uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática
2023-2024

Trabajo Fin de Grado

"Desarrollo de un videojuego roguelike multijugador"

Álvaro Morata Hontanaya

Tutor

Alejandro Rey López

Leganés, 2024

RESUMEN

En este documento se detalla el proceso de diseño y desarrollo de un videojuego multijugador en línea, con componentes del género *roguelike*. Asimismo, se realiza un estudio del desarrollo de videojuegos en la actualidad, además de las diferentes alternativas que existen para implementar un modo multijugador.

En la implementación del prototipo desarrollado, se ha empleado el motor de desarrollo de videojuegos Unity Engine, sobre el que se explica en detalle las funcionalidades más relevantes para el desarrollo de este proyecto. También se analizan las alternativas que existen dentro de Unity para implementar un modo multijugador, explicando en detalle Netcode for GameObjects, la solución escogida en este proyecto. Por último, se realiza un análisis de los problemas encontrados durante la fase de implementación, así como sus posibles soluciones.

Finalmente, se muestra la planificación seguida durante la ejecución del proyecto, además de la elaboración de un presupuesto y el impacto que puede tener un proyecto de esta índole sobre la sociedad, economía y medioambiente.

Palabras clave: videojuego; multijugador; Unity; roguelike; C#; Netcode for GameObjects

ABSTRACT

This document outlines the design and development process of an online multiplayer video game, which incorporates elements of the roguelike genre. A study of current video game development is also carried out, in addition to an examination of the various options for implementing a multiplayer mode.

During the implementation of the prototype developed, the game video development engine Unity Engine has been utilised, on which the most relevant functionalities for the development of this project are explained in detail. The alternatives that exist within Unity to implement a multiplayer mode are also analysed, explaining in detail Netcode for GameObjects, the solution chosen for this project. Finally, an analysis is made of the problems encountered during the implementation phase, as well as their possible solutions.

Lastly, the planning followed during the execution of the project is shown, as well as the elaboration of a budget and the impact that a project of this nature can have on society, economy and environment.

Keywords: video game; multiplayer; Unity; roguelike; C#; Netcode for GameObjects

DEDICATORIA

En primer lugar, quiero dar las gracias a mis padres, porque, si no fuese por ellos, no me hubiese sido posible llegar a este nivel educativo, ni estaría hoy en este mundo. Gracias a su insistencia ha salido adelante el desarrollo de este proyecto, algo que agradezco con todo mi corazón.

También quiero dar las gracias a mis hermanos y cuñados, por los consejos, apoyo y comprensión que me han ofrecido en todo momento. Tampoco me olvido de mis dos sobrinas, que motivan a cualquiera con la energía que desprenden.

A mis amigos, por el apoyo psicológico constante que me han demostrado, además de ayudarme a comprobar el funcionamiento del prototipo desarrollado y ofrecerme sugerencias.

Además, quiero dar las gracias a mi tutor, Alejandro, quien me ha orientado y respondido a cualquier duda que tuviese casi al instante, a pesar de no haber mantenido una comunicación constante con él.

Por último, quiero expresar mis agradecimientos a la Universidad Carlos III de Madrid, por haberme ofrecido un nivel educativo incomparable.

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN 1
1.1. Motivación 1
1.2. Objetivos 3
1.3. Estructura del documento 3
2. ESTADO DEL ARTE 5
2.1. Videojuegos y el modo multijugador 5
2.2. Topologías de red en el diseño de juegos multijugador 6
2.2.1. Topologías de área local 6
2.2.2. Arquitecturas de área extensa 7
2.3. Desafíos en el desarrollo de videojuegos multijugador en línea 12
2.4. El género *roguelike* 13
2.4.1. Juegos del género *roguelike* similares a la solución propuesta 15
2.5. Motores de desarrollo de videojuegos 17
2.5.1. Comparativa de los tres motores de videojuegos 20
2.6. Conclusiones del estado de la cuestión 21
3. ANÁLISIS 23
3.1. Casos de uso 23
3.1.1. Diagramas de los casos de uso 23
3.1.1.1. Diagrama del escenario 1 24
3.1.1.2. Diagrama del escenario 2 26
3.1.1.3. Diagrama del escenario 3 27
3.1.2. Descripción de los casos de uso 28
3.2. Especificación de requisitos de software 33
3.2.1. Requisitos funcionales 34
3.2.2. Requisitos no funcionales 44
3.3. Matriz de trazabilidad 45
3.4. Tecnologías utilizadas 48
3.5. Metodología de desarrollo utilizada 49
4. DISEÑO 51
4.1. Apartado artístico 51
4.2. Apartado técnico 53
4.2.1. Interacciones del jugador con el sistema 53

4.2.2. Lógica de los enemigos 56
4.2.3. Flujo de una partida 58
4.2.4. Flujo de escenas y menús 61
4.2.5. Diseño del multijugador 63
4.2.5.1. Propiedad y autoridad sobre los objetos 63
4.2.5.1.1. Autoridad del servidor vs. Autoridad del cliente 63
4.2.5.2. Soluciones para implementar el multijugador en Unity 64
4.2.5.3. Implementación en el prototipo 66
5. EVALUACIÓN 71
5.1. Plan de pruebas 71
5.2. Problemas encontrados en el plan de pruebas 82
5.2.1. P-01 82
5.2.2. P-02 83
5.2.3. P-03 83
6. MARCO REGULADOR 84
6.1. Legislación aplicable 84
6.2. Estándares técnicos 84
6.3. Licencias utilizadas 85
7. PLANIFICACIÓN 87
8. ENTORNO SOCIOECONÓMICO 90
8.1. Presupuesto 90
8.2. Impacto socioeconómico 92
8.2.1. Impacto social 92
8.2.2. Impacto en la educación y sanidad 92
8.2.3. Impacto económico 93
8.2.4. Impacto medioambiental 93
9. CONCLUSIONES 94
9.1. Objetivos cumplidos 94
9.2. Problemas encontrados 95
9.3. Posibles mejoras 95
BIBLIOGRAFÍA 98
DEVELOPMENT OF A ROGUELIKE MULTIPLAYER VIDEOGAME 105
Introduction 105

State of the art 106
Analysis 108
Design 109
Evaluation 112
Regulatory framework 112
Planning 113
Socio-economic environment 113
Conclusions 114

ÍNDICE DE FIGURAS

Fig. 1.1. 50 años de ingresos por videojuegos [1] 1

Fig. 2.1. Recreación del videojuego Tennis for Two [10] 5

Fig. 2.2. Overcooked 2, juego que implementa el multijugador de sofá con un modo cooperativo [19] 7

Fig. 2.3. Esquema de una red P2P 8

Fig. 2.4. Arquitectura Cliente-Servidor 9

Fig. 2.5. Topología Cliente-Anfitrión 10

Fig. 2.6. Arquitectura de autoridad distribuida 11

Fig. 2.7. Captura de pantalla de Rogue en su versión de 1980 [30] 14

Fig. 2.8. Captura de pantalla de Enter the Gungeon 16

Fig. 2.9. Captura de pantalla de Nuclear Throne 17

Fig. 2.10. El juego de VR Beat Saber, desarrollado en Unity [49] 19

Fig. 3.1. Diagrama de casos de uso del escenario 1 24

Fig. 3.2. Diagrama de casos de uso del escenario 2 26

Fig. 3.3. Diagrama de casos de uso del escenario 3 27

Fig. 4.1. *Sprite sheet* del personaje del jugador 1 52

Fig. 4.2. Ejes de coordenadas y dimensiones empleados 53

Fig. 4.3. Esquema explicativo del apuntado del arma 54

Fig. 4.4. *Sprite* del botiquín 55

Fig. 4.5. Modelos 2D de los enemigos. De izquierda a derecha: enemigo 1, enemigo 2 y enemigo 3 57

Fig. 4.6. Diagrama de estados del flujo de la partida 59

Fig. 4.7. Mapa del nivel 1 del prototipo 59

Fig. 4.8. Habitación 0 con su *collider* correspondiente 60

Fig. 4.9. Disposición de la habitación objetivo y el cofre que contiene 61

Fig. 7.1. Diagrama de Gantt del proyecto 88

ÍNDICE DE TABLAS

Tabla 2.1. Plataformas objetivo de Unity [46] 18
Tabla 2.2. Unity vs Unreal vs Godot 20
Tabla 3.1. Casos de uso del escenario 1 25
Tabla 3.2. Casos de uso del escenario 2 26
Tabla 3.3. Casos de uso del escenario 3 27
Tabla 3.4. Plantilla de la descripción de los casos de uso 28
Tabla 3.5. Caso de uso CU01 28
Tabla 3.6. Caso de uso CU02 28
Tabla 3.7. Caso de uso CU03 29
Tabla 3.8. Caso de uso CU04 29
Tabla 3.9. Caso de uso CU05 29
Tabla 3.10. Caso de uso CU06 29
Tabla 3.11. Caso de uso CU07 30
Tabla 3.12. Caso de uso CU08 30
Tabla 3.13. Caso de uso CU09 30
Tabla 3.14. Caso de uso CU10 30
Tabla 3.15. Caso de uso CU11 31
Tabla 3.16. Caso de uso CU12 31
Tabla 3.17. Caso de uso CU13 31
Tabla 3.18. Caso de uso CU14 31
Tabla 3.19. Caso de uso CU15 32
Tabla 3.20. Caso de uso CU16 32
Tabla 3.21. Caso de uso CU17 32
Tabla 3.22. Caso de uso CU18 32
Tabla 3.23. Caso de uso CU19 33
Tabla 3.24. Caso de uso CU20 33
Tabla 3.25. Plantilla de los Requisitos 33
Tabla 3.26. Requisito F-U-C01 34
Tabla 3.27. Requisito F-S-R02 34
Tabla 3.28. Requisito F-U-C03 35
Tabla 3.29. Requisito F-S-R04 35
Tabla 3.30. Requisito F-U-C05 35

Tabla 3.31. Requisito F-U-C06 35
Tabla 3.32. Requisito F-S-R07 36
Tabla 3.33. Requisito F-S-C08 36
Tabla 3.34. Requisito F-U-C09 36
Tabla 3.35. Requisito F-S-R10 36
Tabla 3.36. Requisito F-S-C11 37
Tabla 3.37. Requisito F-S-C12 37
Tabla 3.38. Requisito F-S-R13 37
Tabla 3.39. Requisito F-S-C14 37
Tabla 3.40. Requisito F-U-C15 38
Tabla 3.41. Requisito F-S-C16 38
Tabla 3.42. Requisito F-U-C17 38
Tabla 3.43. Requisito F-S-R18 38
Tabla 3.44. Requisito F-S-C19 39
Tabla 3.45. Requisito F-U-C20 39
Tabla 3.46. Requisito F-S-C21 39
Tabla 3.47. Requisito F-S-R22 39
Tabla 3.48. Requisito F-S-C23 40
Tabla 3.49. Requisito F-U-C24 40
Tabla 3.50. Requisito F-S-C25 40
Tabla 3.51. Requisito F-S-R26 40
Tabla 3.52. Requisito F-U-C27 41
Tabla 3.53. Requisito F-U-C28 41
Tabla 3.54. Requisito F-U-C29 41
Tabla 3.55. Requisito F-U-C30 41
Tabla 3.56. Requisito F-S-C31 41
Tabla 3.57. Requisito F-S-C32 42
Tabla 3.58. Requisito F-S-C33 42
Tabla 3.59. Requisito F-S-C34 42
Tabla 3.60. Requisito F-U-C35 42
Tabla 3.61. Requisito F-S-R36 43
Tabla 3.62. Requisito F-S-C37 43
Tabla 3.63. Requisito F-U-C38 43

Tabla 3.64. Requisito F-U-C39 43
Tabla 3.65. Requisito F-S-C40 44
Tabla 3.66. Requisito F-S-R41 44
Tabla 3.67. Requisito F-S-R42 44
Tabla 3.68. Requisito NF-S-C43 44
Tabla 3.69. Requisito NF-S-C44 45
Tabla 3.70. Requisito NF-S-C45 45
Tabla 3.71. Requisito NF-S-R46 45
Tabla 3.72. Requisito NF-S-R47 45
Tabla 3.73. Matriz de trazabilidad 46
Tabla 4.1. Diferencias entre Autoridad del Servidor y Autoridad del Cliente 63
Tabla 4.2. Diferencias entre NGO y Netcode for Entities 65
Tabla 5.1. Plantilla de los Casos de Prueba 71
Tabla 5.2. CP-01 72
Tabla 5.3. CP-02 72
Tabla 5.4. CP-03 73
Tabla 5.5. CP-04 73
Tabla 5.6. CP-05 74
Tabla 5.7. CP-06 75
Tabla 5.8. CP-07 75
Tabla 5.9. CP-08 76
Tabla 5.10. CP-09 76
Tabla 5.11. CP-10 77
Tabla 5.12. CP-11 77
Tabla 5.13. CP-12 77
Tabla 5.14. CP-13 78
Tabla 5.15. CP-14 78
Tabla 5.16. CP-15 79
Tabla 5.17. CP-16 79
Tabla 5.18. CP-17 80
Tabla 5.19. CP-18 80
Tabla 5.20. CP-19 80
Tabla 5.21. CP-20 81

Tabla 5.22. CP-21 81
Tabla 5.23. CP-22 81
Tabla 5.24. CP-23 82
Tabla 7.1. Tareas del proyecto 87
Tabla 8.1. Salario medio de las posiciones de trabajo 90
Tabla 8.2. Costes salariales del proyecto 91
Tabla 8.3. Desglose de los costes materiales 91
Tabla 8.4. Presupuesto del proyecto 92
Table I Unity vs Unreal vs Godot 107
Table II Budget of the project 113

1. INTRODUCCIÓN

Este apartado introductorio trata de definir las motivaciones por las que se ha realizado el trabajo de fin de grado sobre esta temática en concreto, además de los objetivos que se busca conseguir mediante su desarrollo.

Asimismo, se incluye también un apartado en el que se explica la estructura que sigue este documento, ofreciendo una breve descripción de cada una de las partes que lo componen.

1.1. Motivación

En la actualidad, la industria de los videojuegos es el sector líder en el mercado del entretenimiento, con aproximadamente 3 mil millones de jugadores alrededor del mundo y generando ingresos superiores a los 180 mil millones de dólares en el año 2022 [1], [2], [3].

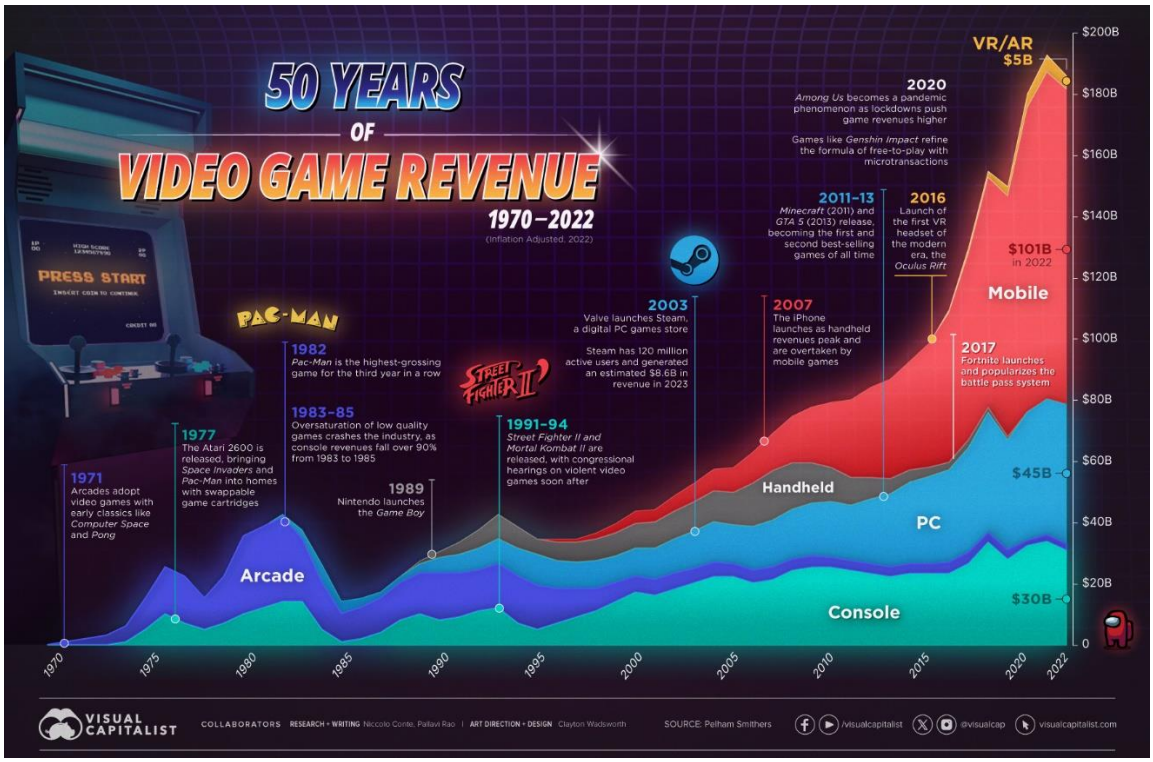

Fig. 1.1. 50 años de ingresos por videojuegos [1]

Como se puede observar en la figura 1.1, con datos ofrecidos por la firma anglosajona Pelham Smithers, los ingresos generados por la industria de los videojuegos han seguido un crecimiento exponencial, empezando a estancarse a partir del año 2022. No obstante, este estancamiento no ha supuesto que esta forma de entretenimiento pierda su liderazgo en el ámbito del ocio. En 2023, este sector generó unos ingresos globales de 184 mil millones de dólares, una subida del 0,6% respecto a 2022. Adicionalmente, se estima que para 2026, esta cifra supere los 205,4 mil millones de dólares [4].

Asimismo, no solo han crecido los ingresos de la industria, puesto que también han crecido el número de videojuegos que se lanzan cada año. Este número ha ido creciendo de manera constante, algo que se puede apreciar fácilmente en plataformas de venta de

videojuegos, como Steam, donde el número de videojuegos que se lanzó en 2018 sumó la cifra de 8.324 juegos. En comparación, en 2023 este número ascendió hasta los 13.971 lanzamientos. Además, aproximadamente el 98% de los videojuegos lanzados anualmente han sido creados por desarrolladores independientes, o indie [5]. Este porcentaje ha aumentado constantemente a lo largo de los años, donde la mayor parte de este aumento la protagonizan este tipo de lanzamientos, mientras que los lanzamientos de videojuegos de gran presupuesto se han mantenido constantes a lo largo de los últimos cinco años.

El crecimiento de lanzamientos *indie* también viene asociado a otro motivo, que es el aumento de motores de desarrollo de videojuegos que han ido surgiendo en la última década. Estos motores de desarrollo han dado la oportunidad de crear videojuegos a personas con un menor conocimiento técnico y experiencia en este sector. Esto se ve reflejado en que cada vez más desarrolladores utilizan estas herramientas para implementar sus videojuegos. Hoy en día, aproximadamente el 60% de los desarrolladores de videojuegos utiliza una de estas herramientas [6]. Esta razón es, principalmente, por lo que se ha decidido utilizar un motor de desarrollo de videojuegos para la ejecución de este proyecto. En concreto, se ha empleado Unity Engine. A lo largo de esta memoria, se habla en detalle sobre este motor de desarrollo de videojuegos.

En cuanto a los motivos por los que se ha decidido diseñar un videojuego del género *roguelike*, se presenta lo siguiente.

El primer motivo es el interés personal. Este género de videojuegos se ha popularizado en los últimos años gracias al auge de los estudios de desarrollo *indie*, quienes han desarrollado videojuegos que han marcado un nuevo estándar en la industria, como Supergiant Games, Motion Twin o Dodge Roll. Esta popularidad se puede observar también en el crecimiento del valor en el mercado de los juegos de este género, que en 2023 consiguieron generar unos ingresos de 23.153 millones de dólares y, según Valuates Reports, se proyecta que para 2030 esta cifra ascienda a los 57.336 millones de dólares [7].

Otro de los motivos es que la mayoría de los juegos del género *roguelike* no disponen de un modo multijugador, si no que se suelen centrar en un modo para un jugador, al menos en el caso de los mayores exponentes de los últimos años, como The Binding of Isaac, Dead Cells, Enter the Gungeon, Hades, etcétera. Esto ofrece la posibilidad de implementar algo relativamente innovador en este género, debido al bajo porcentaje de videojuegos que han implementado esta mecánica.

Por último, el hecho de realizar un juego multijugador permite afianzar y profundizar en conocimientos ya adquiridos durante la carrera, principalmente el conocimiento sobre redes de ordenadores, paso de mensajes, conexión cliente-servidor y sincronización entre equipos. Asimismo, la ejecución de este trabajo de fin de grado permite al alumno experimentar un proyecto de desarrollo de software desde sus comienzos, además de profundizar en las tecnologías que se pueden utilizar en el desarrollo de videojuegos

multijugador, así como las implicaciones que conlleva realizar un proyecto de esta índole.

1.2. Objetivos

Teniendo en cuenta las motivaciones detalladas en el apartado anterior, se pueden enumerar los objetivos que se pretende completar con la ejecución de este trabajo de fin de carrera. Estos objetivos se pueden separar en principales y secundarios, donde los primeros representan aquellos relacionados directamente con el tema del trabajo de fin de grado. Los secundarios representan aquellos que surgen de las motivaciones del alumno.

- Objetivos principales:
  - Diseño y desarrollo, desde cero, de un prototipo completamente funcional de un videojuego, con elementos del género *roguelike* y con un modo multijugador, en el que al menos dos jugadores puedan actuar simultáneamente a través de Internet.
- Objetivos secundarios:
  - Aprender a utilizar el motor de desarrollo Unity y sus herramientas, desde el punto de vista de una persona sin conocimiento previo sobre motores de videojuegos.
  - Aprender a programar en el lenguaje de programación C#, siguiendo una serie de buenas prácticas, reglas y convenciones de nombramiento y código, como las establecidas por Microsoft [8].
  - Reforzar y profundizar los conocimientos aprendidos en el grado de ingeniería informática, principalmente de asignaturas como redes de computadores, sistemas distribuidos, interfaces de usuario y tecnologías de desarrollo para la web.
  - Documentar adecuadamente el proceso creativo y de desarrollo de todo el trabajo de fin de grado mediante la elaboración de esta memoria.

1.3. Estructura del documento

En este epígrafe se detalla la estructura que sigue este documento, además de dar una breve descripción sobre cada uno de los capítulos y el contenido que se puede encontrar en el interior de cada uno de ellos.

1. Introducción. El capítulo actual. En este apartado preparatorio se introducen las motivaciones por las que se ha elegido este tema, además de exponer los objetivos que se busca efectuar con la elaboración de este proyecto final de carrera. Adicionalmente se incluye este epígrafe, en el que se detalla la estructura del documento.
2. Estado del arte. En esta segunda sección se expone la definición, historia y evolución de los videojuegos multijugador, el género *roguelike* y tecnologías relevantes a la elaboración de este proyecto, como los motores de desarrollo de videojuegos y el lenguaje C#. Asimismo, se estudian las alternativas, realizando una síntesis de las ventajas y desventajas que pueden ofrecer las mismas.

3. Análisis. Este capítulo explora el problema planteado, analizando en detalle los requisitos y casos de uso a tener en cuenta, para una correcta ejecución del proyecto, además de estudiar su trazabilidad. También presenta las tecnologías utilizadas y metodología de desarrollo seguida en la ejecución de este trabajo, además de las justificaciones de su elección.
4. Diseño. En esta sección se detalla el apartado técnico del proyecto, así como las decisiones de diseño tomadas durante su ejecución. Se exponen las principales funcionalidades y mecánicas desarrolladas además de los sistemas lógicos más importantes del proyecto.
5. Evaluación. Se corresponde con el plan de pruebas ejecutado para la verificación del correcto funcionamiento del prototipo implementado y los resultados obtenidos. Adicionalmente, se exponen posibles problemas y soluciones a estos.
6. Marco regulador. En él se detalla la legislación aplicable a un proyecto de esta índole. Asimismo, se detallan los estándares seguidos en el proceso y licencias aplicables a los programas y servicios utilizados.
7. Planificación. En este capítulo se presenta la planificación seguida durante el desarrollo de este trabajo, apoyándose en un diagrama de Gantt y presentando una estimación del tiempo que ha llevado cada proceso del trabajo de fin de grado.
8. Entorno socioeconómico. Sección en la que se detalla el presupuesto de la elaboración de este trabajo de fin de grado. Adicionalmente, en este apartado se detalla el impacto socioeconómico que tiene un proyecto de estas características.
9. Conclusiones. Objetivos cumplidos y justificación de cómo se han satisfecho. En adición, se especifican posibles ampliaciones que se pueden ejecutar sobre este proyecto.
10. Bibliografía. Esta sección contiene todas las referencias bibliográficas que se han utilizado a lo largo de este dosier.
11. Development of a roguelike multiplayer videogame. Resumen en inglés del documento.

2. ESTADO DEL ARTE

En este segundo capítulo se presentan y definen las principales bases teóricas sobre las que se construye este proyecto, así como las alternativas estudiadas y proyectos con fines similares.

2.1. Videojuegos y el modo multijugador

Los videojuegos multijugador son aquellos que permiten a dos o más jugadores interactuar en tiempo real. Esta interacción puede tener lugar en el mismo entorno, como un mismo ordenador o a través de una red, ya sea local o una red de área extensa, como Internet.

El primer proyecto que se considera como el primer videojuego multijugador de la historia es el título Tennis for Two, un juego creado por el físico nuclear William Higinbotham que fue introducido por primera vez en el año 1958. Tennis for Two permitía a dos usuarios jugar simultáneamente a un partido de tenis simulado mediante el uso de un ordenador analógico y utilizando un osciloscopio como pantalla [9].

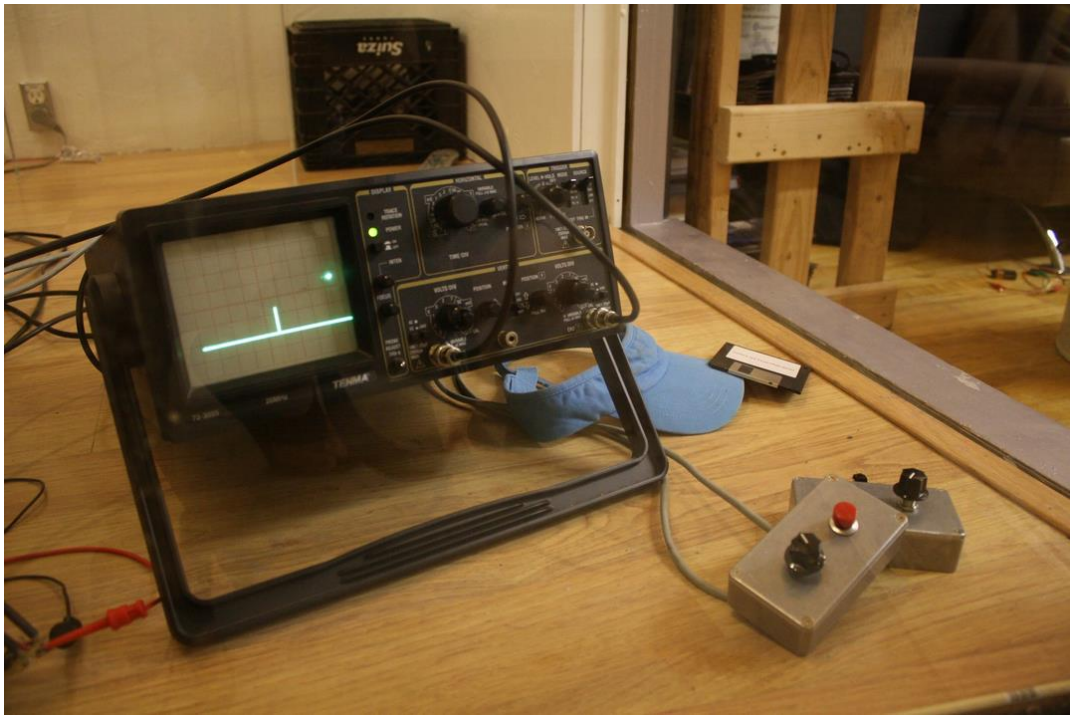

Fig. 2.1. Recreación del videojuego Tennis for Two [10]

Otros videojuegos que se consideran pioneros en este sector por diversas fuentes son el Pong (1972), Astro Race (1973), Spacewar! (1962). Este último se trata del primer videojuego desarrollado para un computador comercial, el Programmed Data Processor-1 [11], [12], [13], [14].

El mayor punto de inflexión en los juegos multijugador local se produjo en la década de los ochenta con la aparición de diversas consolas, pero en particular con la Nintendo Entertainment System, con juegos como Contra, Mario Bros. o Donkey Kong [15].

Cuando realmente se popularizaron los juegos multijugador fue en la década de los 90, con la aparición de juegos como Doom en 1993 y los juegos MMO, del inglés *Massively Multiplayer Online games*, que introdujeron la posibilidad de jugar en línea con una cantidad elevada de jugadores, como Ultima Online en 1997 [11], [16].

Alrededor del año 2000 nacieron juegos multijugador que han conseguido perdurar durante más de dos décadas, como es el caso de la saga Counter Strike, que apareció por primera vez en el año 1999 y 25 años después, sigue teniendo más de un millón de jugadores activos a diario [17].

A partir de aquí, la industria del videojuego ha seguido un curso estable. En la actualidad, se pueden observar todo tipo de juegos multijugador con una alta cantidad de jugadores activos, desde MMOs hasta juegos cooperativos de 2 a 4 jugadores simultáneos. Según SteamDB, una plataforma que recopila información de la plataforma de distribución de videojuegos para ordenadores llamada Steam, la cifra de usuarios simultáneos interactuando en algún juego multijugador oscila entre los 5 millones y 11 millones de usuarios, dependiendo del momento del día [18].

2.2. Topologías de red en el diseño de juegos multijugador

Una topología o arquitectura de red establece cómo se organiza una red de computadores. En este epígrafe se explican las diferentes topologías de red que normalmente se aplican en el diseño de videojuegos multijugador. Se procede a enumerarlas en orden de expansión, es decir, se empieza explicando topologías de área local y se termina con aquellas que permiten implementar un modo multijugador a través de una red de área amplia o WAN (*Wide Area Network*).

2.2.1. Topologías de área local

- Multijugador de sofá. Los multijugadores de sofá, de la expresión en inglés *couch multiplayer games*, son aquellos videojuegos que implementan el multijugador en una misma máquina, por ejemplo, dividiendo la pantalla en partes iguales con un punto de vista para cada jugador o usando una misma cámara para ambos individuos.

Fig. 2.2. Overcooked 2, juego que implementa el multijugador de sofá con un modo cooperativo [19]

- Multijugador local. Esta topología permite a varios usuarios desde distintas máquinas conectarse a una misma sesión dentro de un videojuego. Aunque el acceso a Internet no es necesario, puesto que lo único que necesitan los dos equipos es estar conectas a una misma red de área local (LAN), sí que utilizan una arquitectura Anfitrión-Cliente, de modo que una de las máquinas actuará como un servidor, como se verá en el apartado correspondiente.

2.2.2. Arquitecturas de área extensa

- Red punto a punto. Las redes punto a punto, también conocidas como P2P (*peer-to-peer*), son redes descentralizadas en las que varios computadores comparten sus recursos sin necesidad de que haya un servidor intermedio, o que una de las máquinas actúe como anfitrión, si no que cada uno de los ordenadores conectados a la red tiene el mismo nivel de autoridad [20]. En la figura 2.3 se puede observar un esquema de esta arquitectura.
  Estas topologías de red tienen una ventaja principal respecto a otras, y es que no necesitan un servidor intermedio al que conectarse, lo que abarata costes de manera significativa. No obstante, esto también se convierte en un inconveniente desde el punto de vista de la ciberseguridad, puesto que cualquier usuario de la red puede acceder a otros datos de cualquier equipo conectado a la red si no se hace una implementación correcta, además de ser vulnerable a diversos ciberataques [21].

