



UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

**PROTOTIPO DE VIDEOJUEGO
RPG MULTIJUGADOR CON
GENERACIÓN PROCEDURAL
DE NIVELES**

Alumno: López Ruiz, José Luis

Tutor: Prof. D. Jiménez Pérez, Juan Roberto
Dpto: Informática

Junio, 2018



**Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática**

Don Juan Roberto Jiménez Pérez, tutor del Proyecto Fin de Carrera titulado:
Prototipo de videojuego RPG multijugador con generación procedural de niveles, que
presenta José Luis López Ruiz, autoriza su presentación para defensa y evaluación
en la Escuela Politécnica Superior de Jaén.

Jaén, Junio de 2018

José Luis López Ruiz

Juan Roberto Jiménez Pérez

Agradecimientos

Quiero dedicar este Trabajo Fin de Grado a todas las personas, que de un modo u otro, me han ayudado. Mis padres por confiar en mí, y sacrificarse para que pudiera llegar hasta aquí, a mi hermano por aguantarme todos los días con historias de Valhalla, y por la gran ayuda que me ha ofrecido último sprint, y por supuesto a la gran piña Vanessa por aguantarme a todas horas, animarme, quererme y ayudarme en todo lo que has podido.

Por último, a mi tutor por permitirme hacer este proyecto y asesorarme en todo momento.

Gracias a todos.

*"A veces la persona que nadie imagina capaz de nada es la
que hace cosas que nadie imagina"*

Resumen

Este Trabajo Fin de Grado ha consistido en el desarrollo de un prototipo de videojuego del género RPG (Juego de Rol), con un sistema multijugador en red, con una base de datos remota y con un sistema de niveles con ciertas componentes procedurales. Para realizarlo, se ha hecho uso del motor de videojuegos Unity en lenguaje C-Shard, y de otras tecnologías auxiliares como Photon y PlayFab, que permiten la implementación del sistema multijugador sin mantener ningún tipo de servidor remoto.

Abstract

This Final Project consists of the development of a prototype of an RPG game (Role-Playing Game), with an online multiplayer system, a remote database and a system of levels with certain procedural components. To achieve this, we've used the Unity game engine in C-Shard language, and other secondary technologies such as Photon and PlayFab, which allow the implementation of a multiplayer system without maintaining any type of remote server.

Índice

Índice de ilustraciones.....	11
Índice de tablas	14
Índice de ecuaciones	15
1. Introducción	16
1.1. <i>Objetivos del proyecto</i>	16
1.2. <i>Motivación</i>	17
1.3. <i>Estructura del documento</i>	19
1.4. <i>Visión general del videojuego</i>	19
1.5. <i>Definición de mecánica</i>	20
1.6. <i>Géneros de videojuegos</i>	20
2. Estado del arte	23
2.1. <i>Breve reseña histórica</i>	23
2.2. <i>Motores de videojuegos actuales</i>	24
2.2.1. Unreal Engine 4	25
2.2.2. Unity.....	28
2.2.3. CryEngine	30
2.3. <i>Elección del motor de videojuegos</i>	32
3. Planificación del proyecto	33
3.1. <i>Metodología</i>	33
3.2. <i>Cronología</i>	35
3.3. <i>Estimación de costes</i>	37
4. Diseño del juego	42
4.1. <i>Diagrama de casos de uso. Requisitos del sistema</i>	42
4.2. <i>Diseño de las mecánicas</i>	45
4.2.1. Mecánica de niveles	45
4.2.2. Mecánica libro de hechizos	48
4.2.3. Mecánicas de clase o rol	49
4.2.3.1. Atributos comunes a todas las clases.....	49
4.2.3.2. Definiendo en detalle los atributos.....	50
4.2.4. Mecánica de clase o rol: Cazador	52
4.2.4.1. Valores iniciales de los atributos.....	52
4.2.4.2. Incremento de los atributos al subir de nivel.....	53
4.2.4.3. Habilidades de clase	54
4.2.4.4. Pasivas de clase	62
4.2.5. Mecánica de clase o rol: Guerrero vikingo.....	62
4.2.5.1. Valores iniciales de los atributos	62
4.2.5.2. Incremento de los atributos al subir de nivel.....	63
4.2.5.3. Habilidades de clase	63
4.2.5.4. Pasivas de clase	71
4.2.6. Mecánicas de compañeros.....	71
4.2.6.1. Atributos del compañero	71

