipedia, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Battle_royale_game.

[28] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Simulador.

[29] Unreal Engine, «Unreal Engine,» [En línea]. Available: https://www.unrealengine.com/es-ES.

[30] Godot Engine, «Godot Engine,» [En línea]. Available: https://godotengine.org.

[31] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Macintosh.

[32] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/MacOS.

[33] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/LTS.

[34] Atlassian, «Atlassian,» [En línea]. Available: https://www.atlassian.com/es/agile/scrum.

[35] J. Sutherland, El revolucionario método para trabajar el doble en la mitad de tiempo, Ediciones Deusto, 2015.

[36] Universidad de Alicante, «Repositorio Institucional de la Universidad de Alicante,» [En línea]. Available: https://rua.ua.es/dspace/bitstream/10045/137222/1/JENUI_2023_047.pdf.

[37] OpenWebinars, «OpenWebinars,» [En línea]. Available: https://openwebinars.net/blog/metodologia-lean-que-es-y-como-implementarla-en-empresas-it/.

[38] IONOS Digital Guide, «IONOS Digital Guide,» [En línea]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/#:~:text=El%20desarrollo%20en%20cascada%20(en,ejecuta%20tan%20solo%20una%20vez..

[39] Deloitte, «Deloitte,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-el-desarrollo-en-espiral.html.

[40] Asana, «Asana,» [En línea]. Available: https://asana.com/es/resources/extreme-programming-xp.

[41] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Sucesión_de_Fibonacci.

[42] Deloitte, «Deloitte,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/roles-y-responsabilidades-scrum.html.

[43] Atlassian, «Atlassian,» [En línea]. Available: https://www.atlassian.com/es/agile/project-management/gantt-chart.

[44] PCComponentes, «PCComponentes,» [En línea]. Available: https://www.pccomponentes.com.

[45] Tecfys, «Tecfys Blog,» [En línea]. Available: https://tecfys.com/blog/post/23-conoces-la-vida-media-de-tu-ordenador#:~:text=Por%20lo%20general%2C%20la%20vida,entre%20los%203%20-%205%20años..

[46] Talent.com, «Talent.com,» [En línea]. Available: https://es.talent.com/salary.

[47] Visure Solutions, «Visure Solutions Blog,» [En línea]. Available: https://visuresolutions.com/es/blog/functional-requirements.

[48] Visure Solutions, «Visure Solutions Blog,» [En línea]. Available: https://visuresolutions.com/es/blog/non-functional-requirements/.

[49] Ionos, «Ionos,» [En línea]. Available: www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagrama-de-casos-de-uso/#:~:text=El%20diagrama%20de%20casos%20de%20uso%20es%20una%20forma%20de,de%20programación%20orientada%20a%20objetos..

[50] IBM, «IBM Docs,» [En línea]. Available: https://www.ibm.com/docs/es/product-master/12.0.0?topic=processes-defining-use-cases.

[51] D. J. Díaz, Programación avanzada para videojuegos: Patrones de diseño en C#, Independiente, 2023.

[52] P. Felicia, Hands-On Unity Game Development: Build, customize, and manage your projects with the Unity game development platform (Fourth Edition), Packt Publishing, 2023.

[53] L. Ruelas, Desarrollo de videojuegos para principiantes, ANAYA MULTIMEDIA, 2018.

[54] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/530/Manual/CreatingAndUsingScripts.html.

[55] Refactoring.Guru, «Refactoring.Guru,» [En línea]. Available: https://refactoring.guru/es/design-patterns/singleton.

[56] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html.

[57] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html.

[58] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html.

[59] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/530/Manual/Coroutines.html.

[60] Unity Technologies, «Unity Learn,» [En línea]. Available: https://learn.unity.com/tutorial/interpolacion-linear?language=es&projectId=5dc4a877edbc2a00205e1124#.

[61] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/530/Manual/CreatingScenes.html.