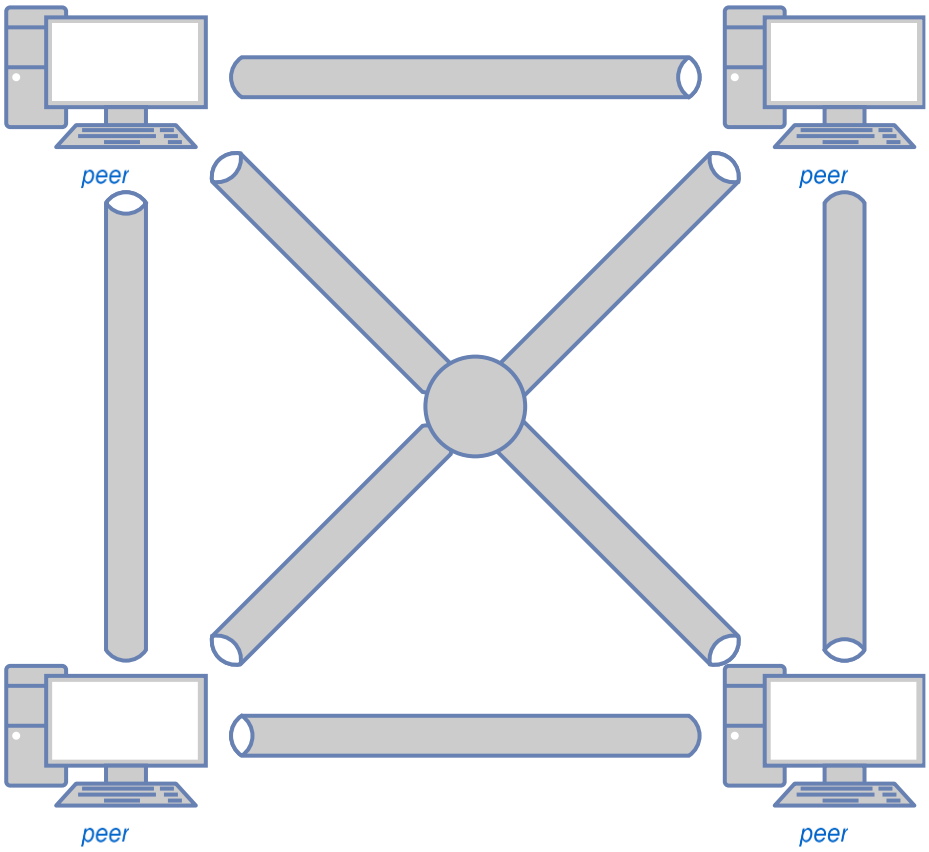

Fig. 2.3. Esquema de una red P2P

- Cliente-Servidor. Las arquitecturas de este tipo se componen de dos tipos de nodos. Por un lado, están los servidores, que son aquellos nodos que se encargan de conectar a los clientes entre sí, además de ocuparse de ejecutar todo el apartado lógico de los videojuegos. En el otro extremo están los clientes, quienes se conectan al servidor a través de peticiones o llamadas remotas para sincronizar los recursos con los otros componentes de la red. La figura 2.4 representa esta arquitectura de red.

  Estas topologías tienen dos desventajas principales. La primera es el coste adicional que supone mantener un servidor dedicado. Otra desventaja que tiene es el aumento de latencia a la hora de sincronizar cambios a través de clientes, ya que todos los datos tienen que pasar previamente por el servidor para ser validados.

  En cuanto a las ventajas que ofrece, se pueden encontrar por una parte el rendimiento, pues al usar un servidor para toda la lógica, los clientes pueden gozar de un mayor rendimiento general, al tener una menor carga de trabajo en esas máquinas. El uso de un servidor dedicado también tiene una ventaja significativa desde el punto de vista de la seguridad; al verificarse toda la lógica en el servidor, se puede evitar el uso de modificaciones ilegales por parte de los clientes que afecten al estado del juego o den ventajas significativas a uno de los jugadores en concreto. Esta última ventaja es la razón esencial por la que todos los juegos de la escena competitiva, también denominados *esports*, utilizan esta arquitectura de red [22], [23].

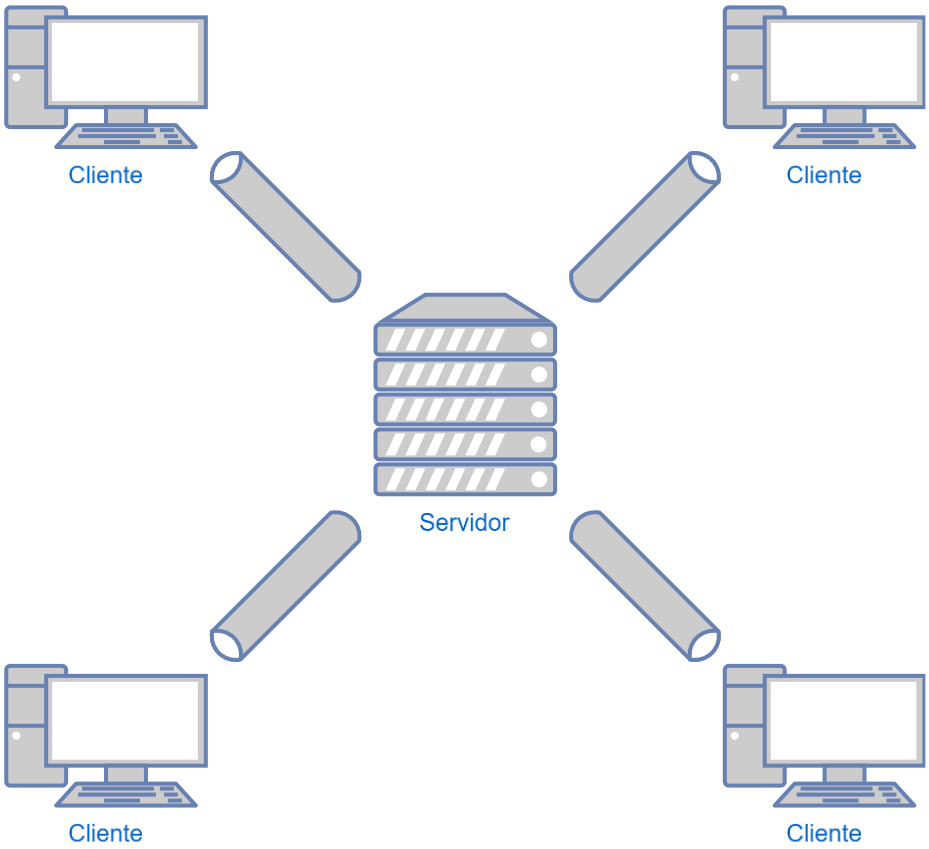

Fig. 2.4. Arquitectura Cliente-Servidor

- Cliente-Anfitrión. A diferencia de las redes de la topología Cliente-Servidor, en esta arquitectura no existe un servidor dedicado, si no que uno de los clientes actúa como cliente y servidor, como anfitrión. Por una parte, en la máquina del anfitrión se inicia un servidor de escucha. Este servidor de escucha tiene la función, al igual que en la topología anterior, de procesar las peticiones de los clientes conectados a la red y de ejecutar toda la lógica del programa. Asimismo, el anfitrión también ejecuta el código que procesa cualquier otro cliente [22]. En la figura 2.5 se puede apreciar el esquema de esta topología.

  Esta topología tiene una ventaja fundamental, siendo esta que se abaratan los costes al no necesitar un servidor dedicado, lo que permite centrar los costes de desarrollo en otras áreas del proyecto. Además, no necesitar un servidor significa que no es necesario utilizar ningún tipo de infraestructura especializada, por lo que agiliza el despliegue y permite al videojuego funcionar independientemente de que un nodo raíz, como es un servidor, esté en ejecución. Adicionalmente, es ideal para videojuegos donde no se requiere un mundo persistente y el número de jugadores en una misma sesión de juego es menor de 12 jugadores [24].

  No obstante, esta arquitectura presenta varios problemas:

  - La máquina del anfitrión tiene una mayor carga de trabajo, por lo que el rendimiento puede verse degradado si este equipo no posee unos componentes que cumplan unos requisitos mínimos, lo que puede afectar no solo al anfitrión, sino también a los clientes.

- El rendimiento de la red depende de la conexión a Internet del anfitrión, siendo esta en la mayoría de los casos residencial. Esto se puede traducir a un aumento de la latencia o tiempo de respuesta en el caso de que el comportamiento de la red no sea estable.
- Al ejecutar el anfitrión toda la lógica y tener acceso al servidor, es más fácil para el anfitrión modificar el estado de la partida, haciendo que pueda llegar a ser injusto el juego para los clientes en el caso de que se presente esta casuística. Por esta razón, esta arquitectura es inadecuada por norma general en juegos que tienen un aspecto competitivo.
- En el caso de que el anfitrión, es decir, el servidor, deje de funcionar o salga del juego, todos los clientes perderán la conexión a esa sesión de juego.

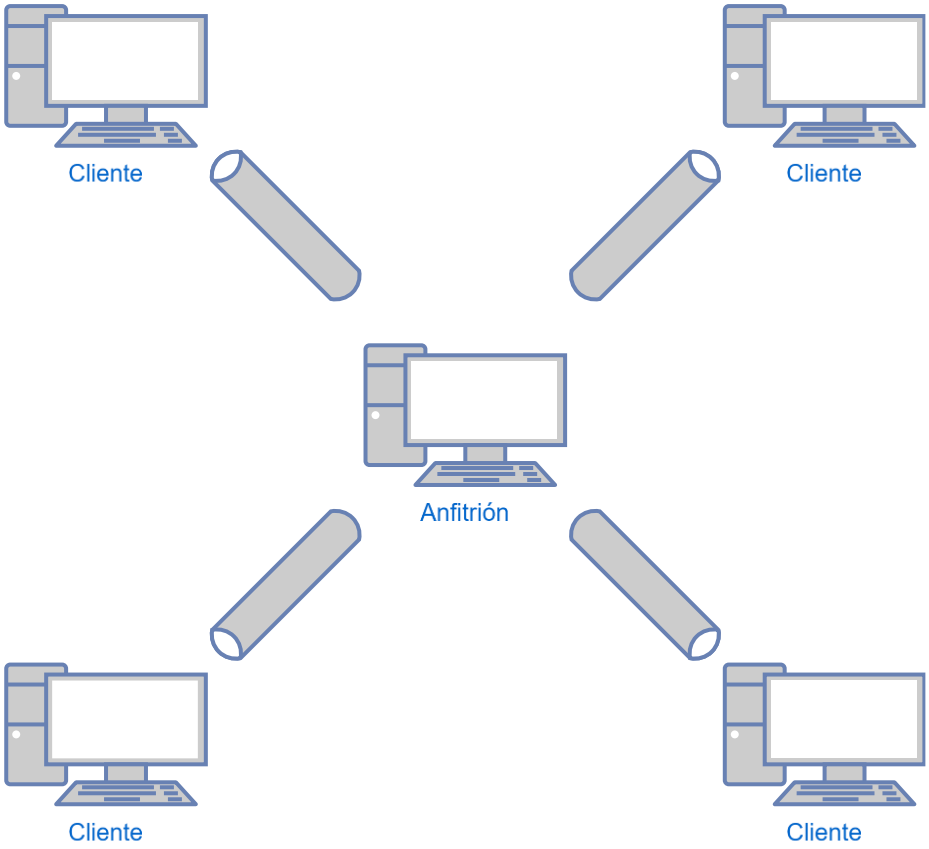

Fig. 2.5. Topología Cliente-Anfitrión

- Red de autoridad distribuida. A diferencia de la topología Cliente-Servidor y su derivada, donde el servidor se encarga de toda la lógica del sistema y es propietario de todos los objetos del entorno, siendo este el único responsable de instanciar nuevos objetos, esta topología propone que los clientes sean capaces de gestionar objetos ellos mismos. Asimismo, al no existir un servidor dedicado que se encargue de procesar la lógica del sistema, cada uno de los clientes tiene que realizar una simulación parcial del estado del juego, sincronizando los cambios al resto de clientes. Sin embargo, aunque no existe un servidor dedicado, sí que es necesario que haya un servicio central que se encargue de realizar un seguimiento de los cambios y enrute el tráfico de red. Esto es

principalmente lo que diferencia esta topología de las redes punto a punto [25]. En la figura 2.6 se muestra esta topología de red, donde se pueden apreciar las diferencias de esta topología con las anteriores.

Esta arquitectura tiene una ventaja principal respecto a la topología Cliente-Anfitrión y se trata del concepto de propietario de la sesión. En estas redes, se asigna a uno de los clientes como propietario de la sesión, teniendo permisos similares a los de un anfitrión, siendo el encargado de gestionar tareas que tienen que ver con el estado general del juego. No obstante, siempre y cuando el estado global se sincronice entre todos los clientes de manera adecuada, si el propietario abandona la sesión, uno de los clientes será designado como propietario de la sesión, pasando a encargarse del estado global de la partida. Otras ventajas que incluye esta arquitectura son unos mejores tiempos de respuesta, al no haber un intermediario que se encargue de validar la lógica y una menor carga de trabajo, al distribuirse toda la carga entre todos los clientes conectados.

Las desventajas notables que presenta esta arquitectura son principalmente dos. La primera de ellas se trata de juegos en los que se involucre un alto uso de físicas o la precisión de estas sea importante. Al no haber una sola unidad encargándose de toda la simulación de las físicas, es necesario tomar un enfoque distinto a la hora de implementar la interacción entre objetos. La segunda desventaja que tiene es que es relativamente fácil modificar el estado del juego por parte de los clientes, por lo que el uso de esta arquitectura en juegos de carácter competitivo no está recomendado.

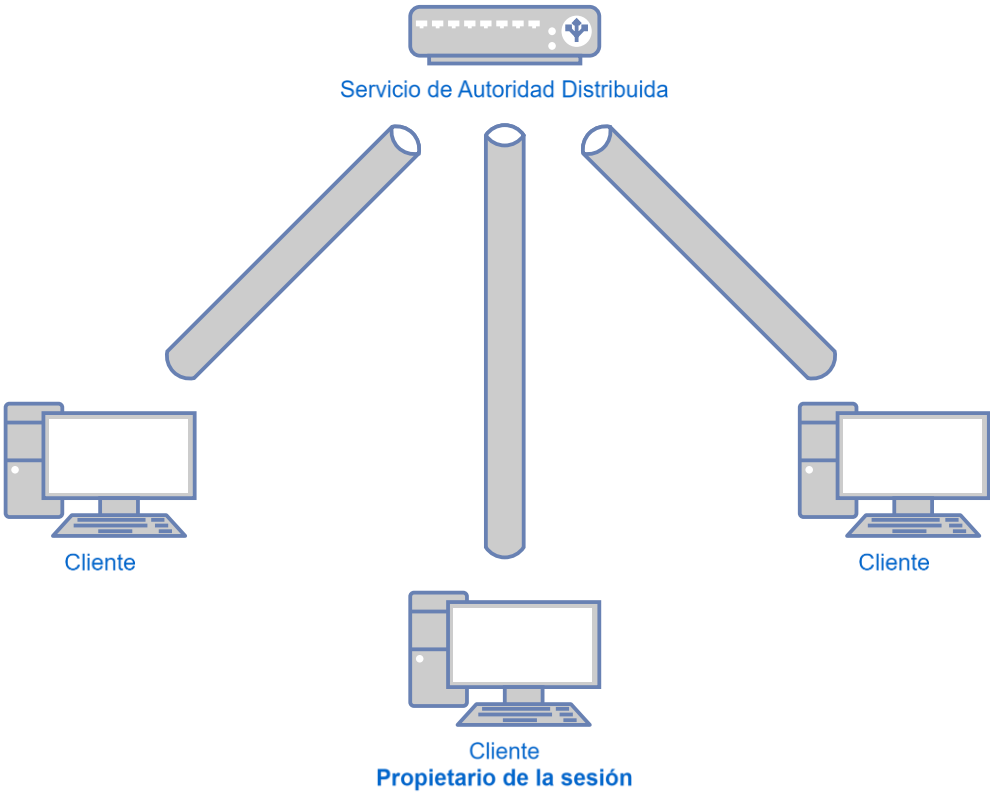

Fig. 2.6. Arquitectura de autoridad distribuida

2.3. Desafíos en el desarrollo de videojuegos multijugador en línea

A la hora de desarrollar videojuegos multijugador, uno de los principales problemas que se puede encontrar es la manera de conectar usuarios a través de Internet. En el caso de arquitecturas que utilizan una entidad centralizada, como las topologías Cliente-Servidor, esto no suele suponer un problema, ya que los servidores no suelen cambiar de direccionamiento. Adicionalmente, suelen tener un rango de puertos dedicados a ofrecer los servicios del videojuego, además de tener cortafuegos y otros mecanismos para evitar ciberataques. Sin embargo, en una topología Cliente-Anfitrión, todos los clientes (incluyendo el anfitrión) se conectan habitualmente desde una red doméstica. Esto presenta un problema, puesto que, salvo en raras excepciones, en estas redes se realiza una traducción de la dirección de red, conocido comúnmente como NAT (*Network Address Translation*). En redes domésticas, se utiliza la traducción de direcciones de red para enmascarar la dirección IP (*Internet Protocol*) privada de un usuario de la red doméstica, protegiéndolo del exterior y además permitiendo que el espacio de direcciones IPv4 no se sature. Esto se traduce a que, por defecto, todos los dispositivos de una red doméstica están asociados a una única IP pública, por lo que tiene que haber una manera de identificarlos desde el exterior, para que el rúter pueda enrutar el tráfico hacia el dispositivo de la red correcto [26].

Para que un cliente pueda conectarse a un anfitrión protegido por un NAT, es necesario encontrar un método que permita identificar al anfitrión desde una red externa a la de este último. En la actualidad, existen tres maneras de solventar esta circunstancia [24]:

- Redirección de puertos. Esta técnica consiste en abrir un puerto público en el rúter del anfitrión, redireccionando el tráfico que llega a ese puerto a la máquina concreta del anfitrión. Sin embargo, esta solución tiene varios inconvenientes. El primero de ellos es que esta solución se tiene que hacer de manera local. Es decir, para que un cliente se pueda unir a la sesión de este anfitrión, este último tiene que abrir un puerto desde su rúter o cortafuegos, por lo que es necesario que el usuario en cuestión tenga ciertos conocimientos técnicos. Además, puede ocurrir que esté en una situación en la que no pueda realizar esta acción, por ejemplo, si el anfitrión está utilizando un dispositivo móvil o una red pública. El otro inconveniente que presenta, son los riesgos de ciberseguridad que conlleva abrir un puerto, ya que esto permite a un atacante tener una ruta directa hacia el sistema del anfitrión.
- Servidor de retransmisión. Un servidor de retransmisión es un servidor dedicado en la nube o en un centro de datos, que tiene preparada previamente una redirección de puertos, lo que permite a los usuarios conectarse a ellos directamente. El servidor de retransmisión actúa como un intermediario entre el anfitrión y los clientes, donde los clientes envían paquetes al servidor y este se encarga de que lleguen al destino deseado. Esta opción, en comparación con la anterior, siempre está disponible para los clientes y el anfitrión, además de poder notificar a los clientes si el anfitrión se desconecta de la red o incluso iniciar un proceso de migración. Las desventajas que presenta esta solución es que tienen

un coste añadido al utilizar una infraestructura especializada, además de que la latencia entre el anfitrión y los clientes puede ser mayor, puesto que para realizar la comunicación tienen que pasar previamente por el servidor de retransmisión.

- NAT *punch-through*. Esta técnica, que en castellano se podría traducir perforación de NAT, intenta abrir una conexión directa entre cliente y servidor sin hacer uso de una redirección de puertos. En caso de que esta acción resulte exitosa, los clientes estarán directamente conectados. No obstante, debido al tipo de traducción de direcciones de red que suele tener lugar entre los clientes, este proceso suele fallar. Por esto mismo, esta técnica se suele implementar con una alternativa, que en la mayoría de los casos consiste en usar un servidor de retransmisión en el caso de que la perforación de NAT no de resultado. En caso de que los clientes puedan realizar la conexión mediante la perforación de NAT, no se realizará la conexión a través del servidor de retransmisión, lo que abarata costes y carga de trabajo por parte de este último.

2.4. El género *roguelike*

El término *roguelike* es un subgénero de videojuegos que recibe su nombre directamente del videojuego Rogue, lanzado en 1980. Este término surgió por primera vez en 1993, a raíz de una discusión en USENET entre usuarios que buscaban un nombre para videojuegos que tenían características parecidas a Rogue, como Adventure, Zork, Moria o Angband. Tras diversas discusiones, la comunidad llegó al consenso de un nombre para este tipo de juegos, que se denominarían como videojuegos del género *roguelike* (*rogue-like*, en castellano, "como Rogue") [27], [28]. Según estas discusiones, el género se caracterizaba por estas particularidades [29]:

- El jugador controla un único personaje.
- Son juegos de rol de fantasía, que generalmente tienen lugar en una mazmorra.
- Poseen una interfaz gráfica sencilla, basada en caracteres ASCII.
- El objetivo del juego es volverse más fuerte, con el propósito de completar una misión difícil.
- Son juegos distribuidos de manera gratuita.

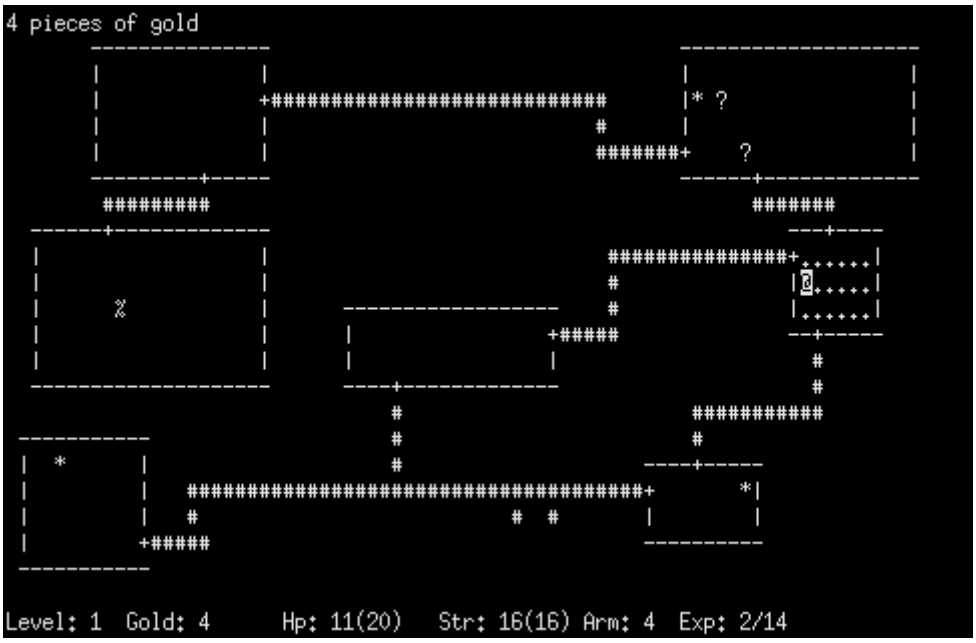

Fig. 2.7. Captura de pantalla de Rogue en su versión de 1980 [30]

Hoy en día, esta definición ha evolucionado, eliminando algunas de las restricciones que presentaba, como que los juegos para pertenecer a este género tienen que ser de distribución gratuita, o que la interfaz tiene que ser simple, basada en caracteres o mosaicos. En la actualidad, un juego se puede considerar, por lo general, como un *roguelike* si cumple las siguientes características [31], [32]:

- Generación procedimental. La generación del entorno, enemigos y demás elementos de la escena es pseudoaleatoria, por lo que cada partida es distinta a la anterior.
- Jugabilidad por turnos. El transcurso del videojuego es parecido al de un juego de mesa, donde el jugador tiene un tiempo infinito para pensar una estrategia con la que poder completar el juego.
- El jugador controla a un solo personaje. Al igual que en la definición clásica, el jugador únicamente manejará a un individuo durante todo el transcurso de la partida.
- Consecuencias permanentes. Cualquier acción del jugador tiene consecuencias permanentes e inmutables. Esto incluye la muerte del personaje del jugador.
- Jugabilidad basada en nuevas partidas o *runs*. La muerte permanente del jugador implica que cada partida empezará desde el principio, perdiendo cualquier progreso que este haya hecho hacia la meta. Lo único que se conserva es la propia experiencia del jugador, conseguida de partidas anteriores.

Esta definición, bajo el contexto de los videojuegos actuales de este género, es la más popular. No obstante, al no ser un término que se haya formalizado, está en constante cambio. Del mismo modo, existen otras definiciones que han surgido de diferentes discusiones o eventos, como en la International Roguelike Development Conference, en Berlín, Alemania, en el año 2008 [33].

También es necesario mencionar un subgénero del anterior, el término *roguelite*. Este término se utiliza hoy día en aquellos videojuegos cuyas bases y características son las propias de los videojuegos *roguelike*, pero que incorporan cambios críticos a su definición, como puede ser algún tipo de progreso, por ejemplo, con el desbloqueo de nuevas armas, habilidades, o mejora de las estadísticas base del jugador [32].

2.4.1. Juegos del género *roguelike* similares a la solución propuesta

A continuación, se van a listar algunos de los videojuegos del género *roguelike* que ofrecen características similares al desarrollado en este proyecto.

Enter the Gungeon.

Enter the Gungeon es un videojuego *roguelike* de disparos en el que el jugador, que controla a una serie de personajes marginados y arrepentidos con su pasado, tiene que superar los diversos desafíos impuestos por la Armazmorra, con el objetivo de hacerse con el tesoro que aguarda en esta: el arma que puede matar al pasado [34].

Este videojuego, desarrollado por el estudio independiente Dodge Roll y lanzado en 2016, es uno de los videojuegos *roguelike* mejor valorados de la última década, con un 95% de reseñas positivas en Steam y un 82 sobre 100 en Metacritic [35], [36]. En enero de 2020, Dodge Roll anunció que había vendido más de tres millones de copias juntando las ventas de todas las plataformas [37]. Este juego, al igual que el prototipo desarrollado en este proyecto se creó con el motor Unity.

En el prototipo desarrollado se pueden ver muchas semejanzas a este juego, principalmente porque este ha sido el primero que se ha utilizado como referencia. Algunas de las similitudes que se pueden observar son las siguientes:

- El estilo artístico utilizado es un estilo bidimensional basado en la forma de arte digital *pixel art*. Es utilizado principalmente por su simpleza
- Ambos juegos utilizan un punto de vista cenital.
- Tanto en Enter the Gungeon como en el prototipo desarrollado, el jugador controla a un personaje que puede disparar con armas a distancia a sus enemigos.
- En ambos juegos el escenario es una mazmorra de la que el jugador tiene que escapar.

Fig. 2.8. Captura de pantalla de Enter the Gungeon

No obstante, Enter the Gungeon no posee un modo multijugador, siendo esta la principal diferencia entre este juego y el prototipo desarrollado a lo largo de este trabajo.

The Binding of Isaac.

The Binding of Isaac es otro *roguelike* que salió por primera vez al mercado en el año 2011, desarrollado por Edmund McMillen y Florian Himsl [38]. Este juego está inspirado fundamentalmente por la historia del Sacrificio de Isaac, recogida en la Biblia, como se puede deducir por su nombre o algunas de las referencias que contiene el videojuego.

Este videojuego fue muy bien recibido por el público, teniendo una media de un 91% de valoraciones positivas en Steam. En 2014, Edmund McMillen, junto a la compañía Nicalis Inc., realizaron una adaptación del juego original, The Binding of Isaac: Rebirth, esta versión, que incluye diversas mejoras respecto al juego original, es considerada por muchos críticos como uno de los mejores juegos *roguelike* de la historia, teniendo un 96,45% de reseñas positivas en Steam [39]. Además, entre ambas versiones, se estima que se han vendido alrededor de once millones de copias en todo el mundo [40].

Este juego, al igual que el prototipo, tiene una vista cenital y un estilo en dos dimensiones, donde el jugador controla a un personaje que dispara proyectiles (en este caso lágrimas).

Cabe destacar que en 2017 el juego recibió la expansión The Binding of Isaac: Afterbirth+, que incorpora la posibilidad de jugar en multijugador. No obstante, solo implementa la arquitectura de multijugador de sofá, aunque gracias al servicio de Steam Remote Play, es posible jugar con otros jugadores a través de Internet, pero con el prerrequisito de tener una cuenta en esta plataforma [41].

Nuclear Throne

De estos tres juegos, Nuclear Throne es el menos conocido de ellos, habiendo vendido alrededor de 680.000 copias [42]. No obstante, este juego tiene un 96% de reseñas positivas en Steam y un 89 sobre 100 en Metacritic [43], [44].

Al igual que los otros dos juegos, se trata de un *roguelike* de disparos en vista cenital y en 2D, con un estilo artístico simple y desarrollado por un estudio independiente, Vlambeer, compuesto por una sola persona [45]. Este juego, como ocurría con The Binding of Isaac, tiene un modo de juego multijugador local y la posibilidad de jugar con otra persona en línea a través de la solución Steam Remote Play.

Fig. 2.9. Captura de pantalla de Nuclear Throne

2.5. Motores de desarrollo de videojuegos

Los motores de desarrollo de videojuegos son herramientas especializadas que ayudan a los diseñadores y programadores de videojuegos a crear este tipo de aplicaciones, formando un entorno de desarrollo que facilite y optimice todo el proceso de creación y mantenimiento del videojuego. Asimismo, los motores de videojuegos proporcionan un motor gráfico con el que poder renderizar los gráficos del programa, además de uno o más motores físicos que permitan emular las leyes de la física.

Aunque su uso está recomendado por la facilidad y organización que aportan, no son indispensables en la creación de videojuegos, pudiendo crear cada uno de los componentes necesarios o usando aplicaciones que implementen las funcionalidades necesitadas. Aun así, hoy en día su uso se ha normalizado. Según un estudio realizado por SlashData sobre una muestra de 2.325 desarrolladores de videojuegos, en la actualidad un 60% de ellos utiliza un motor de videojuegos [6].

A continuación, se va a hablar de algunos de estos motores de desarrollo. Se incorpora un breve resumen de cada uno de ellos y una tabla comparativa entre sus funcionalidades.

Unity Engine

Unity Engine es un motor de desarrollo 3D en tiempo real que permite tanto a desarrolladores como diseñadores crear videojuegos, además de otros tipos de contenidos animados, como simulaciones. Soporta una gran variedad de plataformas, tanto de sobremesa, dispositivos portables y otras plataformas, como las de realidad virtual, como se puede ver en la tabla 2.1 que se detalla a continuación:

TABLA 2.1.
PLATAFORMAS OBJETIVO DE UNITY [46]

Plataforma objetivo	**Sistema Operativo/Plataforma**
Escritorio	Microsoft Windows Mac Linux Universal Windows Platform
Móviles	iOS Android
Realidad extendida	ARKit ARCore Microsoft HoloLens Windows Mixed Reality Magic Leap Oculus PlayStation VR
Consolas	PlayStation 4 y PlayStation 5 Xbox One y Xbox Series X|S Nintendo Switch Google Stadia
Plataformas embebidas	Embedded Linux QNX
Web	WebGL
TV	tvOS Android TV

Unity Engine, creado por la compañía Over The Edge Entertainment (OTEE), se lanzó por primera vez al mercado en 2005 como un motor de videojuegos para Mac OS X, con el objetivo de crear un motor gráfico que estuviese disponible en varias plataformas, fuese asequible y fácil de utilizar. En 2005 se lanzó el primer videojuego creado en esta plataforma, GooBall, creado por OTEE. Este juego no acabó siendo rentable, pero permitió a OTEE contratar más personal para seguir desarrollando Unity Engine. A partir de aquí Unity fue ganando popularidad, primero en juegos para navegadores y dispositivos móviles, hasta la actualidad, donde es el motor de videojuegos líder del mercado [47].