4.2.6.2.	Duración de la invocación	72
4.2.6.3.	Tipo de combate.....	72
4.2.7.	Mecánicas de objetos equipables	72
4.2.7.1.	Atributos.....	73
4.2.7.2.	Equipables comunes.....	74
4.2.7.3.	Equipables no comunes	75
4.2.7.4.	Asignación de recursos.....	75
4.2.8.	Mecánicas de suerte	76
4.2.8.1.	Factor suerte en los objetos equipables	77
4.2.8.2.	Factor suerte en las habilidades de los enemigos elites	79
4.2.9.	Mecánicas de enemigos	81
4.2.9.1.	Atributos comunes	81
4.2.9.2.	Tipos de enemigos.....	82
4.2.9.3.	Definiendo los enemigos	82
4.2.9.4.	Habilidades que pueden tener los enemigos raros.....	88
4.2.9.5.	Tablas de probabilidad de objetos equipables.....	89
4.2.10.	Mecánicas de economía	91
4.2.10.1.	Abalorios para obtener reliquias.....	91
4.2.10.2.	Mercaderes de la zona neutral.....	92
4.3.	<i>Modos de juego</i>	94
4.3.1.	Modo historia	96
4.3.2.	Modo Core	96
4.3.3.	Modo carrera	96
4.4.	<i>Diseño de la base de datos</i>	96
4.5.	<i>Prototipo de la interfaz</i>	100
4.6.	<i>Flowboard</i>	104
5.	Sistema multijugador: Photon Network	106
5.1.	<i>Arquitectura Photon Unity 3D Networking Framework</i>	107
5.2.	<i>Regiones</i>	110
5.3.	<i>Lobby y salas</i>	111
5.4.	<i>Emparejamiento aleatorio</i>	112
5.5.	<i>Emparejamiento no aleatorio</i>	112
5.6.	<i>Instanciación</i>	113
5.7.	<i>RPC</i>	113
6.	Sistema de autenticación y almacenamiento: PlayFab	114
6.1.	<i>Arquitectura de PlayFab</i>	115
6.2.	<i>Adaptación del diseño a PlayFab</i>	117
7.	Sistema de niveles procedurales.....	120
7.1.	<i>Instanciación de enemigos</i>	121
7.1.1.	Estructuras auxiliares para resolver el problema	124
7.1.2.	Pseudocódigo del algoritmo.....	125
7.1.3.	Resultados	125
7.2.	<i>Sistema de cofres</i>	128
7.3.	<i>Habilidades aleatorias de los enemigos elites y atributos de objetos aleatorios</i>	130
8.	Arquitectura y diseño software	130

8.1.	<i>Sistema de scripting en Unity</i>	132
8.2.	<i>Arquitectura cliente-servidor</i>	135
8.3.	<i>Patrones de diseño utilizados</i>	136
8.3.1.	Patrón de diseño Singleton	136
8.3.2.	Patrón de diseño Observer.....	137
8.3.3.	Patrón de diseño DAO	138
8.4.	<i>Diseño de clases</i>	139
8.4.1.	Paquete de clases relacionadas con entidades	149
8.4.2.	Paquete de clases controladoras de movimiento	154
8.4.3.	Paquete de clases relacionadas con habilidades o hechizos.....	155
8.4.4.	Paquete de clases relacionadas con los objetos del juego.....	160
8.4.5.	Paquete de clases controladoras de la interfaz	161
8.4.6.	Paquete de clases de estructuras personalizadas.....	166
8.4.7.	Paquete de clases para el editor de Unity.....	167
8.4.8.	Paquete de clases de geometría	169
8.4.9.	Paquete de clases de utilidades JSON	169
8.4.10.	Paquete de clases relacionadas con el sistema en red	171
8.5.	<i>Comunicación entre clases</i>	175
9.	Prototipo final Valhalla	176
9.1.	<i>Modos de juego</i>	176
9.2.	<i>Escenas o niveles disponibles</i>	176
9.3.	<i>Personajes</i>	179
9.4.	<i>Enemigos</i>	186
9.4.1.	Vikingo.....	186
9.4.2.	Vikingo élite.....	187
9.4.3.	Berseker	188
9.4.4.	Jefe final Thor	189
9.5.	<i>Clases o roles</i>	191
9.6.	<i>Objetos que se pueden romper</i>	193
9.7.	<i>Interfaz definida</i>	194
9.7.1.	Interfaz del menú principal	194
9.7.2.	Interfaz del resto del juego	196
9.8.	<i>Música y sonidos</i>	198
9.9.	<i>Controles del prototipo</i>	198
10.	Mejora en el sistema de colisiones de Unity	200
11.	Control de versiones de prototipos	201
11.1.	<i>Prototipo v1.0</i>	202
11.2.	<i>Prototipo v1.1</i>	203
11.3.	<i>Prototipo v1.2</i>	204
11.4.	<i>Prototipo v1.3</i>	207
11.5.	<i>Prototipo v2.0</i>	209
12.	Evaluaciones de los prototipos	210
12.1.	<i>Primera evaluación</i>	210