[62] Universidad de Valencia, «Guía para el diseño de páginas web,» [En línea]. Available: https://www.uv.es/jac/guia/frame.htm#:~:text=Los%20frames%20(en%20inglés%20frame,distintos%2C%20aunque%20puedan%20estar%20relacionados..

[63] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/AudioSource.html.

[64] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Curva_de_Bézier.

[65] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Composición_alfa.

[66] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/530/Manual/Tags.html.

[67] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/class-MeshCollider.html.

[68] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Collider-isTrigger.html.

[69] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Collider.html.

[70] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/CharacterController.html.

[71] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/Shaders.html.

[72] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/SL-ShaderPrograms.html.

[73] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/SL-SubShader.html.

[74] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/2018.4/Manual/SL-VertexFragmentShaderExamples.html.

[75] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/2018.4/Manual/class-BoxCollider.html.

[76] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/class-AnimatorController.html.

[77] Unity Technologies, «Unity Asset Store,» [En línea]. Available: https://assetstore.unity.com/packages/tools/particles-effects/volumetric-lines-29160.

[78] Unity Technologies, «Unity Asset Store,» [En línea]. Available: https://assetstore.unity.com.

[79] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Packages/com.unity.shadergraph@6.9/manual/Sample-Texture-2D-Node.html.

[80] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Máscara_(informática).

[81] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/2019.3/Manual/class-CapsuleCollider.html.

[82] U. Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/2019.4/Manual/class-Rigidbody.html.

[83] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Animator.html.

[84] Technologies, Unity, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/class-BlendTree.html.

[85] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/530/ScriptReference/CharacterController-isGrounded.html.

[86] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/2019.4/Manual/UICanvas.html.

[87] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/OcclusionCulling.html.

[88] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Transform.html.

[89] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnGUI.html.

[90] Mojang Studios, «Minecraft,» [En línea]. Available: https://www.minecraft.net/es-es.

[91] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/530/Manual/ParticleSystems.html.

[92] Unity Technologies, «Unity Technologies,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html.

[93] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/2018.4/Manual/class-TrailRenderer.html.

[94] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Physics.CheckSphere.html.

[95] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Physics.html.

[96] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnDrawGizmos.html.

[97] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/script-AnimationWindowEvent.html.

[98] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/530/Manual/Prefabs.html.

[99] U. Technologies, «Unity Asset Store,» [En línea]. Available: https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488.

[100] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/class-LineRenderer.html.

[101] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/2018.4/Manual/class-SphereCollider.html.

[102] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/class-ScriptableObject.html.

[103] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/lighting-emissive-materials.html.

[104] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/Lighting.html.

[105] Wikipedia, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Star_Wars_opening_crawl.

[106] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Camera-fieldOfView.html.

[107] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://unity.com/es/features/probuilder.

[108] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/Profiler.html.

[109] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/ScriptReference/Time-deltaTime.html.

[110] The Git Project, «Git SCM,» [En línea]. Available: https://www.git-scm.com.

[111] Unity Technologies, «Unity Documentation,» [En línea]. Available: https://docs.unity3d.com/es/2017.4/Manual/PostProcessingOverview.html#:~:text=Post-processing%20(post-procesamiento,con%20poco%20tiempo%20de%20configuración..

[112] Microsoft, «Windows,» [En línea]. Available: https://www.microsoft.com/es-es/windows?r=1.

[113] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Ejecutable.

[114] R. Pressman, Ingeniería del Software, McGraw-Hill, 2010.

[115] yoo. [En línea]. Available: https://www.amazon.es/Programación-avanzada-para-videojuegos-Patrones/dp/B0CWTYMZ82/ref=sr_1_9?dib=eyJ2IjoiMSJ9.8VLOppeJEgT7aIph3C22MBpXThPpUQvJtCnrxRJeyqsa3_sW1n5N7q7jpF1EC2kPbadfpieLhcQOCMoxSzyRnrZps-teRBSBf2pJFlrm2W6ID6ZC6e3oxUSqZVdyVmmZ0sbzW9o6hO_DAHT.