Algunos ejemplos de títulos populares que se han desarrollado con este motor son Beat Saber (2019), Among Us (2018), Subnautica (2018), Hollow Knight (2017), Rust (2018), Pokémon GO (2016) o Genshin Impact (2020) [48].

Fig. 2.10. El juego de VR Beat Saber, desarrollado en Unity [49]

Unreal Engine

Unreal Engine es un motor de desarrollo de videojuegos 3D, lanzado por la compañía Epic Games en 1998. El primer juego que se desarrolló con este motor gráfico fue el juego Unreal, en el mismo año de su lanzamiento y por la misma compañía que creó el motor gráfico. Este juego fue uno de los primeros en enfatizar la atención al detalle y texturas de mayor resolución a las del resto de videojuegos de su época. Unreal Engine ha seguido evolucionando hasta su última versión principal, Unreal Engine 5, que se utiliza tanto en el desarrollo de videojuegos de todos los presupuestos, simulación y en la cinematografía [50], [51].

Algunos de los juegos más conocidos de los últimos años desarrollados con Unreal Engine son: Fortnite, Senua’s Saga: Hellblade II, Astroneer, Assetto Corsa Competizione, y Ark: Survival Evolved.

Godot Engine

Godot es un motor de videojuegos de código abierto, creado por Juan Linietsky y Ariel Manzur. En 2001 se empezó a desarrollar este motor, siendo un proyecto privativo creado por Juan, Ariel y la empresa en la que trabajaban, OKAM Studio, pero en 2014 se liberó su código bajo la licencia MIT, año en el que además salió su primera versión estable (Godot 1.0). Alrededor de 2016, cuando su versión 2.0 fue lanzada, el proyecto empezó a conseguir popularidad, lo que hizo que cada vez más personas contribuyesen en su desarrollo, agilizándolo y consiguiendo que hoy en día sea uno de los motores de videojuegos más utilizados [52], [53]. Actualmente, el repositorio de GitHub tiene más de 84.400 estrellas y en él han contribuido más de 2.500 personas [54].

Algunos juegos populares creados con Godot Engine son: Brotato (2022), Dome Keeper (2022), Buckshot Roulette (2024) y Casette Beasts (2023).

2.5.1. Comparativa de los tres motores de videojuegos

A continuación, se presenta una tabla comparativa de las características de los tres motores gráficos mencionados, la tabla 2.2:

TABLA 2.2.
UNITY VS UNREAL VS GODOT

Característica	**Unity Engine**	**Unreal Engine**	**Godot Engine**
Gráficos objetivo	3D y 2D	Solo admite 3D	3D y 2D
Lanzamiento	2005	1998	2014
Precio	Gratis para uso personal, siempre y cuando no se generen unos ingresos superiores a los 100.000 dólares. Planes de pago, dependiendo del sector y tamaño de los equipos de desarrollo.	Gratis para particulares y pequeñas empresas (con menos de 1 millón de dólares en ingresos anuales). Basado en regalías o en puestos si se genera más de 1 millón de dólares en ingresos.	Gratuito (licencia MIT)
Plataformas objetivo	Más de 24 plataformas distintas [46]	17 plataformas [55]	6 plataformas [56]
Programación	C# Unity Visual Scripting	C++ Visual scripting con Blueprints [57]	GDScript [58] C# C++ Visual Scripting Con extensiones de la comunidad: Rust, Nim, Python y JavaScript

2.6. Conclusiones del estado de la cuestión

En base a los apartados anteriores estudiados, se argumentan en este epígrafe las decisiones finales tomadas, que llevarán al posterior análisis y diseño de la aplicación implementada.

En cuanto a la arquitectura de red utilizada, se ha decidido emplear una topología cliente-anfitrión. Los motivos por los que se ha usado esta topología son los que se exponen a continuación:

- En cuanto a costes, es más barata que las infraestructuras que necesitan un servidor dedicado o un servicio central en la infraestructura, aunque por lo general es más cara que una red P2P.
- Ofrece un buen balance entre seguridad y rendimiento de la red, aunque la mayoría de la red dependa de un único individuo, el anfitrión, generalmente de origen doméstico.
- Su complejidad es relativamente más baja que en el resto de las arquitecturas, dado que no se necesita gestionar un servidor o un servicio centralizado como en una red de autoridad distribuida. Además, esta última requiere de un enfoque distinto a la hora de implementar las distintas funcionalidades de un proyecto. Por ejemplo, puede llegar a ser necesario desarrollar nuevos sistemas de físicas.

Dado que esta arquitectura de red es la que se emplea en el desarrollo de este trabajo, en el capítulo de diseño se explica en detalle cómo funcionan en Unity algunos temas importantes en el contexto de este proyecto, como las diferentes formas de implementar el multijugador en este motor, la conexión de nuevos clientes a una red de este tipo, o cómo funciona la autoridad y apropiación de objetos de la red.

Para solucionar los problemas que conlleva utilizar una topología cliente-anfitrión, como es la conexión de usuarios a través de Internet, se ha decidido utilizar la opción de un servidor de retransmisión, puesto que Unity ofrece el servicio Unity Relay, un servicio que permite a los desarrolladores utilizar un servidor de retransmisión de manera gratuita, siempre y cuando no se alcance una cuota de uso establecida por el plan mensual gratuito de Unity Gaming Services, como se verá más adelante en el capítulo del marco legal [59]. Como este proyecto durante su desarrollo no va a sobrepasar esa cuota, se ha decidido optar por su utilización.

Como se puede apreciar en las motivaciones del proyecto y en los dos párrafos anteriores, Unity Engine ha sido el motor de desarrollo que se ha decidido utilizar para implementar el prototipo de este proyecto. Las razones por las que se ha decidido utilizar esta plataforma son las que se detallan a continuación.

En primer lugar, Unity permite a los diseñadores y desarrolladores lanzar sus productos en más de 25 plataformas [46]. Esto permite al equipo de desarrollo realizar juegos multiplataforma de una manera sencilla y eficaz.

Otra de las motivaciones que han llevado a escoger esta plataforma de desarrollo es que incorpora un plan personal gratuito, por lo que no es necesario pagar una licencia para poder empezar a desarrollar en este motor.

Además, según un estudio realizado por SlashData en 2017, Unity es el motor de videojuegos más popular entre los desarrolladores, teniendo una cuota de mercado del 38% [6]. Esta popularidad se traduce en una mayor documentación disponible en Internet, además de la extensa documentación de Unity, como su manual de usuario y API de *scripting* [60].

La última razón por la que se ha decidido utilizar Unity es porque no se ha utilizado previamente en el grado universitario, lo que proporciona un componente de aprendizaje nuevo y un reto, al tener que aprender a utilizar una nueva herramienta profesional desde cero y aprender un nuevo lenguaje de programación, ya que Unity utiliza como lenguaje de programación C#, un lenguaje multiparadigma de código abierto desarrollado por Microsoft [61].

3. ANÁLISIS

En esta sección se definen los requisitos y casos de uso a partir de los cuales se ha realizado el diseño de este proyecto, además de la trazabilidad de los anteriores. Asimismo, se justifica la metodología de desarrollo utilizada y se detallan el conjunto de tecnologías empleadas.

3.1. Casos de uso

Los casos de uso de este proyecto se han desarrollado en base a tres escenarios o flujos distintos, cada uno con su diagrama correspondiente:

- Escenario 1 - Flujo de la partida. Manifiesta el curso de una partida, representando las acciones que puede tomar el jugador, así como las acciones que pueden ejecutar los enemigos que aparecen por la escena. Como las acciones que pueden realizar todos los jugadores son las mismas, se ha representado dicho escenario con un solo jugador.
- Escenario 2 - Escenario de conexión para el modo un jugador. Representa el flujo de conexión desde que se inicia el juego hasta que se inicia una partida en el modo de un jugador.
- Escenario 3 - Flujo de conexión para el modo multijugador. Expresa el proceso de conexión del cliente y el anfitrión para crear una partida del modo multijugador, hasta el momento en el que se inicia dicha partida.

3.1.1. Diagramas de los casos de uso

En este epígrafe se muestran los diagramas de casos de uso de cada uno de los escenarios planteados. Posteriormente, se listan los casos de uso nuevos que aparecen en cada uno de estos diagramas, dándoles un identificador como el siguiente:

CUXX

Donde cada CU (Caso de Uso) tiene asociado un número identificativo XX.

3.1.1.1. Diagrama del escenario 1

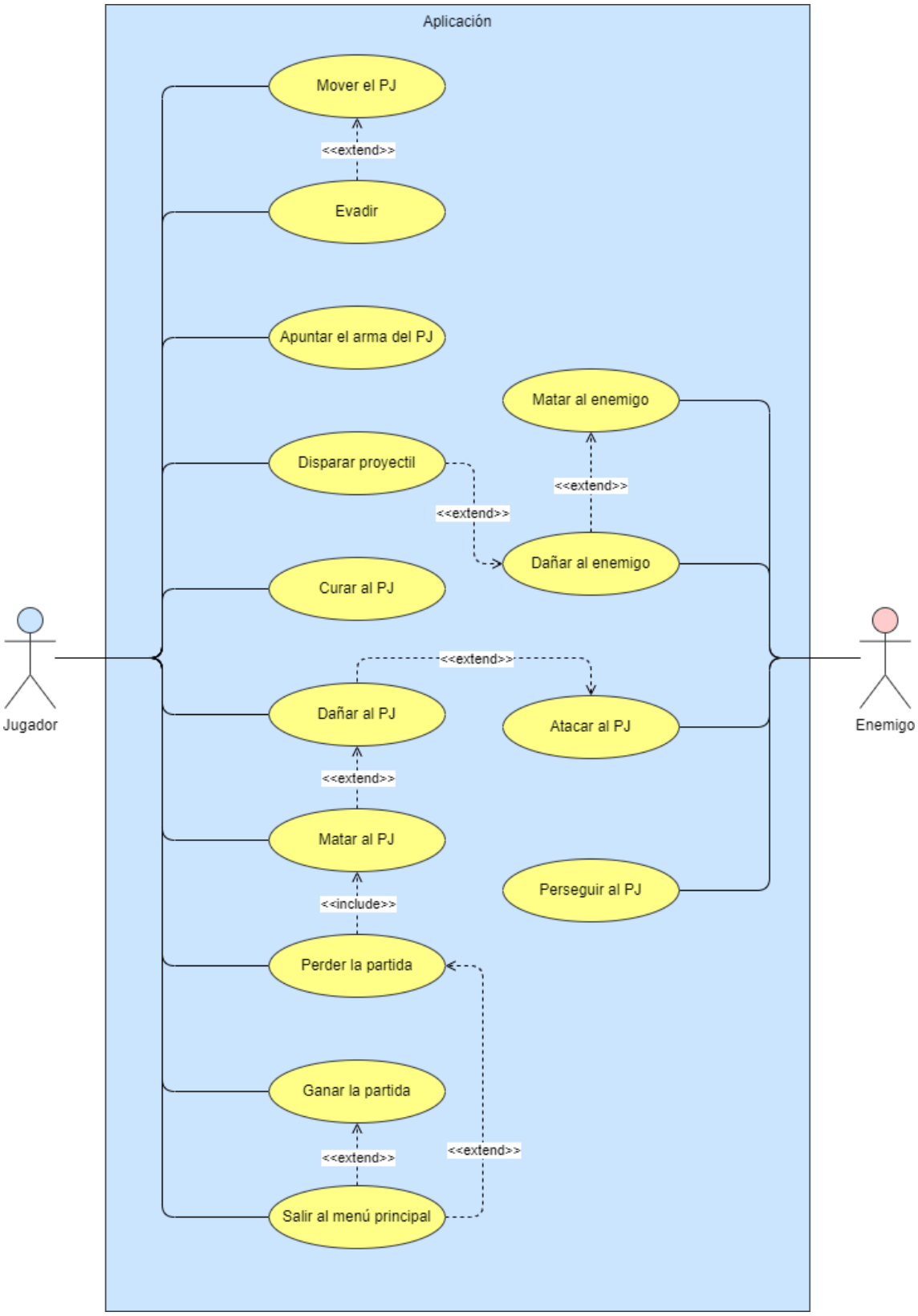

Fig. 3.1. Diagrama de casos de uso del escenario 1

En este primer diagrama, aparecen los siguientes casos de uso:

TABLA 3.1.
CASOS DE USO DEL ESCENARIO 1

Identificador	**Nombre del caso de uso**
CU01	Mover el PJ
CU02	Evadir
CU03	Apuntar el arma del PJ
CU04	Disparar proyectil
CU05	Curar al PJ
CU06	Dañar al PJ
CU07	Matar al PJ
CU08	Perder la partida
CU09	Ganar la partida
CU10	Salir al menú principal
CU11	Matar al enemigo
CU12	Dañar al enemigo
CU13	Atacar al PJ
CU14	Perseguir al PJ

Nótese que, en estos casos de uso, PJ significa Personaje Jugable, es decir, el personaje que maneja el jugador.

3.1.1.2. Diagrama del escenario 2

El diagrama para el segundo escenario es el siguiente, del que surgen dos nuevos casos de uso, como se puede apreciar en la tabla que se encuentra inmediatamente después de la figura 3.2, que representa el diagrama del segundo escenario.

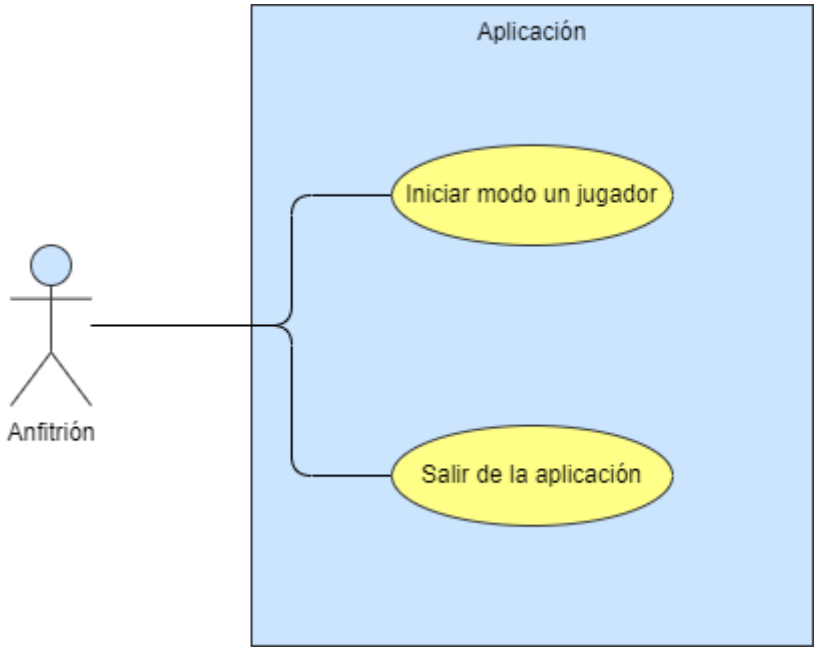

Fig. 3.2. Diagrama de casos de uso del escenario 2

TABLA 3.2.
CASOS DE USO DEL ESCENARIO 2

Identificador	**Nombre del caso de uso**
CU15	Iniciar modo un jugador
CU16	Salir de la aplicación

3.1.1.3. Diagrama del escenario 3

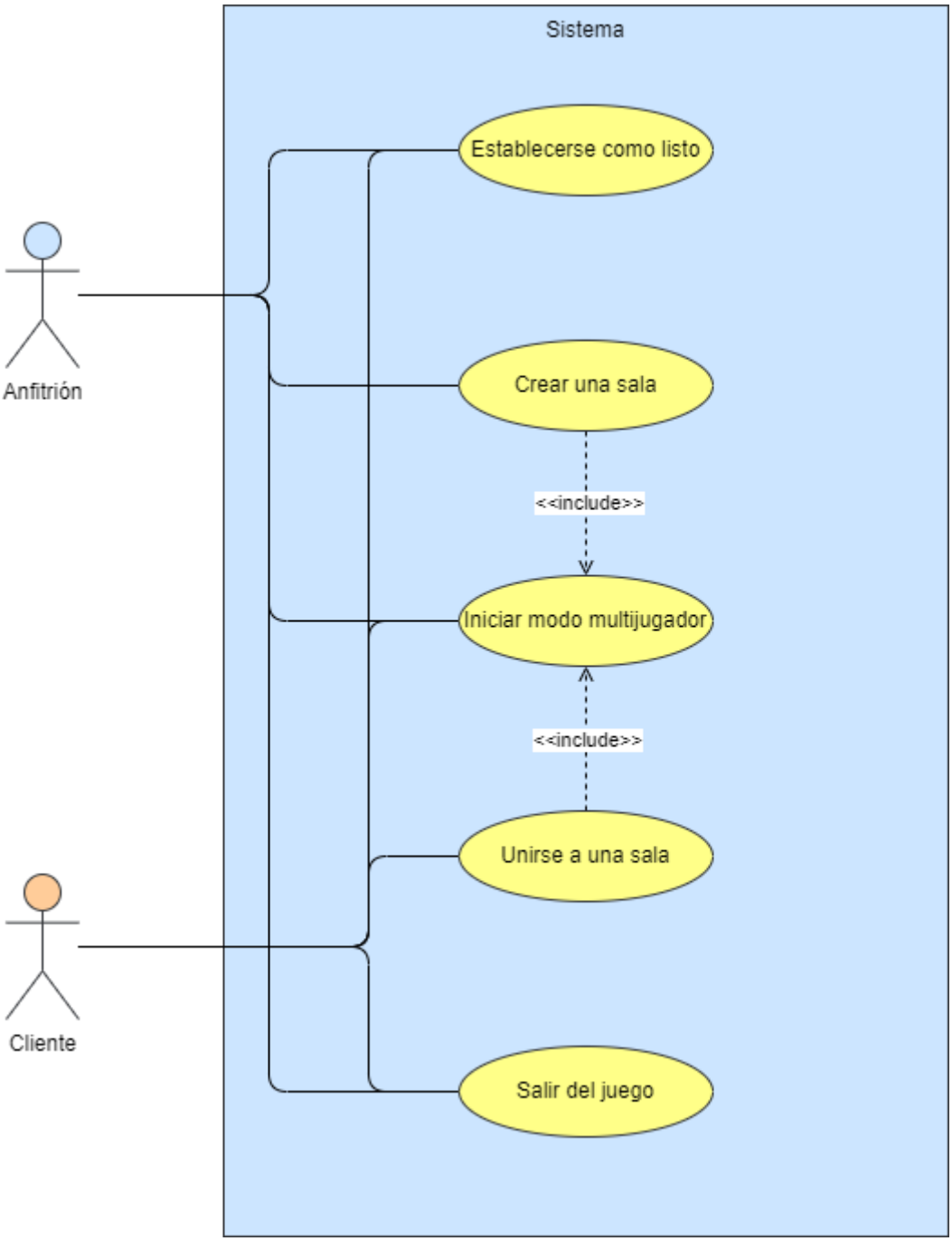

Fig. 3.3. Diagrama de casos de uso del escenario 3

En este diagrama se reutiliza el caso de uso *UC16 - Salir de la aplicación*. Así mismo, se incorporan los siguientes casos de uso nuevos:

TABLA 3.3.
CASOS DE USO DEL ESCENARIO 3

Identificador	**Nombre del caso de uso**
CU17	Iniciar modo multijugador

Identificador	Nombre del caso de uso
CU18	Crear una sala
CU19	Unirse a una sala
CU20	Establecerse como listo

3.1.2. Descripción de los casos de uso

A continuación, se va a proporcionar una descripción de alto nivel de cada caso de uso, mediante el uso de una tabla como la siguiente:

TABLA 3.4.
PLANTILLA DE LA DESCRIPCIÓN DE LOS CASOS DE USO

CUXX	
Nombre	
Actores	
Descripción	

En esta plantilla, se presentan los siguientes campos:

- **Nombre** del caso de uso.
- **Actores** involucrados.
- **Descripción** de alto nivel del caso de uso.

TABLA 3.5.
CASO DE USO CU01

CU01	
Nombre	Mover el PJ
Actores	Jugador
Descripción	El jugador es capaz de mover a su personaje jugable por el escenario mediante una señal de entrada.

TABLA 3.6.
CASO DE USO CU02

CU02	
Nombre	Evadir
Actores	Jugador
Descripción	El jugador puede realizar una esquiva en la dirección en la que se está moviendo. Por ende, si el jugador está parado, no podrá evadir en ninguna dirección.

TABLA 3.7.
CASO DE USO CU03

CU03	
Nombre	Apuntar el arma del PJ
Actores	Jugador
Descripción	El personaje jugable apunta en la dirección designada por una entrada que realiza el jugador.

TABLA 3.8.
CASO DE USO CU04

CU04	
Nombre	Disparar proyectil
Actores	Jugador
Descripción	El PJ del jugador dispara un proyectil en la dirección en la que está apuntando su arma. Este proyectil sigue una trayectoria recta hasta colisionar con un objeto del entorno o un enemigo.

TABLA 3.9.
CASO DE USO CU05

CU05	
Nombre	Curar al PJ
Actores	Jugador
Descripción	El jugador puede restablecer parcial o completamente sus puntos de vida, haciendo uso de algún objeto consumible, que puede ser encontrado por el escenario.

TABLA 3.10.
CASO DE USO CU06

CU06	
Nombre	Dañar al PJ
Actores	Jugador
Descripción	El personaje del jugador puede ser dañado por alguna fuente externa, como el impacto de un proyectil enemigo. Al ser dañado, se reducen sus puntos de salud una cantidad determinada por el daño ocasionado por la fuente externa.

TABLA 3.11.
CASO DE USO CU07

CU07	
Nombre	Matar al PJ
Actores	Jugador
Descripción	Cuando el jugador recibe daño y los puntos de salud de su personaje se agotan (sus puntos de vida actuales son inferiores o igual a 0), el jugador pierde el control de su personaje y desaparece de la escena. La ejecución de este caso de uso desencadena también la ejecución del caso de uso CU08.

TABLA 3.12.
CASO DE USO CU08

CU08	
Nombre	Perder la partida
Actores	Jugador
Descripción	Cuando el personaje del jugador fallece (se ejecuta el caso de uso CU07), el sistema indica visualmente el fin de la partida, permitiendo al jugador volver al menú de inicio.

TABLA 3.13.
CASO DE USO CU09

CU09	
Nombre	Ganar la partida
Actores	Jugador
Descripción	Cuando el jugador llega a un punto específico del escenario, que se considera como la meta o salida, el sistema le muestra de manera visual el fin de la partida, habiendo cumplido el objetivo del juego.

TABLA 3.14.
CASO DE USO CU10

CU10	
Nombre	Salir al menú principal
Actores	Jugador
Descripción	El jugador puede salir al menú principal en cualquier punto de la partida. Esto implica que el flujo de la partida actual termina sin guardarse el progreso actual, por lo que, cuando el jugador acceda nuevamente a alguno de los modos de juego, empezará una partida completamente distinta.

TABLA 3.15.
CASO DE USO CU11

CU11	
Nombre	Matar al enemigo
Actores	Enemigo
Descripción	Cuando los puntos de vida de un enemigo se agoten tras ser dañado, el sistema se encargará de eliminarlo de la escena, proporcionando al jugador una respuesta visual y/o sonora.

TABLA 3.16.
CASO DE USO CU12

CU12	
Nombre	Dañar al enemigo
Actores	Enemigo
Descripción	En el momento en el que un jugador logre impactar uno de los proyectiles disparados por su arma en un enemigo, este último sufrirá una reducción en sus puntos de vida actuales. Esta reducción será igual al daño del jugador.

TABLA 3.17.
CASO DE USO CU13

CU13	
Nombre	Atacar al PJ
Actores	Enemigo
Descripción	Un enemigo puede atacar al personaje de un jugador de diversas formas distintas, ya sea golpeándole directamente o atacándole desde la distancia.

TABLA 3.18.
CASO DE USO CU14

CU14	
Nombre	Perseguir al PJ
Actores	Enemigo
Descripción	Cuando un jugador entre en el rango de detección de un enemigo, este empezará a perseguir al primero, hasta que consiga asesinarlo o muera en el intento.

TABLA 3.19.
CASO DE USO CU15

CU15	
Nombre	Iniciar modo un jugador
Actores	Anfitrión
Descripción	Un jugador puede acceder al modo un jugador desde el menú principal del videojuego. Este modo le permite jugar sin conexión a Internet, accediendo directamente a una nueva partida en solitario.

TABLA 3.20.
CASO DE USO CU16

CU16	
Nombre	Salir de la aplicación
Actores	Anfitrión, cliente
Descripción	Un jugador puede interactuar con el sistema para cerrar la aplicación desde el menú principal, lo que finaliza completamente el proceso.

TABLA 3.21.
CASO DE USO CU17

CU17	
Nombre	Iniciar modo multijugador
Actores	Anfitrión, cliente
Descripción	Los jugadores pueden acceder al menú del modo multijugador desde el menú principal. El menú del modo multijugador permitirá a los jugadores crear o unirse a una sala de juego (CU18 y CU19), o regresar al menú principal.

TABLA 3.22.
CASO DE USO CU18

CU18	
Nombre	Crear una sala
Actores	Anfitrión
Descripción	Un jugador puede, desde el menú del modo multijugador, crear una sala. En este instante, este jugador pasará a ser el anfitrión de una partida. En esta sala el anfitrión tendrá en su poder un código identificador que podrá proporcionar a otro jugador para que se una a su sala de juego.

TABLA 3.23.
CASO DE USO CU19

CU19	
Nombre	Unirse a una sala
Actores	Cliente
Descripción	Un jugador podrá unirse a una sala de juego desde el menú del modo multijugador, introduciendo un identificador proporcionado por el jugador que ha creado la sala a la que se está intentando unir.

TABLA 3.24.
CASO DE USO CU20

CU20	
Nombre	Establecerse como listo
Actores	Anfitrión, cliente
Descripción	Cuando un jugador se haya unido a una sala, podrá establecer su estado a "listo". Una vez todos los jugadores de la sala hayan ejecutado esta acción, se cargará la escena de juego y este dará comienzo.

3.2. Especificación de requisitos de software

Cada requisito se especifica con una tabla como la que se encuentra a continuación:

TABLA 3.25.
PLANTILLA DE LOS REQUISITOS

Identificador:	
Descripción:	
Prioridad:	**Necesidad:**
Verificabilidad:	**Casos de uso asociados:**

Donde cada uno de los campos representa lo siguiente:

- **Identificador:** representa la identificación del requisito con una cadena de caracteres con el formato *<F/NF>-<U/S>-<C/R><XXX>*, donde:
  - *<F/NF>*
    - **F:** requisito funcional.
    - **NF:** requisito no funcional.
  - *<U/S>*
    - **U:** requisito de usuario.
    - **S:** requisito del sistema.
  - *<C/R>*
    - **C:** requisito de capacidad.
    - **R:** requisito de restricción.
  - *<XXX>*

        - Número identificador del requisito.
- **Descripción:** breve explicación del requisito.
- **Prioridad:** indica la preferencia del requisito sobre el resto. Obtiene los siguientes valores:
    - **Alta.** El requerimiento es esencial para el funcionamiento del sistema y otros requisitos dependen de este.
    - **Media.** El requisito depende de otro de mayor prioridad y otros dependen de él.
    - **Baja.** Nivel de prioridad más bajo. La realización del requisito no es fundamental o depende de otros requisitos de mayor prioridad.
- **Necesidad:** representa la obligatoriedad del requisito. Obtiene los siguientes valores:
    - **Obligatorio:** el requisito es indispensable para el sistema.
    - **Opcional:** representa una característica optativa del sistema.
- **Verificabilidad:** indica si el requisito se puede verificar con un plan de pruebas. Puede obtener los siguientes valores:
    - **Total:** se puede verificar en su totalidad mediante un plan de pruebas.
    - **Parcial:** no se puede verificar en su totalidad su correcto funcionamiento.
    - **Nula:** no es posible verificar el requisito en base a las pruebas.
- **Casos de uso asociados:** indica el identificador de los casos de uso asociados al requisito.

3.2.1. Requisitos funcionales

A continuación, se van a listar todos los requisitos funcionales especificados para este proyecto.