12.2. <i>Segunda evaluación</i>	218
13. Conclusiones finales.....	226

Índice de ilustraciones

Ilustración 1. Juego Clash Royale. Foto izquierda con una partida 2 vs 2, foto central con una tabla de clasificaciones y foto derecha con la apertura de un cofre con cartas aleatorias	18
Ilustración 2. Editor de Unreal Engine 4	27
Ilustración 3. Editor de Unity	30
Ilustración 4. Proceso de una metodología ágil	34
Ilustración 5. Caso de uso para el menú principal	43
Ilustración 6. Caso de uso jugar	44
Ilustración 7. Gráfico que representa la función	47
Ilustración 8. Libro de hechizos de World of Warcraft	48
Ilustración 9. Clases de diablo 3. De izquierda a derecha: guerrero, cazador de demonios, monje, médico brujo y mago	49
Ilustración 10. Ataque básico	55
Ilustración 11. Ataque explosivo	56
Ilustración 12. Multidisparo	57
Ilustración 13. Lanzamiento envenenado	58
Ilustración 14. Trampa congelante	59
Ilustración 15. Salto energético	60
Ilustración 16. Rugido salvaje	61
Ilustración 17. Ataque básico	64
Ilustración 18. Ataque secundario del rezo de Thor	65
Ilustración 19. Ataque torbellino	67
Ilustración 20. Grito ensordecedor	68
Ilustración 21. Lanzamiento heroico	69
Ilustración 22. Terremoto	71
Ilustración 23. Objetos equipados en el videojuego Diablo 3. A la izquierda aparece información acerca de los atributos	73
Ilustración 24. Vista Top-down en el videojuego Diablo 3	94
Ilustración 25. Vista Top-down en el videojuego Path of Exile	95
Ilustración 26. Modelo entidad-relación de la base de datos	99
Ilustración 27. Prototipo de la interfaz. Menú principal	101
Ilustración 28. Prototipo de la interfaz. Dentro del juego en modo historia	102
Ilustración 29. Prototipo de la interfaz. Dentro del juego en modo carrera	103
Ilustración 30. Prototipo de la interfaz. Dentro del juego en modo core	104
Ilustración 31. Flowboard	105
Ilustración 32. Información general de Photon	108
Ilustración 33. Juego de padre de familia realizado con unity y photon	109
Ilustración 34. Arquitectura de la conexión con regiones	110
Ilustración 35. Sistema de salas y lobbies	112
Ilustración 36. Panel de información de una cuenta de PlayFab	117
Ilustración 37. Ejemplo de información guardada sobre un usuario en formato JSON	118
Ilustración 38. Mapa de los niveles o zonas del juego World of Warcraft. En el mapa hay un grafo con el orden que siguen las zonas por complejidad	121
Ilustración 39. Ejemplo de restricción de 4 enemigos como máximo en la zona de influencia. Los puntos rojos son los enemigos que se están instanciando y los verdes los enemigos instanciados. La imagen de la izquierda no cumple la restricción y la derecha sí	123
Ilustración 40. Primera instancia procedural	126
Ilustración 41. Segunda instancia procedural	127
Ilustración 42. Tercera instancia procedural	127
Ilustración 43. Conexiones del lateral izquierdo del mapa donde se realizó la instancia	128
Ilustración 44. Conexiones del lateral derecho del mapa donde se realizó la instancia. El círculo rojo es el punto de respawn del jugador	128
Ilustración 45. Posibles posiciones de los cofres en el nivel uno	129
Ilustración 46. Sistema de clases para las entidades en el diseño inicial en papel	130
Ilustración 47. Atributos de la clase entidad en el diseño inicial	131
Ilustración 48. Callbacks y orden de llamadas en la vida de un script en Unity	134
Ilustración 49. Diferencia entre cliente servidor y cliente normal	136
Ilustración 50. UML de la clase que representa el patrón observador	138
Ilustración 51. Interfaz común para DAO	139