TABLA 3.26.
REQUISITO F-U-C01

Identificador: F-U-C01	
Descripción: El jugador deberá poder mover su personaje jugable por el escenario mediante una señal de entrada.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU01

TABLA 3.27.
REQUISITO F-S-R02

Identificador: F-S-R02	
Descripción: El sistema deberá controlar el movimiento de los personajes del juego, para que nunca realicen movimientos inválidos.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU01

TABLA 3.28.
REQUISITO F-U-C03

Identificador: F-U-C03	
Descripción: El jugador deberá poder realizar una evasión en la dirección y sentido del vector de movimiento de su personaje.	
Prioridad: Baja	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU02

TABLA 3.29.
REQUISITO F-S-R04

Identificador: F-S-R04	
Descripción: El sistema deberá controlar la frecuencia con la que el usuario puede realizar una evasión.	
Prioridad: Baja	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU02

TABLA 3.30.
REQUISITO F-U-C05

Identificador: F-U-C05	
Descripción: El jugador deberá poder apuntar el arma de su personaje, en cualquier dirección, alrededor de este.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU03

TABLA 3.31.
REQUISITO F-U-C06

Identificador: F-U-C06	
Descripción: El jugador deberá poder disparar un proyectil con su arma en la dirección en la que esté apuntando.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU04

TABLA 3.32.
REQUISITO F-S-R07

Identificador: F-S-R07	
Descripción: El sistema deberá controlar la cadencia de disparo del jugador.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU04

TABLA 3.33.
REQUISITO F-S-C08

Identificador: F-S-C08	
Descripción: El sistema deberá controlar la vida de los jugadores y enemigos, basándose en un sistema de puntos de salud.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU05, CU06, CU11, CU12

TABLA 3.34.
REQUISITO F-U-C09

Identificador: F-U-C09	
Descripción: El usuario deberá poder restablecer los puntos de vida de su personaje haciendo uso de un botiquín que se encontrará por el escenario.	
Prioridad: Media	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU05

TABLA 3.35.
REQUISITO F-S-R10

Identificador: F-S-R10	
Descripción: El sistema deberá controlar que nunca se sobrepasen los puntos de vida máximos del jugador.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU05

TABLA 3.36.
REQUISITO F-S-C11

Identificador: F-S-C11	
Descripción: El sistema deberá dañar al personaje del jugador cuando este reciba un golpe proveniente de un enemigo.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU06

TABLA 3.37.
REQUISITO F-S-C12

Identificador: F-S-C12	
Descripción: El sistema deberá eliminar al personaje del jugador cuando los puntos de vida del susodicho alcancen el valor de 0.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU07

TABLA 3.38.
REQUISITO F-S-R13

Identificador: F-S-R13	
Descripción: El sistema deberá controlar que los puntos de vida del jugador nunca alcancen un valor negativo y este pueda seguir jugando.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU06, CU07, CU08

TABLA 3.39.
REQUISITO F-S-C14

Identificador: F-S-C14	
Descripción: El sistema deberá terminar el flujo de la partida cuando el personaje del jugador muere.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU08

TABLA 3.40.
REQUISITO F-U-C15

Identificador: F-U-C15	
Descripción: El jugador deberá percibir el final de la partida mediante estímulos visuales o sonoros, con el motivo por el que ha terminado la partida.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU08, CU09

TABLA 3.41.
REQUISITO F-S-C16

Identificador: F-S-C16	
Descripción: El sistema deberá terminar el flujo de la partida cuando un personaje jugable alcance la meta establecida.	
Prioridad: Baja	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU09

TABLA 3.42.
REQUISITO F-U-C17

Identificador: F-U-C17	
Descripción: El usuario deberá poder volver al menú principal en cualquier momento, sin tener que cerrar y volver a abrir la aplicación.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU08, CU09, CU10

TABLA 3.43.
REQUISITO F-S-R18

Identificador: F-S-R18	
Descripción: El sistema no deberá guardar el progreso actual del jugador, una vez este vuelva al menú principal.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU10

TABLA 3.44.
REQUISITO F-S-C19

Identificador: F-S-C19	
Descripción: El sistema deberá eliminar de la escena a un enemigo cuando sus puntos de vida se reduzcan a 0.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU11

TABLA 3.45.
REQUISITO F-U-C20

Identificador: F-U-C20	
Descripción: El usuario deberá recibir una respuesta visual y sonora cuando un enemigo muera.	
Prioridad: Media	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU11

TABLA 3.46.
REQUISITO F-S-C21

Identificador: F-S-C21	
Descripción: El sistema deberá reducir los puntos de vida de un enemigo cuando el proyectil de un jugador lo alcance, reduciendo los puntos de vida en base al daño del proyectil.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU12

TABLA 3.47.
REQUISITO F-S-R22

Identificador: F-S-R22	
Descripción: El sistema deberá controlar que la vida de un enemigo no alcance un valor negativo.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU11, CU12

TABLA 3.48.
REQUISITO F-S-C23

Identificador: F-S-C23	
Descripción: El sistema deberá permitir a un enemigo atacar al personaje de un jugador, mediante el uso de proyectiles o de su propio cuerpo.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU13

TABLA 3.49.
REQUISITO F-U-C24

Identificador: F-U-C24	
Descripción: El usuario deberá recibir un estímulo visual y sonoro cuando su personaje jugable reciba daño.	
Prioridad: Media	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU06, CU13

TABLA 3.50.
REQUISITO F-S-C25

Identificador: F-S-C25	
Descripción: El sistema deberá permitir a un enemigo perseguir a un jugador a través del escenario.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU14

TABLA 3.51.
REQUISITO F-S-R26

Identificador: F-S-R26	
Descripción: El sistema deberá controlar que un enemigo no persiga a un jugador si este está fuera de su rango de detección.	
Prioridad: Baja	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU14

TABLA 3.52.
REQUISITO F-U-C27

Identificador: F-U-C27	
Descripción: Un jugador deberá poder iniciar una partida en el modo un jugador desde el menú principal del juego.	
Prioridad: Media	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU15

TABLA 3.53.
REQUISITO F-U-C28

Identificador: F-U-C28	
Descripción: El jugador deberá poder jugar al modo un jugador sin una conexión a Internet.	
Prioridad: Baja	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU15

TABLA 3.54.
REQUISITO F-U-C29

Identificador: F-U-C29	
Descripción: Un jugador deberá ser capaz de salir de la aplicación desde el menú principal, finalizando por completo el proceso.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU16

TABLA 3.55.
REQUISITO F-U-C30

Identificador: F-U-C30	
Descripción: Un jugador deberá poder acceder al modo multijugador desde el menú principal.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU17

TABLA 3.56.
REQUISITO F-S-C31

Identificador: F-S-C31	
Descripción: El sistema deberá mostrar al usuario un menú para el modo multijugador, que permita al usuario elegir entre crear una sala de juego o unirse a una.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU17, CU18, CU19

TABLA 3.57.
REQUISITO F-S-C32

Identificador: F-S-C32	
Descripción: El sistema deberá crear una sala de juego cuando un usuario seleccione dicha opción desde un menú contextual, haciendo que el usuario se convierta en el anfitrión de una sesión multijugador.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU18

TABLA 3.58.
REQUISITO F-S-C33

Identificador: F-S-C33	
Descripción: El sistema deberá mostrar al usuario un código de acceso que permita a otro usuario unirse a la sala que ha creado.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU18, CU19

TABLA 3.59.
REQUISITO F-S-C34

Identificador: F-S-C34	
Descripción: El sistema deberá permitir a un jugador unirse a una sala creada por otro usuario, siempre y cuando este disponga del código de la sala.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU19

TABLA 3.60.
REQUISITO F-U-C35

Identificador: F-U-C35	
Descripción: Un usuario deberá poder establecer su estado a listo una vez se haya unido a una sala.	
Prioridad: Baja	**Necesidad:** Opcional
Verificabilidad: Parcial	**Casos de uso asociados:** CU20

TABLA 3.61.
REQUISITO F-S-R36

Identificador: F-S-R36	
Descripción: El sistema deberá comenzar la partida solo si todos los usuarios de la sala han establecido su estado a listo.	
Prioridad: Baja	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU20

TABLA 3.62.
REQUISITO F-S-C37

Identificador: F-S-C37	
Descripción: El sistema deberá mostrar en todo momento al jugador los puntos de vida actuales de su personaje, así como cualquier otra estadística relevante para su supervivencia.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU05, CU06, CU07

TABLA 3.63.
REQUISITO F-U-C38

Identificador: F-U-C38	
Descripción: El jugador deberá obtener un breve periodo de invulnerabilidad tras recibir daño.	
Prioridad: Baja	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU06

TABLA 3.64.
REQUISITO F-U-C39

Identificador: F-U-C39	
Descripción: El jugador deberá obtener por pantalla la razón por la que ha finalizado la partida.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU08, CU09

TABLA 3.65.
REQUISITO F-S-C40

Identificador: F-S-C40	
Descripción: El sistema deberá mostrar al jugador un mensaje cuando se produzca un error o comportamiento inesperado en el flujo de la partida.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Parcial	**Casos de uso asociados:** CU17, CU18, CU19

TABLA 3.66.
REQUISITO F-S-R41

Identificador: F-S-R41	
Descripción: El sistema deberá terminar la partida cuando uno de los jugadores sufra una desconexión en el modo multijugador.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU17, CU18, CU19

TABLA 3.67.
REQUISITO F-S-R42

Identificador: F-S-R42	
Descripción: El sistema deberá limitar la cantidad máxima de jugadores en el modo multijugador a 2 por sala de juego.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** CU17, CU18, CU19

3.2.2. Requisitos no funcionales

Los requisitos no funcionales especificados para este proyecto son los siguientes:

TABLA 3.68.
REQUISITO NF-S-C43

Identificador: NF-S-C43	
Descripción: La aplicación deberá estar disponible para el sistema operativo Microsoft Windows 10/11 x64.	
Prioridad: Alta	**Necesidad:** Obligatorio
Verificabilidad: Parcial	**Casos de uso asociados: -**

TABLA 3.69.
REQUISITO NF-S-C44

Identificador: NF-S-C44	
Descripción: El sistema deberá estar traducido completamente al idioma inglés.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** -

TABLA 3.70.
REQUISITO NF-S-C45

Identificador: NF-S-C45	
Descripción: El sistema deberá proporcionar al jugador, antes de comenzar la partida, un tutorial con las acciones que puede realizar y los controles asociados.	
Prioridad: Baja	**Necesidad:** Opcional
Verificabilidad: Total	**Casos de uso asociados:** CU01, CU02, CU03, CU04

TABLA 3.71.
REQUISITO NF-S-R46

Identificador: NF-S-R46	
Descripción: El sistema deberá respetar la privacidad de los jugadores en todo momento, ocultando la IP pública de los jugadores cuando se conecten a una sesión multijugador.	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Parcial	**Casos de uso asociados:** CU17, CU18, CU19

TABLA 3.72.
REQUISITO NF-S-R47

Identificador: NF-S-R47	
Descripción: El código de la aplicación deberá seguir las convenciones de codificación y nombramiento de C# [8].	
Prioridad: Media	**Necesidad:** Obligatorio
Verificabilidad: Total	**Casos de uso asociados:** -

3.3. Matriz de trazabilidad

Acto seguido a la especificación de los casos de uso y requisitos, se presenta la tabla 3.73, que representa la matriz de trazabilidad de los casos de uso y los requisitos funcionales del proyecto. Como se puede observar en esta tabla, todos los casos de uso son cubiertos por uno o más requisitos.

Para facilitar la visualización de la matriz de trazabilidad, en el caso de los requisitos solo se ha especificado su número identificador, puesto que este es único para cada requisito. Además, los otros caracteres del identificador del requisito no ofrecen ninguna funcionalidad en esta tabla.

TABLA 3.73.
MATRIZ DE TRAZABILIDAD

	CU 01	**CU 02**	**CU 03**	**CU 04**	**CU 05**	**CU 06**	**CU 07**	**CU 08**	**CU 09**	**CU 10**	**CU 11**	**CU 12**	**CU 13**	**CU 14**	**CU 15**	**CU 16**	**CU 17**	**CU 18**	**CU 19**	**CU 20**
01	X																			
02	X																			
03		X																		
04		X																		
05			X																	
06				X																
07				X																
08					X	X					X	X								
09					X															
10					X															
11						X														
12							X													
13						X	X	X												
14								X												
15								X	X											
16									X											
17								X	X	X										
18										X										

	CU 01	CU 02	CU 03	CU 04	CU 05	CU 06	CU 07	CU 08	CU 09	CU 10	CU 11	CU 12	CU 13	CU 14	CU 15	CU 16	CU 17	CU 18	CU 19	CU 20
19											X									
20											X									
21												X								
22											X	X								
23													X							
24						X							X							
25														X						
26														X						
27															X					
28															X					
29																X				
30																	X			
31																	X	X	X	
32																		X		
33																		X	X	
34																			X	
35																				X
36																				X
37					X	X	X													
38						X														
39								X	X											
40																	X	X	X	

	CU 01	CU 02	CU 03	CU 04	CU 05	CU 06	CU 07	CU 08	CU 09	CU 10	CU 11	CU 12	CU 13	CU 14	CU 15	CU 16	CU 17	CU 18	CU 19	CU 20
41																	X	X	X	
42																	X	X	X	

3.4. Tecnologías utilizadas

A lo largo del desarrollo de este proyecto, se han utilizado las tecnologías que se detallan a continuación:

- Unity Engine: el motor de desarrollo de videojuegos utilizado durante la creación del prototipo. La versión del editor de Unity utilizada ha sido la 22.3.8f1, versión con soporte a largo plazo (LTS). Adicionalmente, se han utilizado los siguientes paquetes y *assets*:
  - A* Pathfinding Project: paquete de Unity creado por Aron Granberg, que implementa una solución de búsqueda heurística A*, usada para la inteligencia artificial de los enemigos [62]. Se ha utilizado en su versión 4.2.17 gratuita que, aunque presenta varias limitaciones respecto a la versión de pago (que tiene un valor de 140$ en la Unity Asset Store), es suficiente para el desarrollo de este prototipo.
  - Input System: paquete de Unity que permite utilizar el nuevo sistema de entrada de Unity, que pretende sustituir al sistema anterior, el Input Manager [63].
  - UI Toolkit: es el último sistema desarrollado por Unity para implementar interfaces de usuario. Incluye diversas herramientas y recursos para crear interfaces de usuario en los videojuegos desarrollados o para el propio editor de Unity [64].
  - Universal RP: paquete que permite utilizar la Universal Render Pipeline (URP). La URP posibilita la utilización de técnicas de renderizado avanzadas y efectos de posprocesado [65]. Se ha utilizado en su versión 14.0.8.
  - Netcode for GameObjects: solución utilizada para la implementación del multijugador. Se ha empleado en su versión 1.5.2.
  - Multiplayer Tools: este paquete incluye varias herramientas que ayudan a desarrollar juegos multijugador y el correcto funcionamiento de estos [66]. Se ha empleado en su versión 1.1.0.
  - Lobby: paquete de Unity que permite utilizar el servicio Lobby, de Unity Gaming Services. Este servicio permite a los usuarios crear salas de juego para poder interactuar con otros jugadores, así como unirse a una sala en concreto o buscar las sesiones disponibles [67]. Se ha utilizado la versión 1.2.1 del paquete.

    - Relay: servicio de Unity Gaming Services que permite a los jugadores conectarse a un servidor de retransmisión, lo que facilita la interconexión entre jugadores que estén en distintas redes [68]. La versión del paquete de Relay que se ha utilizado durante el desarrollo de este proyecto ha sido la 1.0.5.
- Visual Studio 2022: entorno de desarrollo integrado (IDE) empleado para la creación y edición de los *scripts* de C#.
- Aseprite: herramienta utilizada para la creación de los *sprites* (imágenes bidimensionales) desarrollados durante este proyecto. Aseprite es un editor de *pixel-art*, que permite crear imágenes y animaciones en este estilo artístico [69].
- Audacity: herramienta de código abierto para la edición de pistas de audio. Se ha empleado para modificar los sonidos utilizados en este prototipo.
- Microsoft Office Word: editor de texto utilizado para la creación y edición de este documento.
- Microsoft Office Excel: editor de hojas de cálculo utilizado para el mantenimiento, principalmente, de los requisitos, así como algunas tablas que se encuentran en este documento.
- Draw.io: aplicación de código abierto utilizada para la edición de los diagramas y figuras que se encuentran a lo largo del documento [70].
- Zotero: herramienta empleada para la gestión de las referencias bibliográficas de esta memoria.
- GitHub/GitHub Desktop: GitHub es un servicio basado en la nube que implementa el sistema de control de versiones Git [71]. Se ha utilizado para el alojamiento de los contenidos de este proyecto. También se ha empleado GitHub Desktop, el cliente de escritorio para el sistema operativo Windows y macOS, que facilita la gestión de este repositorio [72].
- Ordenador con el sistema operativo Windows 10 de 64 bits: computador personal utilizado durante el desarrollo del trabajo de fin de grado.

3.5. Metodología de desarrollo utilizada

Durante el transcurso de este proyecto, se ha utilizado la metodología de desarrollo Kanban. Esta metodología, que tiene sus orígenes en Japón, consiste en dividir las tareas en tarjetas que se clasifican en un tablero. Este tablero está dividido en varias columnas o categorías, lo que permite mover las distintas tarjetas dentro de las categorías, dependiendo de su fase de desarrollo. En este caso, las columnas en las que se ha dividido el tablero han sido:

- Para hacer: categoría en la que se añaden las tarjetas inmediatamente después de su creación. Aquí estarán aquellas tareas que no se ha comenzado a implementar.
- En curso: las tareas se mueven a esta categoría cuando están en realización.
- Finalizado: a esta categoría se mueven aquellas tarjetas que han terminado su ciclo de desarrollo porque han sido completadas con éxito.

- Desestimado: en esta sección del tablero se mueven aquellas tareas cuya realización se ha desestimado, ya sea por tiempo o por complejidad.

Los motivos por los que se ha utilizado esta metodología de desarrollo han sido principalmente la familiaridad del estudiante con esta metodología, puesto que ya había sido empleada previamente. Además, ofrece un alto grado de flexibilidad, pudiendo dar prioridad a ciertas tareas sobre otras según las necesidades puntuales del proyecto, o crear nuevas tareas que surjan durante la progresión de otra. También ayuda a tener una imagen global del proyecto, siendo posible visualizar el progreso de una manera sencilla y eficaz.

4. DISEÑO

En el apartado actual se van a explicar y justificar las decisiones de diseño tomadas durante la ejecución del trabajo. Estas decisiones parten del análisis realizado anteriormente, interpretando las conclusiones halladas en este y las amplían.

El capítulo se va a dividir en dos epígrafes principales. En el primero se detallan las decisiones de diseño relacionadas al apartado artístico o estético del proyecto. En el apartado técnico se especifican las decisiones asociadas a la lógica del sistema.

Adicionalmente, se puede acceder al repositorio del proyecto a través del enlace que se presenta a continuación:

https://github.com/AMorata086/TFG-Roguelike

En este repositorio se puede encontrar toda la documentación asociada a este trabajo de fin de estudios, así como el proyecto completo de Unity, que incluye el código fuente desarrollado y resto de recursos utilizados. También se incluye la versión v1.0 del prototipo, compilada para el sistema operativo Windows 10/11 x64, que se puede encontrar bajo el apartado *releases* del repositorio.

4.1. Apartado artístico

En primer lugar, es necesario hablar de la ambientación que se ha decidido proporcionar a este videojuego. La trama del videojuego se desarrolla en una estación espacial abandonada, donde el jugador, que tomará el control de un astronauta armado con una pistola, tendrá que aventurarse por esta mazmorra, en busca de un tesoro perdido. El jugador se encontrará con enemigos de origen robótico en las habitaciones de esta estación, que intentarán acabar con su vida.

Para el apartado visual del proyecto, se ha tenido como primera consideración el hecho de que el proyecto ha sido desarrollado por una sola persona, con unos conocimientos artísticos relativamente bajos. Por ello, se ha decidido usar un estilo lo más simple posible, el estilo *pixel-art*. Este estilo artístico digital se basa en la edición de imágenes píxel por píxel, permitiendo al artista crear imágenes con un nivel detalle muy bajo, como podrían ser los gráficos que se utilizan en videojuegos retro, o aumentar el nivel de detalle hasta los cientos de píxeles por pulgada.

En este prototipo, se ha decidido utilizar cuadrículas de 16x16 píxeles. Es decir, cada imagen que represente una unidad mínima del sistema, como una baldosa del suelo, tendrá 16 píxeles de ancho y 16 píxeles de alto, con un total de 256 píxeles. Esto permite simplificar el diseño visual del prototipo, al tener un bajo nivel de detalle.

En cuanto a la dimensionalidad del arte, se ha decidido optar por un estilo bidimensional, puesto que es, por lo general, más sencillo trabajar con gráficos 2D que con gráficos en 3D, lo que implica, además, la creación de modelados tridimensionales.

Es necesario comentar que otra opción hubiese sido utilizar un paquete de *assets* (recursos) de la Unity Store, lo que hubiera acortado el proceso creativo considerablemente. Esta idea se desestimó porque normalmente los paquetes de

recursos gratuitos no ofrecen imágenes para todas las partes del escenario que se quieren desarrollar en un proyecto, lo que acaba creando un conflicto entre los estilos artísticos empleados. Otro problema que surge del uso de estos paquetes es que generalizan los proyectos, ya que las personas tienden a utilizar aquellos paquetes más populares, por lo que puede conseguir que se pierda el carácter creativo de un proyecto.

Para este proyecto cabe destacar también que se han creado animaciones, principalmente para los personajes que aparecen en la escena, como los jugadores o enemigos. En la figura 4.1, encontrada debajo de este párrafo, se puede visualizar la *sprite sheet* (conjunto de *sprites* en una sola imagen, que representan en su conjunto a una sola entidad) del jugador 1, que incluye todas las imágenes que se utilizan para la animación del personaje del jugador.

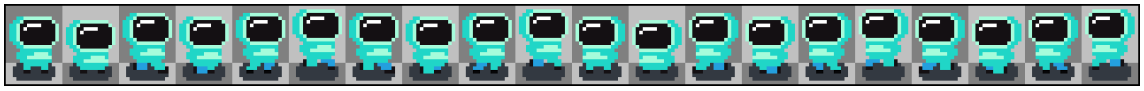

Fig. 4.1. *Sprite sheet* del personaje del jugador 1

En total, se han diseñado los siguientes elementos visuales:

- *Sprite sheets* para el jugador 1 y jugador 2: variantes para el jugador 1 y el jugador 2. El personaje del jugador 1 (anfitrión) tiene un tono azul celeste, mientras que el jugador 2 (cliente) tiene un tono amarillo.
- Armas de los jugadores: dos imágenes que representan las armas del jugador 1 y jugador 2, con forma de pistola.
- *Sprite sheets* para los enemigos: se han modelado *sprite sheets* distintas para los tres enemigos implementados en el prototipo.
- *Tileset* del escenario: contiene cada una de las baldosas, paredes, puertas y variaciones distintas que se colocan para formar cada una de las habitaciones del nivel o mapa.
- *Sprites* de los proyectiles: se han modelado un total de 5 proyectiles distintos, dos de ellos para los jugadores y tres para los distintos enemigos.
- Mira o puntero: se ha creado un puntero customizado para el juego. Este puntero tiene forma de cruceta o mira, para ayudar a los jugadores con el apuntado.
- *Sprites* de los obstáculos y objetos del mapa: un total de tres imágenes han sido diseñadas para los objetos y obstáculos que se pueden encontrar por el mapa, estos son: un botiquín, un cofre y una caja de madera, siendo esta última el único obstáculo que se puede encontrar por el mapa.
- *Sprites* para la interfaz gráfica y sistemas de partículas: se han diseñado dos imágenes para su uso en la interfaz gráfica del usuario y para su uso en el sistema de partículas.

Por último, en cuanto al apartado sonoro y musical, dado que no se dispone de un estudio de sonido, ni del conocimiento necesario para crear efectos de sonido, se ha decidido utilizar recursos gratuitos y de código abierto. Estos recursos se han obtenido en todos los casos de la página Freesound, un repositorio con efectos de sonido y música con licencias de código abierto [73].

4.2. Apartado técnico

En esta sección se habla de las decisiones de diseño relevantes a la lógica del videojuego. Por tanto, se comentan las interacciones posibles que puede realizar el jugador con el sistema, además de las mecánicas implementadas más importantes para el funcionamiento del prototipo.

4.2.1. Interacciones del jugador con el sistema

En este fragmento del documento se van a tratar las distintas interacciones que puede llevar a cabo un jugador, explicando también las mecánicas asociadas a esas interacciones.

Movimiento del personaje jugable

Como se ha comentado en el apartado artístico, el juego se ha diseñado en un plano bidimensional, por lo que el movimiento también se ejecuta en dos dimensiones. También es necesario tener en consideración, como se comentaba anteriormente, que una unidad en Unity contendrá 16x16 píxeles por unidad. Esto es importante aclararlo y mantenerlo constante durante la ejecución de todo el proyecto, puesto que afecta directamente al sistema de físicas de Unity y al funcionamiento del paquete A* Pathfinding Project. En la figura 4.2 se muestra una ilustración que explica de manera visual los ejes de coordenadas utilizados.

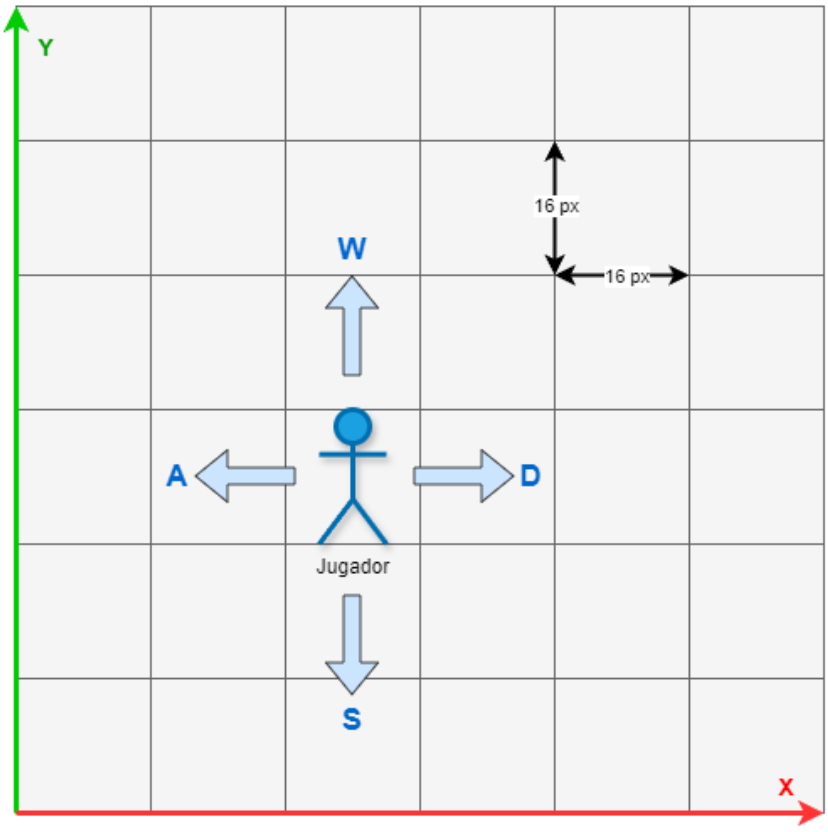

Fig. 4.2. Ejes de coordenadas y dimensiones empleados

Como se puede observar en la figura 4.2, el eje Z de coordenadas no se ha ilustrado, ya que su uso no es relevante en un videojuego en 2D, salvo para casos puntuales, como el

ordenamiento de los *sprites* en la escena o la posición de la cámara. Por ello, cuando un objeto de la escena tiene un valor menor que otro, el primero estará superpuesto sobre el segundo. Por esta razón, es recomendable establecer el valor del eje Z de la cámara al menor valor que se vaya a utilizar en este eje de manera global.

Como consecuencia de la explicación anterior, se puede hablar sobre el movimiento del personaje. Para que un jugador pueda mover al personaje dentro de la escena, es necesario que el sistema registre una señal de entrada. En este prototipo, el personaje se mueve mediante la pulsación de las teclas "W", "A", "S" y "D", tal y como se muestra en la figura 4.2.

El mecanismo que se utiliza para mover al personaje del jugador en el sistema hace uso del motor de simulación de físicas de Unity, Rigidbody. Para ello, el sistema aplica una fuerza determinada al personaje del jugador, que es un objeto físico en la escena, cuando una de las teclas anteriores se pulsa. Esto produce como resultado que el jugador se pueda mover en el plano XY, incluyendo en diagonal cuando se pulsan dos teclas simultáneamente.

Apuntar el arma del personaje

Para apuntar el arma del jugador, el sistema registra en cada fotograma la posición del cursor en la ventana del videojuego. Una vez consigue esta posición, en forma de vector, el sistema realiza la diferencia entre la posición del cursor y la posición del objeto sobre el que pivota el arma del jugador. Después, calcula la arcotangente de dos parámetros de las componentes X e Y del vector resultante de la diferencia. Este cálculo ofrece como resultado el ángulo, en radianes, en el que el jugador está apuntando su arma, por lo que, al aplicar este ángulo al pivote del arma en forma de una rotación, el arma acaba apuntando donde señala el cursor del jugador. En la figura 4.3 se detalla visualmente esta operación, para una mayor claridad.

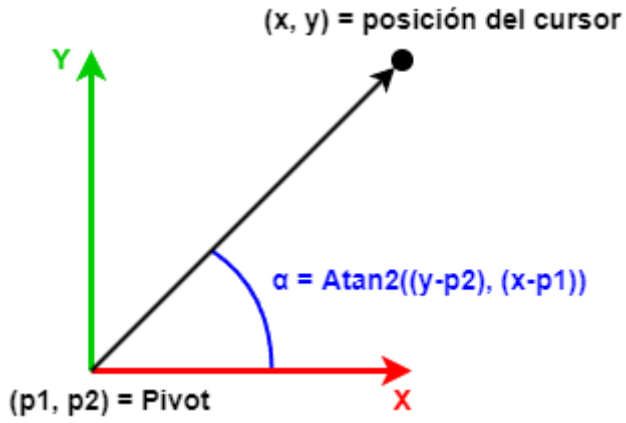

Fig. 4.3. Esquema explicativo del apuntado del arma

Disparar el arma del personaje

El jugador puede disparar el arma de su personaje mediante la utilización asignada a la acción de "*attack*" del esquema de controles creado, en este caso, asociada al clic izquierdo del ratón. Cuando el sistema registra esta acción, el sistema instancia una bala,

que tiene como punto origen la boca del cañón del arma del jugador y recibiendo la rotación deseada. Una vez instanciada, se le aplica una fuerza, que la propulsa en sentido opuesto a la ubicación del arma.

El proyectil disparado viajará siguiendo una trayectoria recta, hasta que impacte en un objeto del entorno, un proyectil enemigo o un enemigo. Cuando impacte con alguno de los objetos anteriores, el sistema eliminará el proyectil, produciendo unas partículas en el punto de colisión.

Además, es necesario remarcar que se ha implementado un periodo de enfriamiento para esta función. En la versión 1.0 del prototipo, este enfriamiento se ha establecido en 0,5 segundos, por lo que el jugador puede realizar 2 disparos por segundo.

Realizar una evasión

Al invocar la acción llamada "*Dodge*" en el esquema de controles implementado, que escucha la pulsación del botón derecho del ratón o la tecla "espacio", el personaje del jugador ejecuta una evasión. El sistema estará suscrito a este evento, por lo que, cuando se ejecute, se disparará la función asignada a ese evento. En la versión 1.0, la evasión tiene un periodo de enfriamiento de 2 segundos. Por ende, si el jugador ejecuta esta acción durante este periodo, el personaje del jugador no realizará una evasión. Por otro lado, si el personaje puede evadir, entonces el sistema aplica una fuerza sobre el vector de movimiento normalizado del jugador. En consecuencia, el personaje saldrá propulsado una pequeña distancia en la dirección en la que se esté moviendo. Sin embargo, si el personaje no está previamente en movimiento, este no realizará ninguna acción, manteniéndose quieto en la ubicación en la que está parado.

Curarse con un botiquín

Por el mapa se pueden encontrar botiquines con la apariencia que se puede apreciar en la figura 4.4. Cuando el jugador dispare a uno de estos botiquines o, dicho de otro modo, cuando uno de los proyectiles del jugador impacte contra un botiquín, la vida del jugador se restablecerá una cantidad determinada por el propio botiquín. El jugador se podrá curar tantas veces como se lo permita el botiquín, que desaparecerá una vez sus usos desciendan a cero. Además, si el jugador tiene sus puntos de vida al máximo, su personaje no se curará impactando al botiquín, pero sí gastará los usos de este último.

Fig. 4.4. *Sprite* del botiquín

Abrir el menú de pausa

Mediante la pulsación de la tecla "Escape", el jugador puede acceder al menú de pausa. En este menú, se despliega una lista de opciones, que permiten al jugador resumir la partida o terminarla, volviendo al menú principal del juego.

Cabe destacar que, cuando se ejecuta esta acción, no se pausa el tiempo en el juego. Es decir, cuando un jugador abre este menú, el juego sigue su transcurso natural, por lo que su uso no está recomendado cuando el jugador esté bajo ataque, puesto que va a seguir siendo dañado por los enemigos que estén presentes. La razón por la que se ha implementado esta solución sin paralizar el tiempo en el juego es, principalmente, el efecto negativo que tiene sobre el multijugador. Cuando un jugador pausa la partida, el estado del juego se tiene que sincronizar entre todos los jugadores, por lo que es necesario propagar este efecto entre los clientes. Esta paralización temporal no es complicada de implementar, pero sí que tiene efectos negativos en la experiencia de juego de los otros usuarios de la partida, que tienen que esperar, en ese caso, a que el jugador que ha paralizado la partida, la resuma.

4.2.2. Lógica de los enemigos

En el desarrollo de este prototipo se han desarrollado tres tipos de enemigos, según su forma de atacar, características de movimiento y apariencia.

En este prototipo se han modelado tres tipos de enemigos, expuestos a continuación:

- Enemigo 1 – Enemy_melee: este enemigo se caracteriza por atacar al jugador cuerpo a cuerpo, siendo el único de los tres tipos que ataca de este modo. Persigue al jugador sin cesar, intentando impactar contra él. Cuando se produce esta colisión, el jugador recibe una cantidad de daño establecida, además de ser empujado por una fuerza de reacción. Esta fuerza aleja al jugador del enemigo, ofreciéndole una ventana de tiempo para reaccionar y alejarse del enemigo. Este enemigo, además, es el que más puntos de vida tiene de los tres, por lo que es el que más golpes tiene que recibir para morir. En cuanto a la movilidad, es ligeramente más rápido que los personajes de los jugadores, teniendo como consecuencia que se tenga que hacer uso de la habilidad de evasión implementada, para así poder crear distancia.
- Enemigo 2 – Enemy_ranged: el segundo enemigo implementado es lento en cuanto a movimiento, pero capaz de atacar a grandes distancias. Además, los proyectiles que dispara, en caso de impactar al jugador, aplican a este último una cantidad de daño mayor que el resto de los enemigos. Aunque su cadencia de disparo es baja, disparando 1 proyectil por segundo en la versión v1.0, la velocidad a la que viajan estos proyectiles es alta, dando al jugador una ventana de tiempo menor para reaccionar. El uso de la esquiva también puede ayudar a evadir uno de estos proyectiles en un apuro. En cuanto a puntos de vida se refiere, se sitúan en un punto medio de los otros dos tipos de enemigos.
- Enemigo 3 – Enemy_flying: este último tipo es el más veloz de los tres, siendo mucho más rápido y ágil que el personaje de los jugadores. Este enemigo ataca a distancias medias, disparando proyectiles en la dirección del jugador con una cadencia muy alta. No obstante, estos proyectiles infligen una baja cantidad de daño al jugador, además de tener una velocidad de movimiento ligeramente superior a la del jugador, por lo que el jugador tiene tiempo suficiente para reaccionar. Sin embargo, si el jugador se descuida, puede ser abrumado por la

cantidad de proyectiles enemigos que puede haber en la sala. Estos enemigos son los más frágiles de los tres, requiriendo el menor número de impactos para destruirlos.

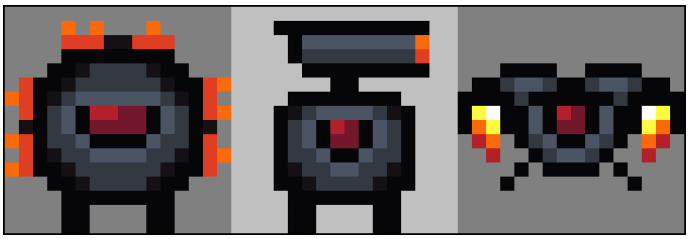

Fig. 4.5. Modelos 2D de los enemigos. De izquierda a derecha: enemigo 1, enemigo 2 y enemigo 3

A continuación, se explican en detalle las mecánicas principales de los enemigos, como la lógica detrás de la persecución a un jugador.

Persecución de los jugadores

La lógica base para la persecución de los enemigos es para los tres casos la misma: el enemigo busca a todos los jugadores presentes en la escena, eligiendo como objetivo al jugador que está más cerca. Una vez el personaje está dentro del rango de detección del enemigo, que varía según el tipo de este, el enemigo empezará a perseguir al personaje del jugador. Para ello, se emplea el paquete A* Pathfinding Project. Este paquete utiliza el algoritmo de búsqueda heurística A*, buscando el camino de menor coste entre la posición del enemigo y la del jugador. Este paquete permite utilizar varias heurísticas predefinidas, entre las que se encuentran la distancia de Manhattan y la distancia euclídea. Para este prototipo, se ha optado por usar la segunda, puesto que no se ha detectado ninguna diferencia significativa en el rendimiento o en el *pathfinding* de los enemigos (búsqueda de rutas).

Métodos de ataque

Según el tipo de ataque de los enemigos, se han definido dos métodos de ataque.

- En primer lugar, se explica el tipo de ataque de los enemigos cuerpo a cuerpo (enemigo 1). Estos enemigos no dejan de perseguir al jugador en ningún momento, pero cuando su *hitbox*, esto es, su “caja de impacto”, entra en contacto con la *hitbox* del personaje del jugador, este último será dañado. Entonces, el sistema restará a sus puntos de vida actuales la cantidad establecida por el daño de ataque del enemigo en cuestión.
- En segundo y último lugar, se definen los ataques de los enemigos a distancia (enemigos 2 y 3). En este caso, cuando los enemigos están a una distancia menor a una definida como distancia de disparo, los enemigos dejan de perseguir al jugador momentáneamente, mientras este se encuentre dentro del rango de disparo. Cuando el jugador está dentro de dicho rango, el enemigo empezará a disparar proyectiles, que siguen una trayectoria recta hacia la posición en la que se encuentra el jugador en el momento del disparo. Cuando estos proyectiles colisionan con el jugador, aplican un daño definido por el daño de ataque del enemigo en cuestión.

Este segundo tipo de ataque también tiene otra forma de interactuar con los jugadores, que se trata del choque de proyectiles del jugador con proyectiles enemigos. Cuando se produce una colisión entre estos dos tipos de proyectiles, se destruyen ambas municiones. Esto presenta una nueva mecánica que el jugador puede usar a su favor, siendo esta la destrucción de proyectiles enemigos que no puede esquivar a tiempo, pudiendo interceptarlos en pleno vuelo para defenderse. No obstante, esta mecánica también puede jugar en su contra, puesto que, si hay una gran cantidad de proyectiles entre los enemigos y el jugador, estos servirán de escudo para los primeros, dificultando el acierto de un proyectil para bajarles la vida o destruirlos.

4.2.3. Flujo de una partida

En este epígrafe se describe el flujo de una partida, desde su comienzo, hasta su finalización. Asimismo, también se realiza una descripción del escenario en el que se desarrolla esta partida.

Lo primero a detallar del flujo de la partida es la máquina de estados finita que almacena la etapa en la que se encuentra la partida. Esta máquina, en el prototipo implementado, está compuesta por cuatro estados:

- S0 – *SelectingGamemode*. Estado inicial del que se parte tras iniciar la partida. Se le ha asignado este nombre porque es el estado del que se parte. Cuando el jugador inicia una partida, lo hace desde un menú de selección del modo de juego, siendo los modos disponibles “un jugador” y “multijugador”. En este estado, el sistema realiza todas las acciones relevantes a la preparación de la escena, como la generación de los personajes de los jugadores y objetos de la escena.
- S1 – *WaitingToStart*. Una vez se han instanciado a todos los jugadores y objetos, el sistema avanza a este estado. Cuando el juego se encuentra en este estado, los jugadores que están presentes en la escena no se pueden mover. El sistema presenta al jugador un menú superpuesto, donde se enseñan los controles disponibles y las acciones asociadas a ellos. Una vez todos los jugadores presionen la tecla relevante para cambiar su estado a listo, tras haber leído el tutorial, el menú superpuesto desaparece y el sistema avanza al estado S2 – *InGame*.
- S2 – *InGame*. Al transitar a este estado, los jugadores pueden mover su personaje por la escena y realizar el resto de las interacciones que permite el programa, como disparar o realizar una evasión. Este estado es en el que el programa pasa la mayor parte del tiempo, puesto que representa todo el proceso de juego. En él, los jugadores pueden investigar el mapa con total libertad, enfrentarse a los enemigos que aparecen en cada una de las salas, etcétera.
- S3 – *GameFinished*. S3 es el estado final. Cuando un jugador alcanza la meta o, en su defecto, cuando muere, la máquina de estados avanza a este estado, en el que el sistema muestra un menú con la causa de finalización de la partida, además de una opción para volver al menú principal.

En la figura 4.6, se muestra el diagrama de estados del flujo de la partida, sin las transiciones, dado que estas se han descrito arriba en los párrafos anteriores.

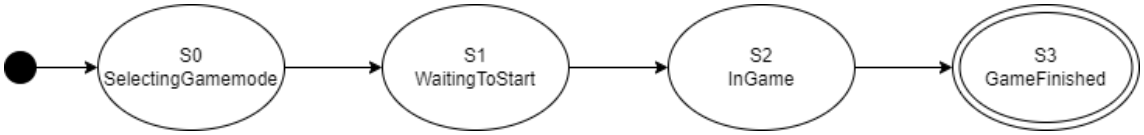

Fig. 4.6. Diagrama de estados del flujo de la partida

A continuación, se explica en detalle el flujo interno del estado S2, donde el jugador pasará la mayor parte del tiempo durante el desarrollo de la partida.

En primer lugar, es necesario hablar acerca del mapa o escenario de las partidas. Un elemento que hoy en día define a los juegos del género *roguelike*, es la generación aleatoria de los elementos de cada partida, como se habló en el estado de la cuestión. No obstante, debido a las limitaciones de tiempo y carga de trabajo, se ha decidido realizar un solo nivel estático, donde el mapa es siempre el mismo. Este mapa es el que se puede observar a continuación, en la figura 4.7.

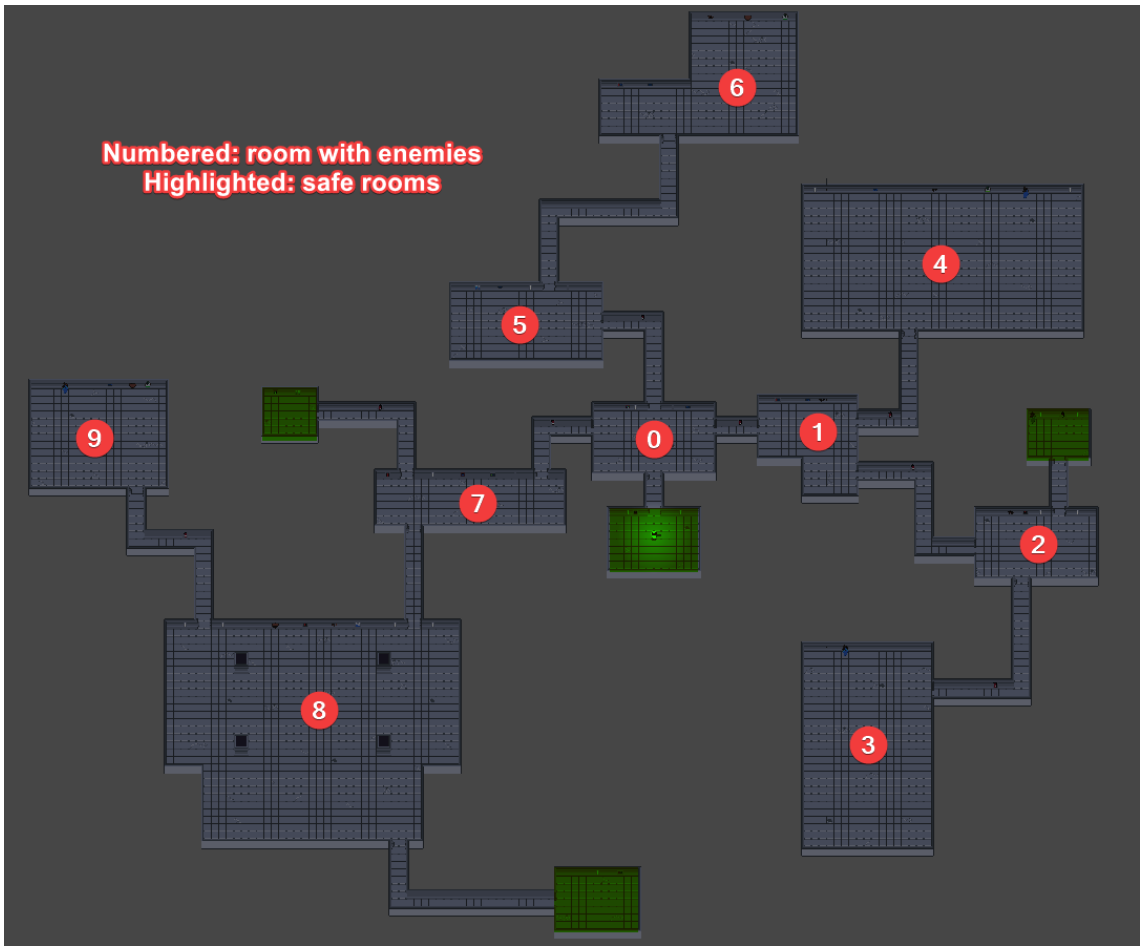

Fig. 4.7. Mapa del nivel 1 del prototipo

En este mapa, como se puede apreciar en la figura 4.7, está compuesto por dos tipos de habitaciones:

- Habitaciones enumeradas: estas habitaciones son aquellas en las que aparecerán enemigos cuando los jugadores intenten atravesarlas. En el mapa diseñado, hay un total de 10 habitaciones de este tipo, con un número de enemigos distinto en cada sala. Además, los enemigos que pueden aparecer en estas salas son pseudoaleatorios. En Unity, cada sala queda representada por un GameObject, con un *script* de C# asociado. Este *script* contiene una variable, que almacena un número del 1 al 3. Este número indica los enemigos que pueden aparecer en esta sala, obedeciendo la siguiente regla:
    - Si el valor de este parámetro es igual a 1, solo pueden aparecer enemigos del tipo 1, los enemigos cuerpo a cuerpo.
    - Si el valor de la variable es igual a 2, podrán aparecer enemigos de los tipos 1 y 2.
    - Si su valor es 3, se pueden instanciar todos los tipos de enemigos.

  Aunque los enemigos que aparecen en cada sala son aleatorios, el número de enemigos que aparecen por sala siempre es el mismo, dado que las posiciones en las que pueden instanciarse siempre son las mismas. Además, no se ha contemplado permitir que en una de esas posiciones no aparezcan enemigos, aunque esto sería un cambio sencillo de implementar. Adicionalmente, las salas pueden estar compuestas de varias oleadas de enemigos. Para ello, el sistema comprueba el número de oleadas asignadas a esa habitación, instanciando en primer lugar los enemigos de la primera oleada. Una vez el jugador ha destruido todos los enemigos de la primera oleada, se procede a instanciar los de la segunda oleada, así consecutivamente, hasta que se instancian los de la última oleada.

  Para que comience el proceso de aparición de los enemigos, es necesario que el jugador atraviese un punto determinado de las salas. Estos puntos están definidos por un *collider*, o colisionador. Cuando la *hitbox* de un jugador entra en contacto con uno de estos *colliders*, se cierran las puertas de la habitación y se empiezan a instanciar los enemigos. En la figura 4.8, se puede apreciar el *collider* de la habitación 0 del mapa. Cada *collider* se representa en Unity como un polígono o curva, cuyas aristas o arcos están pintados en un tono verde claro.

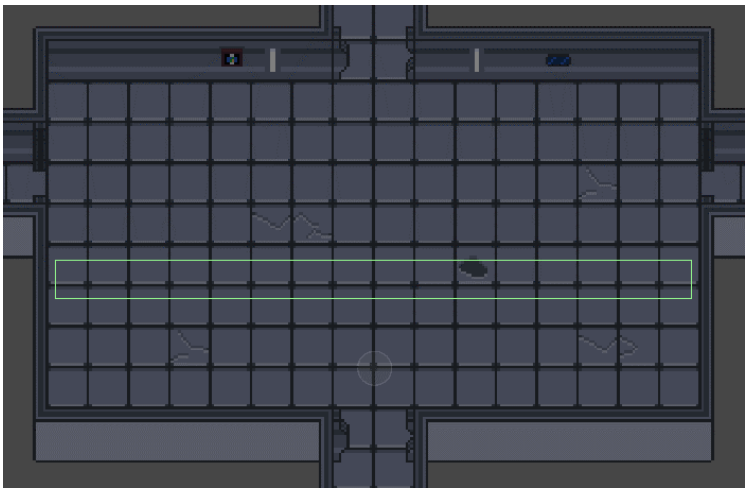

Fig. 4.8. Habitación 0 con su *collider* correspondiente

Para controlar que no se vuelva a ejecutar este proceso cada vez que el jugador colisiona con el *collider*, la primera vez que se produce una colisión, este se desactiva durante el resto de la partida.
Cuando el jugador ha derrotado a todos los enemigos de la sala, el sistema abre las puertas de la sala, permitiendo al jugador seguir explorando la mazmorra.

- Habitaciones resaltadas en verde: estas habitaciones representan las zonas seguras de la mazmorra, como la habitación en la que aparecen por primera vez los jugadores, zonas en las que se pueden encontrar botiquines para curarse, o la habitación en la que está la meta para ganar la partida. A continuación, se describen las tres variantes de zonas seguras:
    - Habitación inicial: es la primera zona en la que se instancian los jugadores. Esta habitación no contiene enemigos, permitiendo a los jugadores familiarizarse con los controles antes de entrar en combate.
    - Zonas con botiquín: son dos habitaciones, más pequeñas que el resto de las habitaciones encontradas en la mazmorra, que contienen un botiquín en el centro.
    - Habitación objetivo o meta: es una habitación que contiene un cofre en el centro. Cuando el jugador colisione con este cofre, el juego cambiará al estado S3, dando la partida por finalizada. En la imagen 4.9 se puede observar esta habitación y la apariencia del cofre que representa la meta.

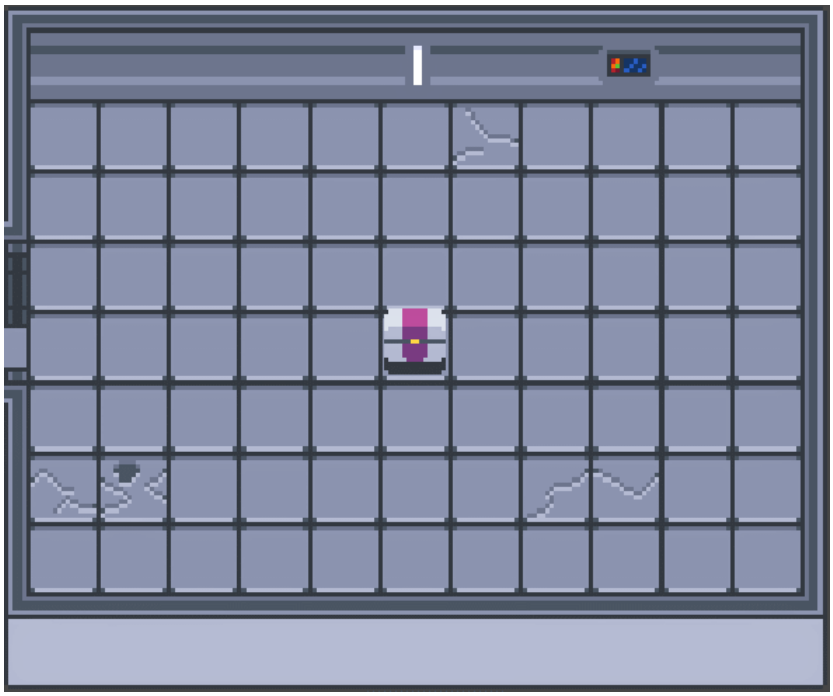

Fig. 4.9. Disposición de la habitación objetivo y el cofre que contiene

4.2.4. Flujo de escenas y menús

Los proyectos de Unity están formados por escenas. Estas escenas contienen todos los objetos relevantes para ese escenario concreto, por lo que normalmente, los proyectos se suelen organizar en varias escenas.

En este proyecto se han diseñado un total de cinco escenas, cada una de ellas con una función específica. Estas escenas son las que se describen a continuación:

- *Main Menu Scene.* Esta es la escena que se carga nada más iniciar la aplicación, mostrando el título del juego y tres opciones:
    - *Single Player*: este botón permite al jugador iniciar una partida en solitario, evitando toda la lógica relevante a la creación de una sala para jugar en multijugador y cargando directamente la escena del juego.
    - *Multiplayer*: este botón deriva al jugador a la escena *Lobby Scene*, permitiendo al jugador iniciar una partida multijugador.
    - *Quit game*: al usar este botón, se cierran todos los procesos asociados a la aplicación.
- *Loading Scene*. Se utiliza como escena intermedia mientras el sistema realiza la transición de las escenas. Esto ofrece al jugador retroalimentación cuando se está cargando una nueva escena, puesto que Unity, cuando carga una escena, muestra por pantalla el último fotograma dibujado de la escena origen, hasta que se carga en su totalidad la escena destino.
- *Lobby Scene*. Esta escena ofrece al jugador dos opciones:
    - *Create game*: cuando se presiona este botón, el sistema crea un nuevo *lobby* o sala de juego, a través del servicio Lobby, de Unity Game Services. Tras crear una sala, el juego sistema cargará la escena *Player Selection Scene*.
    - *Join game*: tras pulsar este botón, el sistema tratará de unirse al *lobby* asociado al código introducido en la caja de texto, situada inmediatamente encima de este botón. Si el jugador ha introducido un código incorrecto, la sala está llena, o ha habido algún error, el sistema muestra un mensaje de error al jugador, con un botón que le permite volver al menú principal. En caso de no ocurrir ningún error, el sistema avanzará a la escena *Player Selection Scene.*
- *Player Selection Sce*ne. Esta escena permite al jugador elegir un nombre para su jugador que, actualmente, solo tiene una función estética. Asimismo, muestra al jugador el código del *lobby*, permitiendo al jugador comunicárselo a la persona que quiere que se una a su partida. Además, en la escena se incluye un botón para volver al menú principal y otro botón, que permite al jugador establecer su estado a listo. El sistema solo cambiará a la escena *Level 1 Scene* cuando todos los jugadores presionen este botón. Sin embargo, esto tiene una implicación. Cuando solo hay un jugador en la escena, el sistema avanzará a la escena de juego igualmente, por lo que, actualmente, es necesario que el anfitrión espere a que el segundo jugador se haya unido a su sala.
- *Level 1 Scene*. Esta escena se corresponde a la escena de juego o partida, en la que ocurren todos los sucesos explicados en el epígrafe 4.2.3. Contiene todos los objetos relevantes a una partida, como la disposición de la mazmorra, obstáculos, jugadores y el resto de lógica asociada.

Como se puede apreciar, el nombre de esta escena indica que se trata del nivel 1 de la partida. Esto se debe a que, en los comienzos, se pensó en diseñar más de un nivel, pero por falta de tiempo se acabó desestimando. No obstante, esto se trata más adelante, en las conclusiones del proyecto y líneas de trabajo futuras.

4.2.5. Diseño del multijugador

Antes de hablar sobre el diseño del multijugador de este prototipo, es necesario hablar de varios temas relacionados con la forma en la que se implementa este en Unity. Estos dos temas se comentan en los siguientes subapartados.

4.2.5.1. Propiedad y autoridad sobre los objetos

En Unity, la propiedad sobre un objeto de red se refiere a qué actor en específico tiene el control sobre dicho objeto, pudiendo cambiar su estado y propagando estos cambios al resto de clientes de la red. En el caso de la autoridad, se refiere a qué entidad de la red tiene permiso para tomar decisiones finales sobre el objeto de red, como su creación o destrucción, o un cambio de propietario [74].

En las arquitecturas Cliente-Servidor y derivadas, quien tiene autoridad y propiedad sobre todos los objetos de red es el servidor, siendo este el único responsable de realizar cambios sobre ellos, como su instanciación, modificación o destrucción, por lo que en el caso de que un cliente quiera realizar un cambio sobre un objeto de red, es necesario que este envíe una petición al servidor y que este ejecute los cambios. En una topología de autoridad distribuida, la autoridad siempre la tiene el propietario de la sesión, mientras que la propiedad de los objetos de red se puede transferir entre los distintos clientes conectados de manera dinámica.

4.2.5.1.1. Autoridad del servidor vs. Autoridad del cliente

En Unity, normalmente es el servidor quien posee la autoridad sobre los objetos de la escena. No obstante, es posible cambiar la autoridad de un objeto a un cliente, por ejemplo, para el objeto de red que represente al jugador asociado. Esto tiene ciertas ventajas y desventajas, como se va a observar en la tabla 4.1 [75]:

TABLA 4.1.
DIFERENCIAS ENTRE AUTORIDAD DEL SERVIDOR Y AUTORIDAD DEL CLIENTE

	Autoridad del servidor	**Autoridad del cliente**
Seguridad	Mayor seguridad. Al ser todas las peticiones de los clientes verificadas por el servidor, se evita cualquier modificación posible que pueda afectar al entorno de juego.	Menor seguridad. No se verifican las acciones del cliente, si no que se sincronizan al servidor directamente los cambios. Esto implica que cualquier cliente puede modificar el estado del juego. Por esta razón, su uso no está aconsejado en juegos donde hay un carácter competitivo o existe el *Player versus Player* (jugador contra jugador).

	Autoridad del servidor	Autoridad del cliente
Respuesta	Menor capacidad de respuesta. Como todas las acciones tienen que ser verificadas por el servidor, esto implica que, por ejemplo, si un cliente quiere mover su jugador, tiene que realizar una petición al servidor y que este realice el cambio, sincronizando posteriormente los cambios realizados por el servidor. Esto se traduce en un aumento del tiempo de respuesta y si la conexión con el servidor no es estable, puede dar lugar a problemas de sincronización por parte del cliente. No obstante, el resto de los clientes verán los cambios reflejados correctamente.	Mayor capacidad de respuesta. Al no tener que realizar una petición al servidor para realizar una acción, el cliente que la ejecuta observa los cambios realizados de manera instantánea. Esto se traduce en una menor latencia por parte del cliente, lo que es ideal en juegos que no sean competitivos y se desee que los clientes tengan una mayor capacidad de respuesta.
Sincronización	No existen problemas de sincronización. El servidor es quién ejecuta todas las acciones, por lo que, si un cliente tiene una conexión inestable con el servidor, el resto de los clientes no se verán afectados y se seguirán sincronizando todos los cambios.	Pueden existir problemas de sincronización. En el caso de que un cliente no tenga una conexión estable con el servidor, los cambios que realice este cliente no se sincronizarán adecuadamente con el servidor, por lo que el resto de los clientes no observarán el mismo comportamiento que el cliente local que está sufriendo la inestabilidad de red.

4.2.5.2. Soluciones para implementar el multijugador en Unity

En la actualidad, Unity dispone de dos soluciones oficiales para la implementación del multijugador, siendo estas Netcode for GameObjects y Netcode for Entities. Además, incluye otras implementaciones desarrolladas por la comunidad o empresas privadas. En este proyecto, solo se han valorado las soluciones de Fish-Networking y Mirror.

Netcode for GameObjects, en adelante NGO, es una biblioteca oficial de alto nivel desarrollada por Unity, que permite al desarrollador enviar datos del mundo y de los objetos de la sesión, en Unity llamados *GameObjects* (en castellano, objetos del juego), a través de una red a todos los jugadores de la sesión. Actualmente, está disponible en todas las plataformas de escritorio, móviles y realidad extendida, además de la mayoría de plataformas cerradas, como las consolas [76]. Esta solución ha sido la elegida en el desarrollo de la práctica, al ser una solución con soporte oficial y una documentación muy detallada.

Por otro lado, Unity ofrece la biblioteca de Netcode for Entities. Esta biblioteca, que forma parte de Unity DOTS (*Data-Oriented Technology Stack*), depende fundamentalmente de la arquitectura ECS (*Entity Component System*) de Unity. Esta

arquitectura emplea entidades, o *entities*, como unidades discretas para representar un conjunto de datos, a diferencia de las unidades que utiliza por defecto Unity, los *GameObjects*. Al contrario de estos últimos, una entidad actúa como un identificador agrupando componentes únicos, en vez de contener directamente código o servir como un contenedor asociado a otros componentes, como es en el caso de los *GameObjects* [77], [78].

A continuación, se muestra la tabla 4.2, una tabla resumen con las diferencias entre estas dos soluciones.

TABLA 4.2.
DIFERENCIAS ENTRE NGO Y NETCODE FOR ENTITIES

	Netcode for GameObjects	**Netcode for Entities**
Arquitecturas de red soportadas	Por defecto, NGO trabaja como una topología Cliente-Servidor, pero también permite utilizar topologías Cliente-Anfitrión y arquitecturas de autoridad distribuida.	Solo soporta topologías Cliente-Servidor y Cliente-Anfitrión.
Unidades sobre las que trabaja	Utiliza los componentes por defecto de Unity, los denominados *GameObjects* y sus componentes, como *Rigidbody*, *SpriteRenderer*, etc.	Emplea las *entities* del ECS y componentes especiales, como sistemas de físicas.
Complejidad	Funciona exactamente igual que el resto de los componentes estándar de Unity, por lo que no es necesario realizar ningún cambio significativo en la forma de trabajar.	Requiere saber utilizar el ECS de Unity y depende directamente de este, por lo que cambia completamente la manera de trabajar.
Juegos objetivo	Se recomienda para juegos de pequeña a mediana escala, como juegos cooperativos. No obstante, también es capaz soportar juegos multijugador de una escala mayor.	Es recomendable para juegos de gran escala y juegos competitivos que requieran de métodos de predicción del cliente.

En cuanto a las bibliotecas desarrolladas por entidades externas a Unity, se ha realizado un estudio de dos de ellas, expuestas a continuación:

- En primer lugar, se habla de Mirror Networking, una solución de código abierto creada por la comunidad que nació a raíz UNET, una solución para la gestión del multijugador lanzada por Unity en 2015 y que en la actualidad está obsoleta. Mirror nació a raíz de la falta de soporte que Unity daba a UNET y al descontento de la comunidad, solucionando diversos problemas que tenía UNET una vez se liberó cierta parte del código, la parte de alto nivel (HLAPI). Una vez Unity dejó de dar soporte a UNET, Mirror creció como una alternativa totalmente distinta, ofreciéndose el día de hoy como

una solución alternativa para la implementación del multijugador en videojuegos desarrollados en Unity [79].

- Fish-Networking es una solución desarrollada por la compañía First Gear Games. Es una solución que tiene como objetivo proporcionar flexibilidad, eficacia, facilidad de uso y fiabilidad. Incorpora ciertas soluciones como predicción en la parte del cliente, compensación de latencia y soporte para diversos tipos de transporte y servidores de retransmisión [80].

4.2.5.3. Implementación en el prototipo

El prototipo desarrollado en este trabajo de fin de grado utiliza el paquete de Netcode for GameObjects (NGO) para implementar la sincronización de los datos de una sesión multijugador. El primer paso para utilizar NGO en un proyecto de Unity es crear un objeto que contenga el componente NetworkManager. Este contiene todas las configuraciones relevantes a NGO y a la sesión de red, por tanto, es necesario crearlo antes de trabajar con NGO. Por esta razón, este objeto aparece por primera vez en el flujo del juego cuando el jugador accede a la escena *Lobby Scene*. En esta escena, además, está situada la lógica necesaria para la conexión de un jugador a un *lobby*, así como la configuración necesaria para que la máquina del usuario se conecte a un servidor de retransmisión del servicio Relay, lo que le permite jugar en línea con otros jugadores a través de Internet. En este momento, además, es donde se define qué jugador es el anfitrión y cuál es el cliente. El jugador que crea una sala de juego será el anfitrión de la partida y, por tanto, será quien actúe a su vez como servidor y cliente. Un usuario que se une a un lobby solo tiene el cargo de cliente de la sesión, dependiendo este de la conexión con el *host* (anfitrión). Una vez establecida la conexión del cliente al anfitrión, y cuando los dos jugadores han cambiado su estado a listo en la escena *Player Selection Scene*, la escena del nivel 1 cargará, manteniendo los ajustes de la sesión establecida.

A continuación, se va a hablar de los distintos elementos de la escena a sincronizar, ya que, en cada caso, puede haber ligeras variaciones en la manera de sincronizar los elementos de la escena, según las necesidades que tenga ese objeto.

Game Manager

La lógica del *Game Manager* se refiere a toda aquella lógica que es común al sistema del juego. Es decir, el *Game Manager* se encarga, principalmente, de inicializar las salas, generar los enemigos en las salas cuando un jugador entra a una, sincronizar el estado actual del juego o instanciar a los jugadores una vez comienza la partida, por ello, es lógica que se debe ejecutar una sola vez.

Como se comentaba anteriormente, en Unity, la autoridad de todos los objetos de red la posee el servidor. Por esta razón, el servidor es el único que puede instanciar nuevos objetos de red en una escena. En Unity, y más específicamente en Netcode for GameObjects, todos los objetos de red necesitan tener un componente NetworkObject, que les asigna una ID de red en la sesión, además de necesitar, al menos, un componente NetworkBehavior [81].

Debido a lo explicado en los dos párrafos anteriores, en el diseño del multijugador se ha decidido que todo el código referente al *Game Manager* lo ejecute el servidor. El cliente solo se comunicará con el servidor a través de llamadas a procedimientos remotos (RPC) y variables de red, en Unity, llamadas NetworkVariables. Estos dos métodos de sincronización se explican a continuación.

- Llamada a procedimiento remoto: una llamada a un procedimiento remoto, en adelante RPC por sus siglas en inglés, *Remote Procedure Call*, es una llamada que realiza un proceso para comunicarse con otro proceso que está localizado en una ubicación remota. En sí, un RPC no ejecuta ninguna lógica importante, sino que es el procedimiento remoto el que se encarga de ejecutar la lógica relevante. En la versión de NGO utilizada, la 1.5.2, Unity define dos tipos de RPC, según el rol que desempeñe la máquina remota [82]:
    - *Server* RPC: representa una llamada remota a un procedimiento que debe ejecutar el servidor. En Unity, los *Server* RPC suelen ejecutarlos los clientes, para pedir al servidor que ejecute una lógica que solo puede ejecutar este, como la instanciación de objetos de red o la destrucción de los mismos. No obstante, también es posible que se utilicen para el resto de lógica del juego, en caso de querer que el videojuego sea controlado completamente por el servidor, o *server-authoritative*.
    - *Client* RPC: son aquellos RPC que tienen que ejecutar todos los clientes de la sesión. Esto puede corresponderse a cambios visuales en la escena y objetos que no tienen el componente NetworkObject, principalmente. Es importante notar que, como el anfitrión ejerce el rol de cliente y el de servidor, responde a ambos tipos de RPC.
- NetworkVariable: las NetworkVariables son variables de red que sincronizan su valor automáticamente entre todos los integrantes de la red, sin necesidad de utilizar un RPC para actualizar a los otros usuarios de la red de los cambios producidos sobre la variable. Por defecto, el servidor tiene permisos de lectura y escritura, mientras que los clientes solo pueden leer el valor de la NetworkVariable, necesitando realizar una llamada remota al servidor en caso de querer cambiar su valor [83].

Una vez descritos los dos métodos disponibles para sincronizar datos entre los integrantes de una red, se puede justificar la razón por la que se ha decidido que toda la lógica del Game Manager la ejecute el servidor. Las funcionalidades que tiene el servidor son:

- Estado actual de la partida. El estado de la partida se ha definido como una variable de red, por lo que el cliente puede leer en cualquier momento el estado de esta. Sin embargo, es el servidor el que realiza todas las comprobaciones relevantes a la lógica del estado actual del juego, efectuando él mismo cualquier cambio en el valor de esta variable.
- Inicialización de las salas. Dado que es el servidor quien genera los objetos de red, como los enemigos o botiquines, es irrelevante para el cliente

consumir recursos en la inicialización de las salas. Este proceso consiste en "rellenar" cada sala con los enemigos que aparecerán una vez el jugador entre en ellas.

- Instanciación de los personajes jugables. En NGO, aquellos objetos que sean propiedad de los jugadores, como sus personajes jugables, deben generarse en tiempo de ejecución. Como es el servidor el único que puede instanciar nuevos objetos en tiempo de ejecución (*prefabs*), esta tarea solo debe ejecutarla en anfitrión.
- Generación de enemigos cuando un jugador entra a una sala. Al igual que la inicialización de las salas, la lógica relacionada a la generación de enemigos y oleadas solo la realiza el servidor. No obstante, en este caso se utilizan *client* RPCs para sincronizar el estado de las puertas de la mazmorra, dado que el mapa en sí no es un objeto de red, por lo que es necesario que el servidor realice esta llamada remota a todos los clientes, conmutando el estado de las puertas entre cerradas o abiertas.

Personajes de los jugadores

A continuación, se detalla la forma en la que se han sincronizado cada una de las mecánicas o interacciones que pueden realizar los jugadores.

- Movimiento del personaje. Para poder sincronizar la posición de un objeto de red con NGO, es necesario emplear el componente NetworkTransform. Este elemento se encarga de sincronizar de manera automática la posición, rotación y escalado del objeto al que está asociado, además de los hijos de dicho objeto. Sin embargo, NGO por defecto utiliza un modelo *server-authoritative*, por lo que es el servidor el único integrante de la sesión que puede realizar cualquier cambio en la posición, rotación o escala del objeto. No obstante, aunque esto es útil en videojuegos competitivos y similares, puesto que evita cualquier modificación por parte de los clientes, en un videojuego cooperativo como este solo añade diversos problemas, como un mayor tiempo de respuesta o retardo cuando el cliente intenta mover su personaje. Esto es aún más notable cuando la conexión de red es inestable. Por ello, se ha decidido seguir un modelo *client-authoritative* para el movimiento de los jugadores. Realizando este cambio, es el cliente el que ejecuta toda la lógica relevante al movimiento del jugador, siendo capaz de ver el resultado de su acción inmediatamente cuando se realiza la señal de entrada correspondiente.

  En este apartado también es necesario hablar de la capacidad de apuntado del jugador, puesto que los objetos sobre los que se realizan los cambios forman parte del personaje del jugador. Esta acción también la realiza el cliente directamente en su máquina, dado que, si la ejecutase el servidor, produciría una demora en la capacidad de apuntado del jugador, disminuyendo la fluidez de sus acciones.
- Habilidad de evasión. Como la autoridad del movimiento del jugador la tienen los clientes, la aplicación de una fuerza a su personaje la realiza el propio

jugador desde su máquina, sincronizando a través de la red los cambios en su posición.

- Capacidad de disparar proyectiles. Para esta mecánica, es el cliente el que leerá la entrada que produzca el jugador en su máquina correspondiente, comprobando, además, que el disparo solo se pueda ejecutar cuando esta capacidad no esté en enfriamiento. No obstante, como el servidor es el único que puede instanciar los proyectiles en tiempo de ejecución, se tiene que realizar una llamada remota al servidor, quien genera el proyectil como un objeto de red.
- Recibir daño y curarse. En este caso, el servidor primero comprueba que el jugador pueda ser dañado o curarse, realizando posteriormente una llamada a los clientes para que ejecuten la acción, actualizando la vida del jugador en el proceso.
- Animaciones. Las animaciones de movimiento de los jugadores se sincronizan en tiempo real, mediante la utilización del componente NetworkAnimator. Del mismo modo que NetworkTransform, por defecto este componente sigue un modelo *server-authoritative*, lo que impide al cliente controlar las animaciones de su personaje. Sin embargo, con un cambio sencillo puede indicarse a Unity que estas animaciones las controlen directamente los propietarios del objeto, permitiendo al cliente (y propietario correspondiente) cambiar las animaciones de su personaje, siendo estas sincronizadas al resto de integrantes de la sesión de red.

Enemigos

En el caso de los enemigos, es el servidor quien tiene la propiedad y autoridad de estos, por lo que toda la lógica de los enemigos se ejecuta en el servidor, incluyendo la persecución de los jugadores, ser dañado, destrucción, etcétera.

Botiquines

Toda la lógica correspondiente a los botiquines la tiene que ejecutar el servidor, dado que son objetos de red instanciados en tiempo de ejecución. No obstante, como los jugadores tienen que disparar a los botiquines para curarse, es necesario que sus proyectiles los identifiquen. Para ello, se emplean etiquetas, o *tags* de Unity. Cuando un jugador realiza un disparo, el proyectil disparado tiene una etiqueta que lo asocia a ese jugador, permitiendo al sistema saber qué usuario ha tratado de curarse. Toda la lógica asociada a los proyectiles, la ejecuta el servidor, ya que este es el propietario de estos objetos de red.

Efectos visuales y sonoros

Durante el diseño de este prototipo, se ha decidido que los efectos visuales y de sonido no sean objetos de red. Al ser efectos estéticos, no existe ningún inconveniente si estos efectos no están correctamente sincronizados. Esto alivia también la carga del servidor, ya que la instanciación de estos objetos la pueden realizar los clientes, distribuyendo así la carga de trabajo. Lo único que tiene que efectuar el servidor es la llamada a los

procedimientos remotos correspondientes cuando los clientes deban ejecutar estos efectos.

5. EVALUACIÓN

En este capítulo se incluye el plan de pruebas utilizado para comprobar el funcionamiento del prototipo implementado, además de verificar el cumplimiento de los requisitos especificados en el análisis del proyecto. Asimismo, se especifican problemas encontrados en la implementación del proyecto y posibles soluciones a estos.

5.1. Plan de pruebas

Para la ejecución del plan de pruebas se ha empleado una serie de tablas, idénticas a la tabla 5.1, que se utiliza como plantilla para cada caso de prueba.

TABLA 5.1.
PLANTILLA DE LOS CASOS DE PRUEBA

Identificador	
Descripción	
Resultado esperado	
Resultado obtenido	
Requisitos asociados	
Estado	

En esta platilla correspondiente a la tabla 5.1, cada una de las de las columnas contiene los siguientes datos:

- Identificador: esta columna contiene el identificador del caso de uso, que sigue el siguiente formato:

  CP-XX

  Donde:
  - CP indica que es un Caso de Prueba
  - XX indica el número identificador del caso de prueba.
- Descripción: breve descripción de lo que comprueba el caso de prueba.
- Resultado esperado: respuesta esperada de la ejecución del caso de prueba, en base a los requisitos asociados a este.
- Resultado obtenido: respuesta que se ha obtenido de la ejecución de la prueba en el prototipo desarrollado.
- Requisitos asociados: lista de requerimientos que se cotejan con el caso de prueba.
- Estado: resultado de la ejecución del caso de prueba. Puede contener dos valores:
  - OK: el caso de prueba se ha validado correctamente, dado que el resultado obtenido corresponde al esperado. El color de la celda, además, se colorea de verde.

- OK PARCIAL: el resultado obtenido es similar al deseado, pero presenta alguna discrepancia con este. Adicionalmente, la celda se colorea de naranja.
- KO: el resultado obtenido es distinto al esperado. La celta se colorea en rojo para expresar este valor.
  Adicionalmente, cuando el estado tiene el valor de "OK PARCIAL" o "KO", se crea un identificador del problema asociado, que sigue el formato:

  P-XX

  Donde XX indica el número de identificación del problema.

TABLA 5.2.
CP-01

Identificador	CP-01
Descripción	Comprobación del movimiento del jugador y la respuesta del sistema ante colisiones.
Resultado esperado	El jugador se puede mover en cualquier dirección del plano XY. El personaje del jugador no puede atravesar paredes u obstáculo que haya situados por el mapa.
Resultado obtenido	El jugador se puede mover en cualquier dirección. Además, cuando colisiona contra una pared, puerta u obstáculo, el personaje deja de poder avanzar en ese sentido.
Requisitos asociados	F-U-C01 F-S-R02
Estado	OK

TABLA 5.3.
CP-02

Identificador	CP-02
Descripción	Comprobación de la habilidad de esquiva del jugador.
Resultado esperado	El jugador puede realizar una evasión en la dirección en la que se está moviendo. Esta habilidad tiene un enfriamiento de 2 segundos.
Resultado obtenido	Cuando se presiona el espacio, el personaje realiza una evasión en función de su vector de movimiento. Para poder ejecutar la habilidad, es necesario que transcurran 2 segundos desde la anterior ejecución.
Requisitos asociados	F-U-C03 F-S-R04
Estado	OK

TABLA 5.4.
CP-03

Identificador	CP-03
Descripción	Prueba de apuntado y disparo del personaje del jugador
Resultado esperado	El jugador puede apuntar el arma de su personaje en cualquier dirección. Además, puede disparar proyectiles desde su arma a una cadencia de fuego determinada.
Resultado obtenido	El arma del personaje del jugador apunta en la dirección en la que se encuentra el cursor. Cuando se pulsa o mantiene pulsado el botón izquierdo del ratón, el arma del jugador dispara un proyectil en la dirección en la que se está apuntando. El sistema controla adecuadamente la cadencia de disparo, limitándola a 2 disparos por segundo. Cuando se realiza esta prueba para el cliente, se puede observar cómo, los proyectiles se "teletransportan" pequeñas distancias, al estar sincronizando su posición en tiempo real. Mientras que el anfitrión ve cómo los proyectiles siguen su trayectoria adecuadamente, el cliente observa estas acciones con cierto retardo, empeorando ligeramente la experiencia de juego.
Requisitos asociados	F-U-C05 F-U-C06 F-S-R07
Estado	OK PARCIAL. Problema asociado: P-01.

TABLA 5.5.
CP-04

Identificador	CP-04
Descripción	Los botiquines curan al jugador correspondiente, no sobrepasando los puntos de vida máximos.
Resultado esperado	Cuando un jugador impacta un botiquín con uno de sus proyectiles, el botiquín le cura parte de la vida máxima. No obstante, si su vida está al completo, el sistema controla que no se supere la vida máxima.
Resultado obtenido	Una vez el proyectil de uno de los jugadores impacta en un botiquín, el jugador se cura una cantidad de daño establecida (en este caso, 1 punto de vida). Esta diferencia se puede observar mediante la interfaz de usuario presente en el juego, donde se ve cómo aumenta la barra de salud correspondiente. Si el jugador tiene su vida al completo, el botiquín no le devuelve salud, pero sí disminuye la salud que puede restaurar el botiquín, destruyéndose si su valor disminuye a 0.
Requisitos asociados	F-S-C08 F-U-C09 F-S-R10 F-S-C37
Estado	OK

TABLA 5.6.
CP-05

Identificador	CP-05
Descripción	Comprobación de la mecánica de daño y muerte del jugador por ataques enemigos.
Resultado esperado	Cuando un enemigo (de cualquier tipo) ataca al jugador, su personaje recibe daño, ofreciendo una respuesta visual y sonora al jugador. Además, no puede ser dañado dos veces en un mismo instante. Cuando la vida del jugador llega a 0, este muere y se acaba la partida.
Resultado obtenido	Los enemigos dañan al jugador, ya sea con proyectiles o cuerpo a cuerpo, dependiendo del tipo de enemigo. Cuando se produce este suceso, la vida del jugador disminuye la cantidad correcta, observándose el cambio en la barra de vida del jugador, además que el personaje del jugador recibe una fuerza de reacción. El jugador no es dañado dos veces si dos proyectiles impactan contra él simultáneamente. Además, cuando su vida llega a 0, su personaje desaparece de la escena y termina el juego, dando por perdida la partida y mostrando esta condición al jugador mediante un menú, que le permite volver al menú principal. Cuando el jugador que se daña es el cliente, en algunos casos el sistema no aplica una fuerza de reacción a su personaje.
Requisitos asociados	F-S-C08 F-S-C11 F-S-C12 F-S-R13 F-S-C14 F-U-C15 F-U-C17 F-S-C23 F-U-C24 F-S-C37 F-U-C38 F-U-C39
Estado	OK PARCIAL. Problema asociado: P-02.

TABLA 5.7.
CP-06

Identificador	CP-06
Descripción	Verificación de que el jugador puede ganar la partida.
Resultado esperado	Cuando el personaje de un jugador toca el cofre, la partida acaba, mostrando al jugador un texto explicando la razón de finalización y permitiéndole volver al menú principal.
Resultado obtenido	Cuando uno de los jugadores entra en contacto con el cofre que representa la meta, el sistema muestra un menú, dándole la enhorabuena. Este menú incluye un botón que le permite volver al menú principal.
Requisitos asociados	F-U-C15 F-S-C16 F-U-C17 F-U-C39
Estado	OK

TABLA 5.8.
CP-07

Identificador	CP-07
Descripción	Validación del menú de pausa y volver al menú principal.
Resultado esperado	El jugador puede abrir un menú de pausa durante la partida, que le permite volver al menú principal en cualquier momento.
Resultado obtenido	Mediante la pulsación de la tecla “Escape”, el sistema muestra un menú, permitiendo al jugador resumir la partida (cerrar el menú) y volver al menú principal. Si se pulsa el segundo botón, la escena cambia al menú principal, terminando la partida anterior. Además, si el jugador vuelve a iniciar una partida, el juego empieza desde cero.
Requisitos asociados	F-U-C17 F-S-R18
Estado	OK

TABLA 5.9.
CP-08

Identificador	CP-09
Descripción	Comprobación de que los enemigos pueden ser dañados y mueren cuando sus puntos de vida se agotan.
Resultado esperado	Cuando el proyectil de un jugador alcanza a un enemigo, este sufre daños, reduciendo sus puntos de vida en base a los puntos de ataque del jugador. Cuando sus puntos de vida llegan a 0, el enemigo muere, desapareciendo de la escena, reproduciendo un sonido y un efecto visual.
Resultado obtenido	El impacto del proyectil de un jugador en un enemigo provoca que el sistema reproduzca un sonido y muestre un efecto visual sobre el enemigo dañado. Cuando se realizan el número de ataques necesario para destruir al enemigo en base al daño del jugador, el enemigo desaparece de la escena, dejando en su lugar unas partículas y el efecto visual de una explosión.
Requisitos asociados	F-S-C08 F-S-C19 F-U-C20 F-S-C21 F-S-R22
Estado	OK

TABLA 5.10.
CP-09

Identificador	CP-09
Descripción	Validación del *pathfinding* de los enemigos.
Resultado esperado	Si un jugador entra en el radio de detección de un enemigo, este último empieza a perseguir al jugador, buscando el camino más corto hasta su objetivo.
Resultado obtenido	Cuando el jugador está a una distancia superior a la establecida como rango de detección del enemigo, el enemigo no persigue al jugador. No obstante, cuando el personaje jugable entra en el radio de detección, el enemigo empieza a perseguir al jugador, a través de un camino óptimo y evitando los obstáculos de la escena.
Requisitos asociados	F-S-C25 F-S-R26
Estado	OK

TABLA 5.11.
CP-10

Identificador	CP-10
Descripción	Verificación del funcionamiento del modo un jugador.
Resultado esperado	El jugador puede iniciar, desde el menú principal, una partida en el modo un jugador. Además, si desconecta la conexión a Internet, puede seguir jugando en este modo de juego.
Resultado obtenido	Al desactivar el adaptador de red, el jugador puede iniciar una partida con el botón para jugar en el modo un jugador. En ese momento, el sistema carga la escena de juego.
Requisitos asociados	F-U-C27 F-U-C28
Estado	OK

TABLA 5.12.
CP-11

Identificador	CP-11
Descripción	Comprobación del funcionamiento del botón de cerrar el juego.
Resultado esperado	El usuario puede cerrar la aplicación por completo desde el menú principal, al pulsar el botón correspondiente.
Resultado obtenido	Al pulsar el botón “Quit game”, finaliza la ejecución de la aplicación.
Requisitos asociados	F-U-C29
Estado	OK

TABLA 5.13.
CP-12

Identificador	CP-12
Descripción	Comprobación del funcionamiento del botón del modo multijugador.
Resultado esperado	Cuando el usuario acciona el botón del modo multijugador, el sistema abre otro menú, que permite crear una sala o unirse a una existente.
Resultado obtenido	Al pulsar el botón del multijugador, se carga el menú correspondiente, mostrando al jugador dos botones, uno para crear un *lobby* y otro para unirse a uno existente.
Requisitos asociados	F-U-C30 F-S-C31
Estado	OK

TABLA 5.14.
CP-13

Identificador	CP-13
Descripción	Comprobación de la creación de una sala de juego.
Resultado esperado	Cuando el jugador presiona el botón para crear un *lobby*, el sistema le garantiza una sala, cambiando el menú al correspondiente, donde le mostrará el código de la sala.
Resultado obtenido	Al pulsar el botón de crear la sala, el sistema, tras un breve instante, cambia la escena a la correspondiente. En esta escena, el jugador puede cambiar su nombre, volver al menú principal, o copiar el código de la sala para que otro jugador se una al *lobby* que acaba de crear.
Requisitos asociados	F-S-C32 F-S-C33
Estado	OK

TABLA 5.15.
CP-14

Identificador	CP-14
Descripción	Comprobación de la unión correcta de un jugador a una sala creada por otro usuario.
Resultado esperado	Cuando el jugador introduce un código de sala correcto, el sistema le permite unirse a esa sala, cambiando la escena a una en la que se le muestra el código de sala correspondiente.
Resultado obtenido	Una vez el usuario introduce el código de la sala creada por otro jugador, el usuario se une a dicha sala, cambiando la escena a la pertinente, mostrándole el código de identificación de la sala.
Requisitos asociados	F-S-C34
Estado	OK

TABLA 5.16.
CP-15

Identificador	CP-15
Descripción	Verificación de que un jugador no se puede unir a una sala inexistente.
Resultado esperado	Si un jugador introduce el código de una sala inexistente, recibe por pantalla un mensaje de error, indicándole que no se pudo unir a esa sala.
Resultado obtenido	Cuando el jugador introduce un código de sala erróneo, el sistema le muestra un error, permitiéndole volver al menú principal.
Requisitos asociados	F-S-C34
Estado	OK

TABLA 5.17.
CP-16

Identificador	CP-16
Descripción	Comprobación de que un jugador no se puede unir a una sala que está llena.
Resultado esperado	Si un jugador intenta unirse a una sala que está llena, recibe un mensaje de error.
Resultado obtenido	Cuando un tercer jugador intenta unirse a una sala llena, el sistema lanza un mensaje de error al jugador, informándole de que no se ha podido unir a ese *lobby*.
Requisitos asociados	F-S-C34 F-S-R42
Estado	OK

TABLA 5.18.
CP-17

Identificador	CP-17
Descripción	Comprobación del funcionamiento del estado de listo de los jugadores.
Resultado esperado	Los jugadores pueden establecer su estado a listo cuando se han unido a una sala. Cuando todos los jugadores establecen su estado a este, el juego avanza a la escena del nivel 1, dando comienzo a la partida.
Resultado obtenido	En una sala con dos jugadores dentro y habiendo establecido su estado a listo el primero, el sistema sigue esperando a que el segundo cambie su estado. Una vez el segundo pulsa el botón de listo, el juego avanza a la escena correspondiente y comienza la partida.
Requisitos asociados	F-U-C35 F-S-R36
Estado	OK

TABLA 5.19.
CP-18

Identificador	CP-18
Descripción	Comprobación de los mensajes de error cuando un jugador se desconecta en medio de la partida.
Resultado esperado	Si uno de los dos jugadores abandona la partida, finaliza esta, mostrando al jugador un mensaje de error con el motivo de la finalización.
Resultado obtenido	Cuando el *host* abandona la sesión, el sistema finaliza la partida y el cliente recibe un mensaje de error con el motivo de la finalización prematura. También ocurre lo mismo cuando es el cliente el que abandona la sesión.
Requisitos asociados	F-S-C40 F-S-R41
Estado	OK

TABLA 5.20.
CP-19

Identificador	CP-19
Descripción	Funcionamiento en Windows x64
Resultado esperado	La aplicación funciona correctamente en Windows x64
Resultado obtenido	La aplicación funciona correctamente en Windows 10 de 64 bits.
Requisitos asociados	NF-S-C43
Estado	OK

TABLA 5.21.
CP-20

Identificador	CP-20
Descripción	El sistema está traducido completamente al inglés.
Resultado esperado	Todos los menús, textos y mensajes de la aplicación están escritos en el idioma inglés.
Resultado obtenido	Toda la aplicación está traducida al inglés.
Requisitos asociados	NF-S-C44
Estado	OK

TABLA 5.22.
CP-21

Identificador	CP-21
Descripción	Comprobación de la aparición del tutorial.
Resultado esperado	Al iniciar la partida, se muestra al jugador un tutorial con los controles e interacciones posibles.
Resultado obtenido	Una vez la escena de la partida cambia a la del nivel 1, se muestra un menú superpuesto al usuario, en el que se incluyen todos los controles y acciones que puede realizar su personaje.
Requisitos asociados	NF-S-C45
Estado	OK

TABLA 5.23.
CP-22

Identificador	CP-22
Descripción	Comprobación de la ofuscación de la IP pública de los jugadores.
Resultado esperado	El sistema no muestra ni requiere en ningún momento la IP pública del jugador para unirse a una partida, manteniendo su privacidad.
Resultado obtenido	La IP pública del jugador no es necesaria en ningún momento, puesto que para unirse a una sala los jugadores emplean códigos que identifican a las salas. Además, como el sistema utiliza el servicio Relay, quien hace uso de las direcciones IP de los clientes es el propio servicio. El código implementa el protocolo DTLS en la comunicación con Relay, no empleando ni recolectando en ningún momento la IP del usuario y cifrando el tráfico en la comunicación.
Requisitos asociados	NF-S-R46
Estado	OK

TABLA 5.24.
CP-23

Identificador	CP-23
Descripción	Comprobación de la aplicación de las reglas y convenciones de codificación y nombramiento de C#.
Resultado esperado	Todo el código escrito cumple las reglas y convenciones de codificación y nombramiento de C#.
Resultado obtenido	Parte del código cumple las convenciones de manera parcial, teniendo variables públicas, por ejemplo, que en algunos casos podrían ser privadas.
Requisitos asociados	NF-S-R47
Estado	OK PARCIAL Problema asociado: P-03.

Con los casos de prueba anteriores, se han validado todos los requisitos descritos en el apartado del análisis del proyecto.

5.2. Problemas encontrados en el plan de pruebas

En este apartado se estudian los problemas encontrados durante la ejecución del plan de pruebas. Adicionalmente se realiza un estudio de posibles soluciones.

5.2.1. P-01

Este problema, asociado al caso de prueba CP-03, se debe a que la capacidad del disparo del cliente se ve limitada por el funcionamiento del sistema y su implementación. La razón por la que, en los proyectiles de la aplicación del cliente, se puede observar un retardo en su avance, se debe a que todos estos proyectiles son propiedad del servidor, es decir, del anfitrión de la partida. Esto explica que, en la máquina del anfitrión, los proyectiles sigan su trayectoria sin ningún impedimento. La razón específica por la que se producen estos teletransportes de los proyectiles en la máquina del cliente se debe a la frecuencia de sincronización de Netcode for GameObjects. Como se explicaba anteriormente, el componente NetworkTransform, asociado a estos proyectiles, sincroniza la posición, rotación y escala de los objetos a una frecuencia de actualización fija. Por defecto, esta tasa de sincronización es de 40 veces por segundo. No obstante, para aumentar la fluidez y, como solución temporal, se ha aumentado esta tasa a 60 veces por segundo, correspondiéndose a la tasa de actualización fija de las físicas del proyecto, también llamada *fixedUpdate*. Este aumento viene asociado a un coste adicional de recursos que, al ser un juego sin demasiados objetos en la escena, no supone un impacto muy significativo. No obstante, esto no es la solución óptima a este problema, teniendo la necesidad de implementar predicción por parte de los clientes, como se detalla más adelante, en el capítulo de conclusiones y líneas de trabajo futuras.

Además, al seguir comprobando este funcionamiento, se ha detectado que la velocidad de los proyectiles del cliente es ligeramente inferior a la de los proyectiles del *host*.

Aunque se realizó un estudio del problema durante la implementación del prototipo y de las causas posibles, no se ha llegado a una conclusión exacta del motivo por el que ocurre este suceso. Sin embargo, se piensa que es algún problema implícito con el motor de físicas de Unity y su funcionamiento con Netcode for GameObjects.

5.2.2. P-02

El problema P-02, asociado al CP-05, consiste en que, en algunos casos, cuando un enemigo daña al personaje del cliente de la partida, no se le aplica una fuerza de reacción. Tras investigar este problema, se ha deducido que el motivo es el motor de físicas de Unity una vez más.

NGO utiliza un modelo de simulación de físicas que únicamente se ejecuta en el servidor. Además, es necesario incluir el componente NetworkRigidbody para poder utilizar estas físicas. No obstante, este componente no interactúa de la misma manera que el componente Rigidbody estándar, dado que todos los objetos cuya autoridad la tiene un cliente, pasan a cambiar su modo de simulación a *kinematic*. Por consecuencia, los eventos de colisión no se disparan, dado que esta propiedad es de los *rigidbodies* que tienen su modo de simulación establecido a *dynamic*. Esto no debería suponer ningún problema en esta implementación, ya que los eventos *OnTriggerEnter*, que son los que se utiliza para comprobar cualquier colisión entre jugador y enemigos o proyectiles, se hace mediante estos eventos, que sí que deberían funcionar como es debido [84]. No obstante, sigue existiendo algún problema con la simulación de físicas o el código del prototipo implementado, ya que, en ciertos momentos, no se aplica esta fuerza al jugador.

5.2.3. P-03

Este problema está asociado al caso de prueba con identificador CP-23. Principalmente, este problema surge del tiempo disponible, puesto que realizar una revisión de todo el código del prototipo no ha sido posible. Algunas acciones que habría que realizar en el código para que cumpliese totalmente las convenciones de codificación y reglas de nombramiento de C#, son:

- Cambiar el nivel de accesibilidad de algunas de las variables, puesto que algunas de las variables que no tienen ninguna restricción de acceso podrían emplear estas protecciones, haciendo que el código, en general, fuera más seguro. Además, habría que ajustar sus nombres de acuerdo con el nivel de acceso.
- Revisar todo el código, puesto que solo se ha podido revisar parte del mismo, debido a la cantidad de líneas de código y *scripts* que se han elaborado.

6. MARCO REGULADOR

En este capítulo se expone la información relevante al marco legislativo que aplica en este proyecto, además de los estándares técnicos que hay que tener en cuenta a la hora de crear un proyecto de esta índole. Asimismo, se detallan las licencias bajo las que están sujetos los programas y elementos utilizados en este trabajo.

6.1. Legislación aplicable

Principalmente, la legislación aplicable a un proyecto de este origen, un videojuego, son aquellas relacionadas con la propiedad intelectual, leyes aplicadas al software y leyes de protección de datos. En la Unión Europea, y más concretamente en España, estas leyes son las siguientes:

- Reglamento General de Protección de Datos (RGPD). Esta ley, ratificada el 27 de abril de 2016, establece las normas a seguir para proteger los datos personales de las personas físicas y poder garantizar la libre circulación del programa en el espacio de la Unión Europea. Esta ley abarca diversos aspectos, como el consentimiento, los derechos de los interesados, las obligaciones de los encargados del tratamiento de los datos y las sanciones que conlleva su incumplimiento. Se promueve, ante todo, la transparencia, seguridad y responsabilidad en el tratamiento de datos de las personas físicas [85].
- Ley Orgánica de Protección de Datos Personales y Garantía de los Derechos Digitales (LOPDPGDD). Esta ley, publicada en 2018, actúa como una extensión y adaptación al ámbito español del RGDP. Abarca todos los aspectos cubiertos por el RGDP, además de incluir el papel de la Agencia Española de Protección de Datos (AEPD) en la aplicación de estas leyes. Asimismo, en ella se introducen y regulan los derechos digitales de los ciudadanos españoles [86].

Esta aplicación cumple con las regulaciones establecidas por las leyes anteriores, puesto que en ningún momento el programa trata directamente con los datos personales de los usuarios. En la implementación del multijugador, se hace uso del protocolo DTLS (en español, Protocolo de Seguridad de la Capa de Transporte), para garantizar la seguridad e integridad de los datos transferidos a través del servicio Relay de Unity Gaming Services, siendo este servicio el que almacena, de manera temporal, la dirección IP e ID de autenticación de los usuarios [87].

El único parámetro de los jugadores que trata el programa es el nombre de usuario que pueden introducir al jugar una partida multijugador. No obstante, este nombre de usuario se almacena a nivel de registro en el ordenador del jugador, teniendo como única finalidad, en la actualidad, mostrar al jugador este nombre en la partida. Si el jugador no incluye un nombre, el sistema le asignará un nombre aleatorio. En ningún momento se emplea este nombre de usuario para otros fines.

6.2. Estándares técnicos

Los estándares técnicos que aplican a un videojuego son aquellos que se deben aplicar al software. Por lo general, son estándares que tienen en cuenta la evaluación y control

de calidad del software y aquellos que se aplican a las tecnologías de la información. También se debe tener en cuenta aquellos estándares y convenciones de codificación de los lenguajes de programación utilizados en el desarrollo del proyecto. Los estándares que se han identificado como relevantes a este trabajo, son los siguientes:

- UNE-ISO/IEC 90003. Este estándar, idéntico a la Norma Internacional ISO/IEC 90003:2004, proporciona unas directrices a seguir para la aplicación de la Norma ISO 9001:2000 en proyectos de desarrollo de software. Esta norma es independiente de la tecnología y procedimientos utilizados durante el desarrollo, por lo que pretende ofrecer una guía que se pueda a cualquier proyecto de desarrollo de software, sea cual sea la finalidad que tenga. En este estándar, se tratan temas como la gestión de la calidad, el desarrollo o el mantenimiento del software [88].
- *Common C# code conventions*. Estándares de codificación de C# especificados por Microsoft. Presentan un conjunto de normas para mejorar el mantenimiento del código, con el objetivo de aumentar la consistencia, legibilidad y capacidad de colaboración sobre este [89].

6.3. Licencias utilizadas

En este apartado se enumeran las licencias de uso de los programas, componentes y servicios utilizados durante la elaboración de este proyecto.

- Unity Engine. Unity dispone de varias licencias según los usuarios que vayan a usar la plataforma y la finalidad que tengan los productos elaborados en ella [90]:
    - Unity Personal. Es la licencia más básica que ofrece Unity, con el menor número de características. No obstante, es la versión gratuita de Unity y se permite su uso siempre y cuando los ingresos no superen los 100.000$ anuales.
    - Unity Student. Incluye la versión Unity Personal y algunas características y descuentos adicionales en la Unity Asset Store.
    - Unity Pro. Es la versión de pago de Unity, incluyendo soporte técnico oficial prioridad en el servicio de atención al cliente y la publicación de los videojuegos desarrollados en diferentes plataformas. Es un servicio de suscripción que tiene un precio de entrada de 185$ al mes por puesto de trabajo.
    - Unity Enterprise. Una licencia aún más especializada de Unity, en la que se ofrece la asistencia de ingenieros expertos, acceso al código fuente y soporte a largo plazo. No tiene un precio definido, teniendo que contactar directamente con el servicio de atención al cliente de Unity para negociar una oferta.
    - Unity Industry. Es una versión especializada para los desarrolladores de aplicaciones 3D de sectores industriales. Incluye todas las características de Unity Enterprise y otras herramientas especializadas. Su precio de entrada es de 450$ al menos por puesto de trabajo.

En este proyecto se ha utilizado el plan Unity Personal, dado que incluye los recursos necesarios para la elaboración de un prototipo como este. Adicionalmente, como no es un proyecto que se vaya a distribuir comercialmente, no hay ninguna probabilidad de que se vaya a superar el margen de beneficios anuales que limita el uso de esta licencia.

- Unity Gaming Services. Como se han utilizado los servicios de Unity Lobby y Unity Relay, se ha empleado los Unity Gaming Services. Estos servicios tienen un nivel gratuito mensual que, siempre y cuando no se superen unas cuotas de uso, no tienen ningún coste adicional. Una vez se supera este margen, Unity cobra en base a una cuota de uso. No obstante, debido a que la página que mostraba estos precios no se encuentra disponible en la actualidad, no se ha podido incluir información sobre estas cuotas de uso [91].
  Dado que el prototipo actual no se está comercializando, en ningún momento se superan los límites gratuitos mensuales establecidos, no llegándose siquiera al 1% de su uso, algo que se puede monitorizar desde el panel de control de Unity Cloud [92].
- A* Pathfinding Project. Se ha utilizado la versión gratuita de este proyecto, ya que no ha existido la necesidad de utilizar las características adicionales que incluye la versión Pro, o se han desarrollado alternativas propias. Esta versión tiene un coste de 139,72€ en la Unity Asset Store actualmente [93].
- Aseprite. Esta aplicación de diseño gráfico tiene un coste de 20$. No obstante, dado que este proyecto empezó como un proyecto de código abierto en 2001 y su código está publicado en GitHub, los desarrolladores permiten, según su EULA y el FAQ detallado en su página web, el uso personal y gratuito del programa, siempre y cuando los usuarios compilen el programa ellos mismos y no lo distribuyan comercialmente. Además, permiten el uso personal y comercial de las *assets* desarrolladas con este programa, incluyendo las desarrolladas con una versión compilada por el usuario [94], [95].
- Freesound. Este repositorio, del que se han obtenido todos los sonidos empleados en este prototipo, incluye una colección de sonidos con diferentes licencias de código abierto. Todas pistas de audio utilizadas para los efectos de sonido están sujetas a la licencia Creative Commons 0 – Dominio Público, por lo que no es necesaria la acreditación de estos trabajos. No obstante, la pista utilizada para el ruido de fondo de la escena, tienen derechos de atribución, por lo que se incluye un enlace a este sonido en la bibliografía de este proyecto [73], [96], [97].
- Microsoft Office. Durante el desarrollo de esta memoria, se ha utilizado la versión educativa del paquete de Microsoft Office, por lo que no se ha presentado ningún coste adicional.
- Zotero. Este programa está sujeto a la licencia AGPL, por lo que su uso es completamente gratuito [98].

7. PLANIFICACIÓN

En este capítulo se incluye la planificación resultante de la realización del proyecto. Para exponer esta planificación se hace uso de la tabla 7.1, que detalla todas las tareas en las que se ha dividido el proyecto. Asimismo, en la figura 7.1 se muestra un diagrama de Gantt, que representa gráficamente el desarrollo de estas tareas a lo largo del periodo de tiempo en el que se ha realizado este proyecto.

Para facilitar luego la creación y visualización del diagrama de Gantt, se ha creado la tabla 7.1. Esta tabla contiene los siguientes campos:

- Identificador: pretende servir como identificador de la tarea. Cada una de ellas sigue el siguiente esquema de nombramiento:

  T-CAT-XX

  Donde:
  - T indica que se refiere a una tarea.
  - CAT indica la categoría a la que corresponde la tarea. Se han definido las siguientes categorías:
    - DIS: categoría en la que se engloban todas las tareas relacionadas con el diseño de la aplicación creada.
    - DEV: esta categoría contiene aquellas tareas que se refieren al desarrollo del prototipo elaborado.
    - MEM: en ella se engloban todas las tareas relacionadas con la creación de esta memoria.
  - XX: indica el índice de la tarea dentro de la categoría correspondiente.
- Descripción: explica brevemente en lo que consiste la tarea.
- Tiempo: estimación, en horas, del tiempo a la ejecución de la tarea.

TABLA 7.1.
TAREAS DEL PROYECTO

Identificador	**Descripción**	**Tiempo (horas)**
T-DIS-01	Elección y formalización del tema del trabajo.	2
T-DIS-02	Elaboración de ideas y diseño preliminar.	4
T-DIS-03	Creación de un borrador con unos requisitos base.	6
T-DEV-01	Aprendizaje y refuerzo de nuevos conceptos y elementos.	20
T-DEV-02	Creación de los gráficos del juego.	40
T-DEV-03	Implementación de un modo un jugador base.	90
T-DEV-04	Diseño de las interfaces de usuario y flujo de escenas.	16
T-DEV-05	Implementación del multijugador.	120
T-DEV-06	Pruebas y corrección de errores.	30
T-MEM-01	Desarrollo del capítulo 1 - Introducción.	10

Identificador	Descripción	Tiempo (horas)
T-MEM-02	Desarrollo del capítulo 2 – Estado del arte.	36
T-MEM-03	Desarrollo del capítulo 3 – Análisis.	12
T-MEM-04	Desarrollo del capítulo 4 – Diseño.	12
T-MEM-05	Desarrollo del capítulo 5 – Evaluación.	6
T-MEM-06	Desarrollo del capítulo 6 – Marco legal.	6
T-MEM-07	Desarrollo del capítulo 7 – Planificación.	10
T-MEM-08	Desarrollo del capítulo 8 – Entorno socioeconómico.	8
T-MEM-09	Desarrollo del capítulo 9 – Conclusiones.	6
T-MEM-10	Desarrollo del resumen en inglés.	18
T-MEM-11	Revisión y correcciones.	2
T-MEM-12	Páginas previas y anexos.	2
TOTAL	Suma de las horas dedicadas a las tareas.	**456**

A continuación, se muestra la figura 7.1, que contiene el diagrama de Gantt elaborado a partir de la información de la tabla 7.1 y las fechas estimadas en las que se ejecutó cada tarea. Como se puede observar, en este diagrama se divide el tiempo en semanas, donde a cada semana se le asigna el número de la semana correspondiente dentro su año.

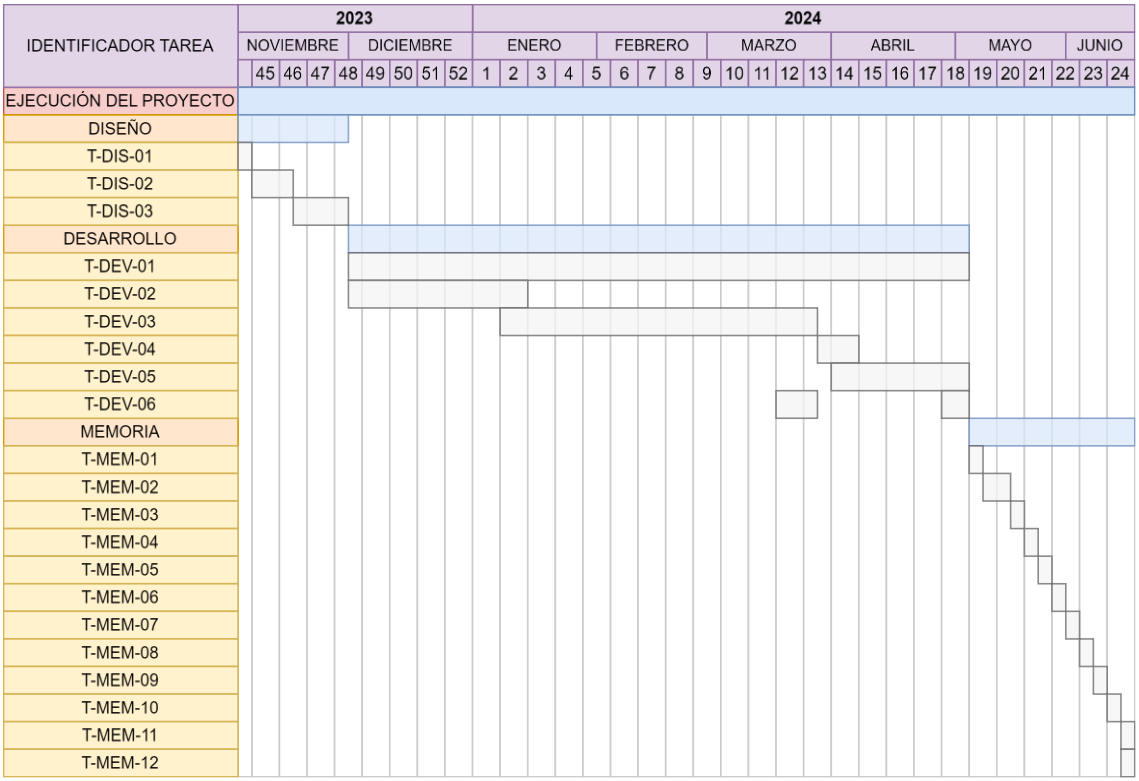

Fig. 7.1. Diagrama de Gantt del proyecto

Como se puede observar en la figura 7.1, la ejecución del proyecto empezó a mediados de noviembre de 2023, finalizando con la entrega del proyecto el 13 de junio de 2024. Además, se puede apreciar el solapamiento de algunas tareas. Esto indica que las tareas se han ejecutado de forma paralela, como es con el caso de las tareas T-DEV-02 y T-DEV-03. Este solapamiento se debe a que, antes de haber terminado con la creación de las imágenes, se empezaron a implementar ciertas mecánicas en el prototipo, como el movimiento del jugador.

Otro solapamiento ocurre entre las tareas T-DEV-03 y T-DEV-06 o entre T-DEV-05 y T-DEV-06. Esto se debe a que esta última, que se corresponde con las pruebas y correcciones del prototipo y que se ejecutaron a la vez que se estaba finalizando esas partes. El solapamiento entre T-DEV-04 (flujo de escenas y diseño de la interfaz de usuario) y T-DEV-05 (implementación del multijugador) es debido a la relación entre ambas partes, ya que, para el modo multijugador, también hubo que tener en cuenta un flujo de escenas y hubo que desarrollar los menús correspondientes para la conexión de los jugadores a una sesión.

El último solapamiento existente ocurre entre T-MEM-11 y T-MEM-12, debido a que ambas partes, que se corresponden a las revisiones y páginas previas, se realizaron en los últimos dos días antes de la entrega.

Por último, hay que mencionar la tarea T-DEV-01, correspondiente con el aprendiza y refinamiento de conceptos relevantes al desarrollo del prototipo, algo que ha estado en constante evolución desde el inicio de la fase de desarrollo.

8. ENTORNO SOCIOECONÓMICO

En este capítulo se realiza una presupuestación del proyecto, además de estudiar el impacto que tiene un proyecto de esta índole en la sociedad y economía.

8.1. Presupuesto

Para la elaboración del presupuesto, se ha tenido en cuenta la duración total observada en el capítulo anterior. El cálculo de los salarios se ha realizado en base a las horas dedicadas a cada una de las tareas, asignando a cada puesto de trabajo las tareas que les corresponden, dependiendo de las funcionalidades que desempeñan dentro de una empresa. En el presupuesto, se han tenido en cuenta los costes del personal, hardware, software y gastos indirectos.

Como este presupuesto pretende simular la venta de un proyecto similar al desarrollado, se incluyen adicionalmente los impuestos relevantes al territorio de España, además de un margen de beneficio del 10% y un riesgo del 15%.

A continuación, se realiza un estudio de los salarios y posiciones de trabajo que serían necesarias para desarrollar este proyecto:

- Jefe de proyecto/Ingeniero de software: se encarga de la coordinación del resto de integrantes, la elaboración de la documentación necesaria y el diseño general del proyecto.
- Diseñador gráfico: su labor es crear los recursos visuales que se emplearán en la aplicación, además del diseño de las interfaces de usuario.
- Desarrollador/Programador: cumple las tareas de implementación del programa, además de la ejecución de las pruebas necesarias para la verificación de su funcionamiento.

Los salarios (brutos) medios de estas posiciones en España y en el año 2024, se pueden encontrar en la tabla 8.1. Todos estos datos se han obtenido de la fuente Talent.com [99].

TABLA 8.1.
SALARIO MEDIO DE LAS POSICIONES DE TRABAJO

Posición de trabajo	**Tasa anual**	**Tasa mensual**	**Tasa horaria**
Ingeniero de software	35.000€	2.917€	17,95€
Diseñador gráfico	20.500€	1.708€	10,51€
Programador	28.500€	2.375€	14,62€

En la tabla 8.2 se detallan los costes asociados a los salarios del personal. Es necesario señalar que el cálculo de estos importes se ha realizado en base a las horas invertidas en el proyecto, que se pueden encontrar en la tabla 7.1. Además, se considera que cada una de estas tareas puede ser ejecutada por una única persona. Asimismo, se detallan a continuación las tareas que ejecuta cada cargo:

- Ingeniero de software: todas las tareas asociadas al diseño (T-DIS-XX) y creación de la memoria (T-MEM-XX).
- Diseñador gráfico: la tarea T-DEV-02, asociada a la creación de los gráficos del videojuego. Además, se incluye la mitad del tiempo de la tarea T-DEV-04, asociada a la interfaz de usuario.
- Programador: todas las tareas relevantes al desarrollo (T-DEV-XX), a excepción de las ejecutadas por el diseñador gráfico.

TABLA 8.2.
COSTES SALARIALES DEL PROYECTO

Puesto de trabajo	**Horas dedicadas**	**Tasa horaria**	**Coste total**
Ingeniero de software	140 horas	17,95€/hora	1.113€
Diseñador gráfico	48 horas	10,51€/hora	504,48€
Programador	268 horas	14,62€/hora	3.918,16€
TOTAL	**456 horas**	**-**	**5.535,64€**

Una vez hallados los costes pertinentes al personal, se calculan los costes materiales del proyecto. Dado que el proyecto se puede desarrollar con la licencia Unity Personal, no se incluyen los gastos asociados a la licencia de Unity Pro. No obstante, si el proyecto se comercializase y superase los 100.000€ en ingresos anuales, sería necesario realizar un nuevo presupuesto incluyendo los costes asociados a esta licencia y los servicios asociados a Unity Gaming Services. En la tabla 8.3, se muestra el desglose en costes materiales.

TABLA 8.3.
DESGLOSE DE LOS COSTES MATERIALES

Material	**Precio**	**Unidades**	**Coste total**
PC [100]	689€/unidad	3	2.067€
Microsoft 365 Empresa Estándar [101]	14,04€/usuario/mes	24	336,96€
TOTAL	**-**	**-**	**2.403,96€**

Nótese que las 24 unidades correspondientes a las licencias de Microsoft 365 provienen del cálculo de los 3 trabajadores por los 8 meses de elaboración del proyecto.

En la tabla 8.4 se muestra el presupuesto final calculado a partir de estos costes. Además, se incluyen los costes indirectos, valorados en un 5% de la suma de los costes del personal y costes materiales, que incluyen gastos como la luz o el mantenimiento de los equipos utilizados. También se añade el margen de beneficios del 10%, un margen de riesgo financiero del 15% y el cálculo del IVA.

TABLA 8.4.
PRESUPUESTO DEL PROYECTO

Concepto	**Valor asociado**
Salarios	5.535,64€
Costes materiales	2.403,96€
Costes indirectos (5%)	396,98€
Beneficios (10%)	833,66€
Riesgo (15%)	1.250,49€
IVA (21%)	2.188,35€
TOTAL	**12.609,08€**

Como se puede apreciar en la tabla 8.4, la valoración realizada del proyecto tiene el coste total de doce mil seiscientos nueve euros con ocho céntimos.

8.2. Impacto socioeconómico

En este apartado se estudian los impactos que puede tener un proyecto de este origen en la economía, sociedad, sanidad y en el medioambiente.

8.2.1. Impacto social

Las contribuciones que puede hacer a cultura y sociedad un proyecto de este carácter o, en líneas generales, un videojuego multijugador, son las siguientes:

- Los videojuegos pueden contribuir en la **creación de nuevas comunidades**, ya sea a través de Internet o en persona. Esto puede ayudar a las personas a relacionarse, conociendo gente que tiene gustos parecidos a los suyos y ayudando a formar nuevas amistades. No obstante, también pueden surgir de estas comunidades comportamientos tóxicos o inadecuados.
- Ayudan a **promover el patrimonio cultural**, creando interés por temas relacionados con el videojuego, debido a que muchos de ellos basan su temática o historia, o se inspiran, en sucesos mitológicos o históricos [102].

8.2.2. Impacto en la educación y sanidad

Las contribuciones que pueden tener estos proyectos en este campo son:

- Pueden ayudar a la **reducción de estrés, mejora de la coordinación y función cerebral** de los usuarios, teniendo un impacto positivo en la salud de las personas. No obstante, también pueden tener un impacto negativo, como el **aumento del sedentarismo** o pueden crear **adicción por los videojuegos** [102].
- Pueden ayudar a **aumentar el nivel de educación** de las personas, debido a su uso para aprender nuevos idiomas o sobre temas relacionados con el videojuego que crean interés en los usuarios.

8.2.3. Impacto económico

En cuanto al impacto económico que tienen los videojuegos en un país, se identifican los siguientes:

- Ayudan a **crear nuevos puestos de empleo**, beneficiando directamente a la economía de un país, al crear nuevas oportunidades de trabajo.
- Hoy en día, la industria de los videojuegos es una de las **más productivas y rentables a nivel mundial**, ayudando en el desarrollo de la economía de los países [102].

8.2.4. Impacto medioambiental

Por último, en cuanto al impacto medioambiental que tienen los videojuegos se pueden identificar dos casos:

- Como punto negativo, **aumentan el consumo eléctrico**, algo que, por lo general, aumenta las emisiones de dióxido de carbono a la atmósfera, dado que la mayoría de los países del mundo siguen dependiendo altamente de combustibles fósiles para la generación de energía eléctrica.
- Los videojuegos se pueden utilizar para **concienciar sobre el cambio climático** y el medioambiente, ayudando a que cada vez más gente tome cartas en el asunto.

9. CONCLUSIONES

En este último apartado se ejecuta una recapitulación de los objetivos que se ha conseguido cumplir con el desarrollo de este trabajo de fin de grado. Asimismo, se tratan los problemas principales encontrados durante la evolución del proyecto y las posibles líneas de trabajo que pueden seguirse para mejorar el prototipo desarrollado.

9.1. Objetivos cumplidos

Con este trabajo se ha logrado cumplir el objetivo principal propuesto: el desarrollo de un prototipo funcional de un videojuego multijugador del género *roguelike*. Este hito incluye el diseño inicial de un videojuego de estas características, hasta la implementación de una demo o prototipo que permita a dos personas jugar en una misma sesión a través de Internet. A este prototipo se puede acceder a través del enlace del repositorio del proyecto, ubicado en el capítulo 3 de este documento. Se puede comprobar su correcto funcionamiento desde cualquier ordenador con el sistema operativo Windows 10 o Windows 11 y una arquitectura de 64 bits. Adicionalmente, se realiza una retrospectiva de los objetivos secundarios, analizando su cumplimiento o incumplimiento.

- Se ha aprendido a utilizar Unity Engine, acumulando más de 250 horas de uso en el desarrollo del prototipo. Se ha llegado a trabajar con diversos subsistemas del motor, entre ellos:
    - Motor de físicas de Unity y Rigidbody.
    - Universal Render Pipeline para un mayor ajuste de los gráficos del juego y efectos de posprocesado.
    - SpriteRenderers y Animators para la utilización de gráficos 2D y animación de estos.
    - El nuevo Input System de Unity, para controlar las señales de entrada que relizan los jugadores.
    - UI Toolkit para la construcción de las interfaces de usuario.
    - Netcode for GameObjects, para la implementación del multijugador.
    - Se ha trabajado con los servicios de Unity Gaming Services, mediante el uso de Lobby y Relay para la implementación del multijugador en línea.
- Se ha aprendido a programar en C#, habiendo elaborado más de 30 *scripts* a lo largo de todo el proyecto, cada uno con sus correspondientes funcionalidades. Adicionalmente, se han seguido de manera parcial las convenciones de codificación de este lenguaje de programación. La razón por la que no se han aplicado en su totalidad ha sido, principalmente, por falta de tiempo y porque su estudio se realizó tras haber dedicado tiempo previamente a algunos componentes del sistema.
- Se ha repasado y profundizado en varios conocimientos adquiridos durante la carrera universitaria, como la computación distribuida, las redes de ordenadores, interfaces de usuario, y varias más, aplicando dichos conocimientos en la elaboración de este prototipo.

- Se ha elaborado esta memoria, que pretende demostrar y documentar todo el proceso de diseño y desarrollo de este proyecto.

9.2. Problemas encontrados

Durante la elaboración del proyecto se han encontrado diversos problemas que han entorpecido el avance del proyecto.

En primer lugar, se tiene que hablar del desconocimiento de las tecnologías utilizadas, ya que, a lo largo de la carrera, nunca se ha utilizado un motor de desarrollo de videojuegos, por lo que en este proyecto se ha tenido que aprender a utilizar desde cero Unity. Este proceso de aprendizaje ha aumentado considerablemente la carga de trabajo, al tener que buscar y estudiar documentación relevante. Además, el haber decidido utilizar tecnologías de Unity más recientes, como el nuevo Input System o el último sistema de desarrollo de interfaces de usuario creado por Unity, UI Toolkit, ha aumentado el tiempo de investigación que se ha tenido que realizar al respecto, puesto que hay menos recursos disponibles de estas tecnologías.

En segundo lugar, se comenta la planificación y organización de este trabajo, dado que, durante la realización de este trabajo de fin de grado, se ha estado trabajando simultáneamente a tiempo parcial. Lo anterior, añadido a otros motivos personales, han obstaculizado el desarrollo de este proyecto y aumentado el tiempo que se ha tardado en implementar el trabajo de manera considerable. Se opina que, si se tiene una organización y planificación óptima, se puede realizar este proyecto en el transcurso de cuatro o cinco meses.

Por último, se nombran los problemas encontrados en la fase de implementación. Al implementar diversas mecánicas del juego se encontraron problemas que, en ocasiones, llevó días solucionar. La mayoría de estos problemas surgieron en la implementación del multijugador, donde se necesitó, en más de una ocasión, refactorizar parte del código, para que fuese posible la sincronización de los elementos de la escena entre el anfitrión y el cliente de la partida. En la actualidad, algunos de estos problemas no se han podido solucionar, debido a las limitaciones que presenta Netcode for GameObjects respecto al motor de físicas de Unity, o por la forma que tiene de trabajar con este motor el sistema.

9.3. Posibles mejoras

A continuación, se detallan algunas posibles adiciones y mejoras que se pueden realizar sobre el prototipo actual. Algunas de las mecánicas y características que se muestran, fueron descartadas durante la implementación del prototipo, debido a su complejidad o la cantidad de tiempo que se debería haber dedicado para desarrollarlas.

- *Client-side prediction*. De todas las mejoras posibles, esta es la que puede ofrecer una mejora inmediata en la jugabilidad para el cliente de la partida. Este mecanismo intenta solucionar los problemas que conlleva el uso de una arquitectura servidor-cliente, en la que el servidor tiene la propiedad y autoridad de todos objetos de la escena, incluyendo los personajes jugables de los clientes.

Estos problemas están asociados a la latencia existente entre las acciones que realiza el cliente y la respuesta del servidor, lo que empeora de manera considerable la experiencia de juego. En el caso del prototipo desarrollado, aunque el cliente tiene la autoridad de su personaje, por lo que su movimiento por la escena, cuando ejecuta una acción, es inmediato, no ocurre lo mismo cuando dispara. Como se explicó, en estos casos, el cliente realiza una petición al servidor o anfitrión, ya que es el único individuo que puede invocar nuevos objetos en la escena. Esto provoca que haya un ligero retardo entre la acción de disparar que realiza el jugador, y el disparo en sí. Este retardo es aún más notable si aumenta la latencia de juego. Por ejemplo, con 100 milisegundos de latencia, se puede apreciar fácilmente cómo los proyectiles del jugador aparecen en la posición en la que estaba el jugador hace unos instantes. La predicción por parte del cliente apunta a solucionar este tipo de problemas.

- Extrapolación. Esta técnica pretende solucionar los problemas que ocurren con la sincronización de la posición de objetos en movimiento en la parte del cliente. En el prototipo implementado, se puede observar fácilmente cómo se teletransportan pequeñas distancias los proyectiles de los jugadores y enemigos. Esto se debe a que, en el cliente, la posición de estos objetos se sincroniza a una velocidad de 60 veces por segundo (40 veces por segundo en caso de mantener la velocidad de actualización estándar ofrecida por el NetworkManager). Esto se traduce a que, en el caso de objetos de la escena que se mueven a una velocidad muy alta, como es el caso de los proyectiles, se aprecie a simple vista los saltos o "teletransportes" que realizan estos objetos cada vez que se sincroniza su posición. Con la implementación de la extrapolación, que pretende predecir la posición en la que estarán esos objetos en todo momento, quedaría solucionado este problema.
- Generación aleatoria de la mazmorra. La implementación de un algoritmo de generación pseudoaleatoria para los niveles permitiría que el juego fuese en cada partida completamente distinto. Esto ofrecería mayor variedad al videojuego y aumentaría considerablemente las horas de juego que podría ofrecer el prototipo. No obstante, la creación e implementación de este algoritmo sería generalmente compleja.
- Diseño de niveles consecutivos. En el prototipo original se pensó en crear dos niveles, pero debido al tiempo disponible se descartó esta idea. No obstante, la implementación de nuevos niveles debería ser relativamente sencilla, dado que se podría reutilizar el mismo *tileset* y solo habría que estudiar cómo realizar el cambio de una escena a otra, conservando la vida y estadísticas de los jugadores.
- Diseño de nuevos enemigos y jefes. Se podrían implementar nuevos enemigos, con mecánicas más avanzadas, que apareciesen en los siguientes niveles de la mazmorra.
- Soporte para mandos. Esta mejora consiste en crear un esquema de controles para mandos, lo que abriría las puertas al soporte para consolas. Sería necesario

realizar una revisión del modo de apuntado del arma. Sin embargo, es una de las mejoras propuestas más sencillas de implementar.

- Soporte para dispositivos móviles. En el caso de diseñar un esquema de controles en pantalla, se podría realizar una compilación para dispositivos móviles, dado que los requisitos para poder ejecutar el juego en una máquina son muy bajos, y Unity ya incluye soporte para iOS y Android, siempre y cuando se instalen los paquetes de desarrollo correspondientes en el ordenador en el que se esté ejecutando Unity.
- Nuevas armas y *power-ups*. Se podrían diseñar nuevas armas para el jugador, algo que se acabó desestimando del prototipo inicial, al no ser esencial en su desarrollo, debido al tiempo que se disponía. También se podrían programar nuevas mejoras para los jugadores durante la partida, como aumentos del daño que realizan, mayor cadencia de disparo, etcétera. La implementación de estas mejoras sería bastante sencilla de realizar.

BIBLIOGRAFÍA

[1] P. Rao, «50 Years of Video Game Industry Revenues, by Platform», Visual Capitalist. Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://www.visualcapitalist.com/video-game-industry-revenues-by-platform/

[2] «LÍDERES DEL ENTRETENIMIENTO: "¿QUIÉN MERECE SER REY?" | LinkedIn». Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://www.linkedin.com/pulse/l%C3%ADderes-del-entretenimiento-qui%C3%A9n-merece-ser-rey-emilio-hurtado-ruiz/

[3] «Estudio mercado videojuegos: ¿Cuánto dinero mueve? (infografía)», Bankinter. Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://www.bankinter.com/blog/finanzas-personales/mercado-videojuegos-dinero-estudio-espana-mundo

[4] «Newzoo's games market revenue estimates and forecasts by region and segment for 2023», Newzoo. Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://newzoo.com/resources/blog/games-market-estimates-and-forecasts-2023

[5] «Steam annual game releases by developer type 2023», Statista. Accedido: 10 de junio de 2024. [En línea]. Disponible en: https://www.statista.com/statistics/1411839/number-games-released-steam-developer-type/

[6] S. Team, «Did you know that 60% of game developers use game engines?», SlashData. Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://www.slashdata.co/post/did-you-know-that-60-of-game-developers-use-game-engines

[7] V. Reports, «Roguelike Game Market Size to Grow USD 57,336 Million in 2030 by 2030 at a CAGR of 12.3% | Valuates Reports». Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://www.prnewswire.com/news-releases/roguelike-game-market-size-to-grow-usd-57-336-million-in-2030-by-2030-at-a-cagr-of-12-3--valuates-reports-302079781.html

[8] BillWagner, «C# identifier names - rules and conventions - C#». Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/identifier-names

[9] «BNL | History:The First Video Game?» Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://www.bnl.gov/about/history/firstvideo.php

[10] H. Labs, *Tennis For Two*. 2011. Accedido: 16 de mayo de 2024. [Photo]. Disponible en: https://www.flickr.com/photos/hslphotosync/5941685811/

[11] Jamal_Aladdin, «The Evolution of Multiplayer Gaming: A Journey Through Time», Medium. Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://medium.com/@Jamal_Aladdin/the-evolution-of-multiplayer-gaming-a-journey-through-time-e34ef59294c2

[12] «Pong - Videogame by Atari», Museum of the Game. Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://www.arcade-museum.com/Videogame/pong

[13] «Astro Race - Videogame by Taito», Museum of the Game. Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://www.arcade-museum.com/Videogame/astro-race

[14] «Introduction | PDP-1 Restoration Project | Computer History Museum». Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://www.computerhistory.org/pdp-1/introduction/

[15] «Nintendo Entertainment System», Nintendo of Europe AG. Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://www.nintendo.com/es-es/Hardware/La-historia-de-Nintendo/Nintendo-Entertainment-System/Nintendo-Entertainment-System-627024.html

[16] «Dictionary.com | Meanings & Definitions of English Words», Dictionary.com. Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://www.dictionary.com/browse/mmo

[17] «Counter-Strike 2 Steam Charts», SteamDB. Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://steamdb.info/app/730/charts/

[18] «Most played Multiplayer Games Steam Charts», SteamDB. Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://steamdb.info/charts/?tagid=3859

[19] «Overcooked! 2 en Steam». Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://store.steampowered.com/app/728880/Overcooked_2/

[20] «Client-server and peer-to-peer networks - Computer networks and topologies - OCR - GCSE Computer Science Revision - OCR», BBC Bitesize. Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://www.bbc.co.uk/bitesize/guides/zvspfcw/revision/4

[21] «¿Cuáles son las ventajas y desventajas de las arquitecturas peer-to-peer y cliente-servidor?» Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://www.linkedin.com/advice/0/what-advantages-disadvantages-peer-to-peer

[22] «Network topologies | Unity Multiplayer Networking». Accedido: 16 de mayo de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/current/terms-concepts/network-topologies/

[23] «Dictionary.com | Meanings & Definitions of English Words», Dictionary.com. Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://www.dictionary.com/browse/eSports

[24] «Create a game with a listen server and host architecture | Unity Multiplayer Networking». Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/current/learn/listen-server-host-architecture/

[25] «Distributed authority topologies | Unity Multiplayer Networking». Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/current/terms-concepts/distributed-authority/

[26] «¿Qué es la traducción de direcciones de red (NAT)? - Software Check Point», Check Point Software. Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://www.checkpoint.com/cyber-hub/network-security/what-is-network-address-translation-nat/

[27] «On the Historical Origin of the “Roguelike” Term», Slashie’s Journal. Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://blog.slashie.net/on-the-historical-origin-of-the-roguelike-term/

[28] «RFD: rec.games.dungeon.* hierarchy». Accedido: 21 de mayo de 2024. [En línea]. Disponible en: https://groups.google.com/g/news.groups/c/CdWOd-M6g-w/m/cgNn2b9uU2sJ

[29] «Roguelike Games Info and FTP Sites (FAQ)». Accedido: 21 de mayo de 2024. [En línea]. Disponible en: https://groups.google.com/g/rec.games.roguelike.misc/c/o66P9tiyxwI/m/EBIpBDSCzOsJ

[30] Thedarkb, *English: A screenshot of the BSD version of Rogue.* 2021. Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://commons.wikimedia.org/wiki/File:Rogue_Screenshot.png

[31] Z. Rogue, «What "Roguelike" Meant», Medium. Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://zenorogue.medium.com/what-roguelike-meant-fb8b0e1601a

[32] «What a roguelike is - RogueBasin». Accedido: 21 de mayo de 2024. [En línea]. Disponible en: https://roguebasin.com/index.php/What_a_roguelike_is

[33] «Roguelike - Fanlore». Accedido: 21 de mayo de 2024. [En línea]. Disponible en: https://fanlore.org/wiki/Roguelike

[34] «Enter the Gungeon», Enter the Gungeon. Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://enterthegungeon.com/

[35] «Ahorra un 70% en Enter the Gungeon en Steam». Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://store.steampowered.com/app/311690/Enter_the_Gungeon/

[36] «Enter the Gungeon». Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://www.metacritic.com/game/enter-the-gungeon/

[37] R. Cowley y Editor, «Enter the Gungeon shoots past three million units sold», pocketgamer.biz. Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://www.pocketgamer.biz/news/72336/enter-the-gungeon-3-million-units/

[38] «The Binding of Isaac en Steam». Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://store.steampowered.com/app/113200/The_Binding_of_Isaac/

[39] «The Binding of Isaac: Rebirth Price history», SteamDB. Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://steamdb.info/app/250900/charts/

[40] M. Media, «About The Creator», Maestro Media. Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://maestromedia.com/pages/about-the-creator

[41] «The Binding of Isaac: Afterbirth+ en Steam». Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://store.steampowered.com/app/570660/The_Binding_of_Isaac_Afterbirth/

[42] «Nuclear Throne Steam stats | Gamalytic». Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://gamalytic.com/game/242680

[43] «Nuclear Throne en Steam». Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://store.steampowered.com/app/242680/Nuclear_Throne/

[44] «Nuclear Throne». Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://www.metacritic.com/game/nuclear-throne/

[45] «Vlambeer». Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://www.vlambeer.com/

[46] «What platforms are supported by Unity?», Unity. Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity

[47] okky, «History of Unity Game Engine», Agate. Accedido: 23 de mayo de 2024. [En línea]. Disponible en: https://agate.id/history-of-unity-game-engine/

[48] J. Drake, «24 Great Games That Use The Unity Game Engine», TheGamer. Accedido: 23 de mayo de 2024. [En línea]. Disponible en: https://www.thegamer.com/unity-game-engine-great-games/

[49] «Beat Saber en Steam». Accedido: 23 de mayo de 2024. [En línea]. Disponible en: https://store.steampowered.com/app/620980/Beat_Saber/

[50] G. M. Y. D. with a keen interest in G. Development y V. Reality!, «Unreal Engine and its Evolution | Extern Labs Inc». Accedido: 23 de mayo de 2024. [En línea]. Disponible en: https://externlabs.com/blogs/unreal-engine-and-its-evolution/

[51] «Unreal Engine potencia la producción de cine y televisión», Unreal Engine. Accedido: 23 de mayo de 2024. [En línea]. Disponible en: https://www.unrealengine.com/es-ES/uses/film-television

[52] G. Engine, «A decade in retrospective and future», Godot Engine. Accedido: 24 de mayo de 2024. [En línea]. Disponible en: https://godotengine.org/article/retrospective-and-future/

[53] «Google Trends», Google Trends. Accedido: 24 de mayo de 2024. [En línea]. Disponible en: https://trends.google.es/trends/explore?date=2014-01-01%202024-05-24&q=%2Fm%2F0_x6l2s&hl=es

[54] «godotengine/godot: Godot Engine – Multi-platform 2D and 3D game engine». Accedido: 24 de mayo de 2024. [En línea]. Disponible en: https://github.com/godotengine/godot

[55] «Unreal Engine 5», Unreal Engine. Accedido: 24 de mayo de 2024. [En línea]. Disponible en: https://www.unrealengine.com/es-ES/unreal-engine-5

[56] «Preguntas Frecuentes», Godot Engine documentation. Accedido: 24 de mayo de 2024. [En línea]. Disponible en: https://docs.godotengine.org/es/4.x/about/about/faq.html

[57] «Introduction to Blueprints». Accedido: 24 de mayo de 2024. [En línea]. Disponible en: https://dev.epicgames.com/documentation/en-us/unreal-engine/introduction-to-blueprints-visual-scripting-in-unreal-engine

[58] «GDScript reference», Godot Engine documentation. Accedido: 24 de mayo de 2024. [En línea]. Disponible en: https://docs.godotengine.org/es/4.x/tutorials/scripting/gdscript/tutorials/scripting/gdscript/gdscript_basics.html

[59] «Relay Solución gratuita de conexión y redes P2P | Unidad», Unity. Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://unity.com/products/relay

[60] U. Technologies, «Unity - Manual: Unity User Manual 2022.3 (LTS)». Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://docs.unity3d.com/Manual/index.html

[61] BillWagner, «Un paseo por C#: información general - C#». Accedido: 15 de mayo de 2024. [En línea]. Disponible en: https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/overview

[62] «A* Pathfinding Project». Accedido: 3 de enero de 2024. [En línea]. Disponible en: https://arongranberg.com/astar/

[63] «Input System | Input System | 1.6.3». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.6/manual/index.html

[64] «Desarrollar interfaces gráficas de usuario | Kit de herramientas de interfaz de usuario de Unity», Unity. Accedido: 9 de junio de 2024. [En línea]. Disponible en: https://unity.com/features/ui-toolkit

[65] «Universal Render Pipeline overview | Universal RP | 14.0.11». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@14.0/manual/index.html

[66] «About Multiplayer Tools | Multiplayer Tools | 1.1.0». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://docs.unity3d.com/Packages/com.unity.multiplayer.tools@1.1/manual/index.html

[67] «Unity Lobby». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://docs.unity.com/ugs/manual/lobby/manual/unity-lobby-service

[68] «Unity Relay». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://docs.unity.com/ugs/manual/relay/manual/introduction

[69] D. Capello, «Aseprite». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://www.aseprite.org/

[70] «draw.io». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://app.diagrams.net/

[71] «GitHub». Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://github.com/

[72] «GitHub Desktop», GitHub Desktop. Accedido: 1 de junio de 2024. [En línea]. Disponible en: https://desktop.github.com/

[73] «Freesound». Accedido: 2 de junio de 2024. [En línea]. Disponible en: https://freesound.org/

[74] «Understanding ownership and authority | Unity Multiplayer Networking». Accedido: 19 de mayo de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/current/basics/ownership/

[75] «Tricks and patterns to deal with latency | Unity Multiplayer Networking». Accedido: 19 de mayo de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/current/learn/dealing-with-latency/

[76] «About Netcode for GameObjects | Unity Multiplayer Networking». Accedido: 19 de mayo de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/1.5.2/about/

[77] «Entities overview | Entities | 1.2.1». Accedido: 19 de mayo de 2024. [En línea]. Disponible en: https://docs.unity3d.com/Packages/com.unity.entities@1.2/manual/index.html

[78] «Entity concepts | Entities | 1.2.1». Accedido: 19 de mayo de 2024. [En línea]. Disponible en: https://docs.unity3d.com/Packages/com.unity.entities@1.2/manual/concepts-entities.html

[79] «A Brief History of Mirror | Mirror». Accedido: 19 de mayo de 2024. [En línea]. Disponible en: https://mirror-networking.gitbook.io/docs/trivia/a-history-of-mirror

[80] «First Gear Games, a game design company and education resource. Home of Fish-Networking.» Accedido: 19 de mayo de 2024. [En línea]. Disponible en: https://firstgeargames.com/

[81] «NetworkObject | Unity Multiplayer Networking». Accedido: 4 de junio de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/

[82] «Sending Events with RPCs | Unity Multiplayer Networking». Accedido: 4 de junio de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/1.5.2/advanced-topics/messaging-system/

[83] «NetworkVariable | Unity Multiplayer Networking». Accedido: 4 de junio de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/1.5.2/basics/networkvariable/

[84] «Physics | Unity Multiplayer Networking». Accedido: 6 de junio de 2024. [En línea]. Disponible en: https://docs-multiplayer.unity3d.com/netcode/1.5.2/advanced-topics/physics/

[85] «REGLAMENTO (UE) 2016/ 679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO - de 27 de abril de 2016 - relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/ 46/ CE (Reglamento general de protección de datos)».

[86] «BOE-A-2018-16673 Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.» Accedido: 6 de junio de 2024. [En línea]. Disponible en: https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673&p=20230509&tn=1

[87] «Privacy overview». Accedido: 6 de junio de 2024. [En línea]. Disponible en: https://docs.unity.com/ugs/en-us/manual/relay/manual/privacy-overview

[88] «AENORmas». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://plataforma.aenormas.aenor.com/pdf/UNE/N0034122

[89] BillWagner, .«NET Coding Conventions - C#». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions

[90] «Real-time tools for 3D, AR, and VR development | Products», Unity. Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://unity.com/products

[91] U. Technologies, «Unity Gaming Services Pricing - Start Building for Free | Unity». Accedido: 20 de mayo de 2024. [En línea]. Disponible en: https://unity.com/solutions/gaming-services/pricing

[92] «Unity Cloud». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://cloud.unity.com/home/login

[93] «A* Pathfinding Project Pro | Behavior AI | Unity Asset Store». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://assetstore.unity.com/packages/tools/behavior-ai/a-pathfinding-project-pro-87744?aid=1011l7JId&utm_campaign=unity_affiliate&utm_medium=affiliate&utm_source=partnerize-linkmaker

[94] D. Capello, «Aseprite». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.aseprite.org/

[95] «aseprite/EULA.txt at main · aseprite/aseprite». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://github.com/aseprite/aseprite/blob/main/EULA.txt

[96] «CC0 1.0 Deed | CC0 1.0 Universal | Creative Commons». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://creativecommons.org/publicdomain/zero/1.0/

[97] «Freesound - sci-fi - ambient 03.wav by AniCator». Accedido: 28 de marzo de 2024. [En línea]. Disponible en: https://freesound.org/people/AniCator/sounds/100482/?

[98] «GNU Affero General Public License - GNU Project - Free Software Foundation». Accedido: 7 de junio de 2024. [En línea]. Disponible en: https://www.gnu.org/licenses/agpl-3.0.html

[99] «Salario en España - Salario Medio», Talent.com. Accedido: 8 de junio de 2024. [En línea]. Disponible en: https://es.talent.com/salary

[100] «PcCom Lite Intel Core i5-11400F / 16GB / 1TB SSD / RTX 3050 Negro | PcComponentes.com». Accedido: 8 de junio de 2024. [En línea]. Disponible en: https://www.pccomponentes.com/pccom-lite-intel-core-i5-11400f-16gb-1tb-ssd-rtx-3050-negro

[101] «Comparar planes y precios de Microsoft 365 para empresas | Microsoft 365». Accedido: 8 de junio de 2024. [En línea]. Disponible en: https://www.microsoft.com/es-es/microsoft-365/business/compare-all-microsoft-365-business-products

[102] Directorate-General for Communications Networks, Content and Technology (European Commission), ECORYS, y KEA, *Understanding the value of a European video games society: final report*. Publications Office of the European Union, 2023. Accedido: 8 de junio de 2024. [En línea]. Disponible en: https://data.europa.eu/doi/10.2759/332575

DEVELOPMENT OF A ROGUELIKE MULTIPLAYER VIDEOGAME

Introduction

This document follows the design and development of a roguelike multiplayer video game. This first section seeks to explain the motivations for carrying out the final thesis on this specific topic. It also lists the objectives to be achieved by implementing this project.

Today, the video game industry has the lead in the entertainment sector, having around three billion players worldwide and generating revenues of more than $180 billion by 2022 [1], [2], [3]. Furthermore, according to Newzoo, this value is expected to exceed $205,4 billion in 2026 [4].

Not only has the revenue that the game industry has made augmented, but also the number of video games that get released per year. According to Statista, the number of video games published in 2018 in Steam, a platform for the digital distribution of video games, reached the value of 8.324 games. In comparison, in 2023 the number of new releases reached the value of 13.971, where most of those releases, around the 98% of them, were video games created by indie developers [5]. The rise in new video game releases is associated as well with the use of game engines. These developing environments have helped designers and programmers with less experience in the sector to develop new video games. In 2021, around 60% of video game developers used a video game engine to create their games [6].

As for the reasons why it has been decided to create a roguelike video game, they are the following:

- In the first place, personal interest has been a deciding factor. This video game genre has gained popularity in the past decade, thanks to more indie developers making games of the genre or its subgenre, roguelite video games. This increased popularity is visible in the income that the genre has generated in recent years. In 2023, this genre’s video games generated an income of $23.153 million, which is expected to increase to $57.336 million by 2023, as stated by Valuates Reports [7].
- Another reason is that the video games of this genre usually do not implement a multiplayer mode, focusing their gameplay on the single player experience. This offers the opportunity to incorporate a relatively new mechanic within the genre, as only a small percentage of these games let players play online with others.

Other reasons that had led to the development of this specific thesis is the possibility of deepening into the knowledge acquired during the bachelor's degree, as well as the opportunity of delving into the tools and implications that carry the development of multiplayer video games.

In the case of the objectives that this project seeks to accomplish, the following have been found:

- The main objective of the project is to follow the design and development of a multiplayer video game from the beginning. The game will include elements of the roguelike genre and will let at least two players play online in real time.
- As for secondary objectives, it is intended to learn to use a game engine, mainly Unity, and the programming language C#, from the point of view of a person with no previous experience. Other objectives are, as mentioned before, reinforcing the knowledge acquired during the computer science degree and adequately the creative and developmental process of this bachelor's thesis through the development of this dossier.

State of the art

Multiplayer video games

Multiplayer video games allow two or more players to interact with each other in real time, whether through the Internet or on the same machine. The first time a multiplayer video game made an appearance was in 1958, with the title Tennis for Two, developed by William Higinbotham. It used an oscilloscope and an analogue computer to simulate a tennis game, allowing two players to play together [9].

Multiplayer video games have kept evolving after several turning points, like the release of the Nintendo Entertainment System in the eighties, the appearance of MMO games in the nineties or the release of Counter Strike in 1999, a game that has more than a million active player as of today [11], [15], [17]. Currently, multiplayer video games accumulate 7,5 million concurrent players on average on Steam [18].

Network topologies in multiplayer game design

A network topology establishes how a computer network is structured. Depending on the area it covers, a network topology can be divided into two categories, which are explained below:

- Local area topologies. These architectures include those that occur within the same machine (couch multiplayer) and those that run in the same local area network (LAN). While the latter does not use an Internet connection, it is closely related to the client-host topology.
- Wide area topologies. These topologies encompass all of those that take place over wide area networks (WAN), such as the Internet. Such networks include peer to peer networks, where every user of the network has the same authority level and acts both as a client and server, as well as client-server and derived architectures. In these topologies, a single machine acts as a server, while every other user of the network acts as a client. In the case that the server also acts as a client, the topologies are called client-host networks. Finally, there are the networks of distributed authority, where the owner of the session can change without closing the connections. This is achieved by utilising a relay server to coordinate every member of the session.

The roguelike genre

The roguelike genre receives its name from Rogue, a game released in 1980. Its origins can be traced back to 1993 and USENET, where a discussion between users led to the creation a topology for video games that shared similar characteristics with Rogue [27], [28]. However, the genre has continued to evolve since the first definition given in that forum. Currently, there are many different definitions of the genre, but overall, all categorize a game as a roguelike video game if it fulfils the following characteristics [31], [32]:

- The elements of the scene and the environment are generated in a procedural manner.
- The player has an infinite amount of time to consider a strategy, as in a tabletop game.
- Any action of the player has permanent consequences, including death.
- The death of the player implies that a new game must be started, resulting in the loss of all accumulated progress.

Over time, a sub-genre of the former has emerged, the roguelite genre. This term is used today for video games whose bases and characteristics are those of roguelike games, but they incorporate critical changes to their definition, such as some form of progression [32].

Some of the most relevant roguelike video games that offer similar results to the solution proposed in this project are Enter the Gungeon, The Binding of Isaac and Nuclear Throne. However, all of these games are single player games and, aside from an expansion of The Binding of Isaac (Afterbirth+), they do not implement a multiplayer game mode [35], [38], [41], [43].

Game engines

Game development engines are specialised tools that assist game designers and programmers in the creation of video game applications, forming a development environment that facilitates and optimises the entire process of developing and maintaining video games. Video game engines also provide a graphics engine to render the programme's graphics, as well as one or more physics engines to emulate the laws of physics.

Table I below shows a comparison between Unity, Unreal and Godot, the three game engines studied in this project.

TABLE I
UNITY VS UNREAL VS GODOT

Characteristic	**Unity Engine**	**Unreal Engine**	**Godot Engine**
Target graphic style	3D and 2D	Only 3D	3D and 2D
Launch date	2005	1998	2014

Characteristic	Unity Engine	Unreal Engine	Godot Engine
Pricing	Free for personal use, as long as no more than $100.000 in revenue is generated. Paid plans, depending on the sector and size of the development teams.	Free for individual and small businesses with less than $1 million in annual revenue. Paid plans based on royalties or size of the development teams if more than 1$ million in revenue is generated.	Free and open source (MIT license).
Target platforms	More than 24 different platforms [46].	17 platforms [55].	6 platforms [56]
Programming languages	C# Unity Visual Scripting	C++ Visual Scripting with Blueprints [57]	GDScript [58] C# C++ Visual Scripting Community extensions: Rust, Nim, Python, JavaScript

State of the art conclusions

In this project, it has been decided to use a client-host network topology because of the following reasons:

- These topologies have a lower cost than those that use a dedicated server, even though they tend to be more expensive than P2P networks.
- They provide an optimal balance between network security and performance, with the inherent inconvenience of relying on a single individual's connection, the host.
- Their implementation is less complex than that of other architectures, as there is no need to manage a central service.

To solve connectivity issues between the users of the session, it has been decided to utilise a relay server, as stated in the design.

In order to implement the prototype, it has been determined that Unity Engine will be used, as it provides a free personal plan, a wide range of platforms for development, and an extensive documentation and community, thanks to its status as the most popular game engine among game developers [6], [60]. Furthermore, Unity and C# have never been used before in the bachelor's degree, so their use presents a new learning opportunity and a challenge.

Analysis

This section specifies the use cases and requirements that have been employed to create the design of the project.

Use cases

To create the use cases of this project, three different scenarios have been defined:

- Scenario 1 - Gameplay flow. It manifests all the actions players can execute during a game. Figure 3.1 represents its use case diagram.
- Scenario 2 – Single player connection flow. It represents the interaction between the player and the system, when the player attempts to initiate a single-player game. Figure 3.2 contains the use case diagram of this scenario.
- Scenario 3 – Multiplayer connection flow. It follows the actions taken by the players to establish a connection and commence a game in the multiplayer mode. Its corresponding use case diagram is represented in figure 3.3.

In conclusion, a total of 20 use cases have been specified. These use cases are presented in tables 3.1, 3.2 and 3.3, with each table corresponding to its respective scenario.

Software requirements specification

A total of 47 requirements have been specified for this project, of which 42 are functional and the remainder non-functional. These requirements can be found in tables 3.26 to 3.72.

To conclude the analysis, table 3.73 presents the traceability matrix, which demonstrates that all the use cases are satisfied by at least one requirement.

Design

In the design phase, a series of decisions have been made, based on the conclusions reached in the analysis phase. These decisions are split into two sections: artistic decisions and technical decisions.

Artistic decisions

The plot of the game has been determined to take place on an abandoned space station. The player, an astronaut, must explore the station to locate a treasure that was lost a long time ago. On its journey, the player will encounter adversaries in the form of robotic entities who will attempt to end its life.

In reference to the visuals, it has been determined that a two-dimensional style will be employed, as it is generally more straightforward to create images than 3D models. The 2D style employed is pixel art, a technique whereby images, or sprites, are created in a pixel-by-pixel basis. This allows the artist to create images with a low level of detail, such as the graphics used in retro games, or images with a high level of detail, such as photorealistic images. This decision has been made due to the low level of artistic experience of the student, enabling the developer to create all the visual assets of the game without the use of external resources.

Regarding the sound landscape, the lack of resources and knowledge in the field of sound effects has led to the decision to utilise free and open-source resources. All the resources have been sourced from the Freesound repository.

Technical decisions

This section provides a comprehensive analysis of the design decisions that were made regarding the logic of the developed video game.

- Player interactions with the system. These encompass all the ways in which the player interacts with the system and the mechanics associated with these interactions.
    - Player movement. As discussed in the artistic decisions, the game has been developed on a two-dimensional plane, so the player can only move in the XY plane. It is also important to note that it has been decided that one unit in Unity is equivalent to 16x16 pixels, as this is the size that has been kept in mind when designing the sprites.

      The player can move their character by holding down the following keys:
        - "W" to move the character in the positive direction of the Y axis (up).
        - "S" to move the character in the negative direction of the Y axis (down).
        - "A" to move the character along the negative direction of the X axis (left).
        - "D" to move the character along the positive direction of the X axis (right).

      Upon the player's pressing of one of these keys, the system will apply a force to the character in the desired direction, thereby causing the character to move in that direction.
    - Aiming and shooting the weapon. The player can direct the weapon of their character by moving the cursor on the screen. The direction of the weapon is determined by the position of the cursor.

      Upon pressing the left mouse button, which represents the "attack" action in the control scheme, the system generates a projectile at the tip of the player's weapon. A force is applied to the projectile, which causes it to be propelled in a straight trajectory until it encounters an obstacle or enemy. At this point, the system will remove the projectile.
    - Evading. The player can perform a dodge by calling the "Dodge" action, which is associated with the right mouse click and the 'space' key. This action has a two-second cooldown period, during which the system applies a small force to the player's character, propelling them a short distance.
    - Healing using a first-aid kit. The player can shoot at first-aid kits that are located on the map, which will cause their character to restore a certain amount of health points.
    - Opening the pause menu. Upon pressing the "Escape" key, the system displays a pause menu, which allows the player to return to the main menu or resume gameplay. However, this menu does not halt in-game time.

- Logic of enemies. The prototype comprises three distinct types of enemies, each exhibiting unique characteristics. However, the logic employed by these enemies when pursuing their targets is identical.
    - Types of enemies.
        - The first type of enemy is distinguished by its melee attacks, greater health than other enemies, and a movement speed comparable to that of the player.
        - The second type of enemy has a slower movement and a lower health than the first type. However, it can launch attacks from a distance with fast projectiles that inflict significant damage.
        - The final enemy type exhibits the highest mobility of the three, yet also the lowest amount of hit points. It attacks the player from a medium distance by firing projectiles with a high rate of fire.
    - Targeting the players and pursuing them. To pursue players, enemies employ the A* search algorithm, which utilises Euclidean distance as a heuristic.
    - Attack modes. Enemies present two distinct forms of attacking the player:
        - The first of these is a melee attack, where the enemy attempts to make contact with the player. Upon the occurrence of a hit, the player is subjected to a certain amount of damage.
        - The second form of attack is based on projectile fire. When the player is within firing range of the enemy, the enemy starts firing at the player character until it leaves this radius.
- Gameplay flow. A finite state machine, comprising four states, is employed to regulate the progression of the game.
    - S0 - Selecting Game Mode: the initial state at the commencement of the game, signifying that it originates from the game mode selection.
    - S1 - WaitingToStart: the state that is reached when the scene has been set, and all players have changed their state to ready.
    - S2 - In Game: once all players have changed their state to ready, the machine transitions to this state, which represents the entire course of the game.
    - S3 - Game Finished: this is the final state, which is reached when a player perishes or the game is won.
- Scene and menu flow. The developed prototype comprises a total of five distinct scenes:
    - Main Menu Scene. This is the main menu scene, which allows the player to select whether to commence a game in single-player mode, multiplayer mode or to exit the application.
    - Loading Scene. This scene is utilised as a transition between the other scenes, indicating to the player that the game has not malfunctioned.

    - Lobby Scene. Scene where players can create a lobby or join an existing one created by another player.
    - Player Selection Scene. This scene represents the moment at which a player joins a lobby. It displays the lobby code, a button to set the player's status to ready, and the option to change the player's username, which is currently used for aesthetic purposes.
    - Level 1 Scene. This corresponds to the game scene, where all the gameplay takes place.
- Multiplayer mode design. The implementation of the multiplayer mode has been achieved through the utilisation of Netcode for GameObjects (NGO), which has enabled the synchronisation of game elements via two distinct mechanisms:
    - Remote procedure calls. An RPC (Remote Procedure Call) is a mechanism that enables the invocation of a process located in a remote service. This allows, with NGO, the possibility of making calls to a client process from the server and vice versa [82].
    - NetworkVariables. A NetworkVariable in NGO is, as its name implies, a variable that is automatically synchronised between all members of a network, without the need of using an RPC to update its value in a remote host [83].

Evaluation

This section presents the test plan employed to verify the correct functioning of the prototype implemented and to ascertain the fulfilment of the requirements specified in the analysis phase. A total of 23 test cases have been created and are presented in tables 5.2 to 5.24.

Following the execution of the test plan, three different problems were identified. The first two, designated P-01 and P-02, are related to the usage of Netcode for GameObjects and the interactions this system performs with the Unity physics engine. P-03 is associated with the partial fulfilment of the common C# code conventions, due to time constraints and the delayed adoption of this standard.

Regulatory framework

The regulations and standards that apply to a project of this nature are the same that apply to any software development project. These laws and standards are mainly those related to intellectual property and data protection. In Spain, the laws that apply are as follows:

- The General Data Protection Regulation (GDPR) is a European Union (EU) regulation that establishes the rules for the protection of personal data and the free movement of software within the EU. It applies to all EU member states and is designed to ensure the protection of the personal data of EU citizens [85].
- The Organic Law on the Protection of Personal Data and Guarantee of Digital Rights is an adaptation of the GDPR to the territory of Spain. It also

includes the role of the Spanish Data Protection Agency (AEPD) in its enforcement [86].

The prototype developed in this project meets the regulations specified by both laws, as the program does not store users' data, aside from the username introduced by the player, and it only stores locally on its computer. Furthermore, the multiplayer is implemented using DTLS to ensure that player data is encrypted while communicating players with the Relay service.

The technical standards that can be applied to a video game are the same that can be applied to any other computer software. The most pertinent standards are as follows:

- UNE-ISO/IEC 90003. This standard, identical to ISO/IEC 90003:2004, provides a guide on quality management, software development and software maintenance [88].
- Common C# code conventions. These conventions, provided my Microsoft, present a set of standards to improve code maintenance, with the aim of increasing code consistency, readability and collaboration [89].

Planning

This section presents the planning that was carried out during the project's execution, which spanned a period of eight months. In order to demonstrate the tasks that were completed, a segmentation of the tasks was made. Table 7.1 shows the list of tasks that were completed, as well as the time dedicated to each one of them. The Gantt chart presented in Figure 7.1 was created using the identifiers of the tasks. It illustrates the temporal sequence of these tasks.

Socio-economic environment

This section presents a simulated budget of the project and analyses the potential impact that a project of this nature could have on society, the economy and the environment.

Budget

To calculate the budget of the project, each task identified in the planning section has been assigned to its corresponding role or job position. The salaries of the aforementioned roles have been calculated by multiplying the number of hours dedicated to each task by the average wage of the relevant job position. The aforementioned average wages have been sourced from Talent.com [99]. In addition to the salaries, the material costs and indirect costs have been considered, as well as the relevant taxes, a profit of 10% and a risk factor based on the 15% of the total. The breakdown of the salaries and material costs can be observed in Tables 8.2 and 8.3. Table II below shows the final budget estimated for this project.

TABLE II
BUDGET OF THE PROJECT

Item	**Value**
Salaries	5.535,64€

Item	Value
Material costs	2.403,96€
Indirect costs (5%)	396,98€
Profit (10%)	833,66€
Risk (15%)	1.250,49€
VAT (21%)	2.188,35€
TOTAL	**12.609,08€**

Socio-economic impact

The socio-economic impact has been analysed in accordance with the specific area affected:

- Sociocultural impact. Video games can contribute to the creation of new communities, allowing players to meet people with similar interests to their own. In addition, they also help to promote interest in cultural heritage, as video games are frequently inspired by historical or mythological events [102].
- Impact in education and health. They have the potential to reduce stress, enhance coordination and cognitive function in individuals. However, they can also have a negative impact, increasing sedentary lifestyles or creating an addiction to video games [102].
- Economic impact. Video games facilitate the creation of new employment opportunities and represent one of the most productive and profitable industries globally [102].
- Environmental impact. The utilisation of video games can result in an increase in electricity consumption, thereby increasing greenhouse gas emissions into the atmosphere. However, they can also be used to raise awareness of climate change.

Conclusions

This chapter presents a retrospective analysis of the project, examining the objectives to ascertain whether they have been met through the completion of this work. It also presents the problems encountered during its elaboration and some possible future improvements that can be made to the project.

Objectives achieved

Upon completion of the project, the primary objective has been achieved, having developed a fully functional prototype of a multiplayer video game with roguelike components.

Regarding the secondary objectives, all have been fulfilled, resulting in the accumulation of over 250 hours of experience in the Unity Engine and in C#. Furthermore, this project has served to expand the knowledge base in areas already

studied in the Computer Science degree and in other subjects that were previously not studied, such as game development and multiplayer networking.

Problems encountered

Part of the problems encountered during the development of the project are due to inadequate planning and lack of time, due to the fact that, during the whole project implementation, the student has been working on a part-time basis. On the other hand, the problems encountered during the implementation phase are associated with the lack of familiarity with the technologies employed. In addition, some issues encountered during the implementation of the multiplayer resulted in significant delays, as in some cases, a refactoring of the code was necessary.

Possible future improvements

A number of potential enhancements that can be made to the project are listed below, some of which were not implemented during the development phase due to time constraints:

- The implementation of client-side prediction and extrapolation.
- An algorithm to generate the levels procedurally.
- The introduction of new consecutive levels, new enemies and bosses.
- Controller support and mobile platforms support.
- New weapons and power-ups for the players.