Ilustración 52. UML de las clases del tipo enum y de la clase ValCommonFunctions con métodos estáticos que son usados por un gran número de clases	141
Ilustración 53. UML de la clase ValConstants que contiene las constantes de Valhalla.....	143
Ilustración 54. UML clase ValGame.....	145
Ilustración 55. UML de la clase ValInfoPlayerManager.....	146
Ilustración 56. UML de la clase ValGameManager	147
Ilustración 57. UML de las clases relacionadas con entidades.....	149
Ilustración 58. UML de las clases controladoras de movimiento.....	154
Ilustración 59. UML de las clases relacionadas con los hechizos o habilidades	155
Ilustración 60. Lista de hechizos del componente ValPlayerHunter	156
Ilustración 61. UML de las clases relacionadas con los objetos	161
Ilustración 62. UML de clases relacionadas con la interfaz del menú principal	162
Ilustración 63. UML de clases relacionadas con la interfaz dentro del juego	165
Ilustración 64. UML de las clases de estructuras personalizadas	166
Ilustración 65. UML con las clases que personalizan el editor de Unity	168
Ilustración 66. Editor personalizado para la clase ValPlayerHunter	168
Ilustración 67. UML con la clase de geometría 2D	169
Ilustración 68. UML de clases con utilidades JSON	170
Ilustración 69. UML con las clases del sistema multijugador y la base de datos	171
Ilustración 70. Comunicación entre las componentes lógicas y de la interfaz de un personaje	175
Ilustración 71. Primera escena: Menú principal	177
Ilustración 72. Escena de la zona neutral vista desde el modo editor. Los círculos verdes son NPCs con los cuales se puede hablar, y el círculo rojo es donde son colocados los jugadores inicialmente	178
Ilustración 73. Valores de los atributos de la clase maestra de juego en el nivel uno	178
Ilustración 74. Nivel uno visto desde el editor	179
Ilustración 75. Modelo 3D del Dios Valiel	179
Ilustración 76. Sistema de animaciones para el Dios Valiel	180
Ilustración 77. Interfaz del Dios Valiel	180
Ilustración 78. Modelo 3D del Dios Ull	181
Ilustración 79. Interfaz del Dios Ull	181
Ilustración 80. Modelo 3D del posadero	182
Ilustración 81. Sistema de animaciones para el posadero	182
Ilustración 82. Interfaz del posadero	183
Ilustración 83. Modelo 3D del gigante Ysvar	184
Ilustración 84. Interfaz del gigante Ysvar	184
Ilustración 85. Modelo 3D del barquero	185
Ilustración 86. Interfaz del barquero	185
Ilustración 87. Modelo 3D de los espectros	185
Ilustración 88. Modelo 3D del enemigo vikingo.....	186
Ilustración 89. Sistema de animaciones para el enemigo Vikingo	187
Ilustración 90. Modelo 3D del vikingo élite	188
Ilustración 91. Modelo 3D del enemigo berseker	189
Ilustración 92. Modelo 3D del Dios Thor	190
Ilustración 93. Subida de Thor a la plataforma Mixamo para la obtención de un sistema de huesos	190
Ilustración 94. Sistema de animaciones del jefe final Thor.....	191
Ilustración 95. Modelo 3D del cazador	192
Ilustración 96. Sistema de animaciones para el cazador	192
Ilustración 97. Modelo 3D del lobo Sköll	193
Ilustración 98. Sistema de animaciones del lobo Sköll	193
Ilustración 99. Modelo 3D de la columna que puede romperse	194
Ilustración 100. Panel con el formulario para iniciar sesión	195
Ilustración 101. Mensaje de error al equivocarse en las credenciales	195
Ilustración 102. Mensaje de error cuando no hay conexión a Internet	195
Ilustración 103. Panel con el formulario de registro	196
Ilustración 104. Panel para seleccionar un personaje de la lista y comenzar una partida	196
Ilustración 105. Panel para la creación de un personaje.....	196
Ilustración 106. Interfaz principal dentro del juego	197
Ilustración 107. Interfaz del libro de hechizos	197
Ilustración 108. Interfaz del inventario	198
Ilustración 109. Interfaz del menú dentro de una partida	198

Ilustración 110. Matriz de colisiones final del proyecto	201
Ilustración 111. Imagen del prototipo v1.0 en el escenario para testear.....	203
Ilustración 112. Nuevo menú principal	204
Ilustración 113. Jugador en la escena de testeo con la nueva interfaz y con otro jugador al lado	204
Ilustración 114. Jugador en la zona neutral	205
Ilustración 115. Nueva interfaz para libro de hechizos	206
Ilustración 116. Interfaz del barquero para cambiar de nivel, y la notificación que envía a todos los jugadores para poder cambiar.....	206
Ilustración 117. Nueva interfaz para el inventario. Zona superior izquierda atributos actuales del jugador, zona superior derecha objetos armadura equipados y la zona inferior para la mochila y las calaveras conseguidas ..	207
Ilustración 118. Nivel de testeo con objetos que han dejado caer los enemigos	207
Ilustración 119. Nuevo menú principal para poder autentificar a un jugador con una cuenta	208
Ilustración 120. Jugador en la zona neutral modificada y con la interfaz definitiva	209

Índice de tablas

Tabla 1. Visión general del videojuego.....	20
Tabla 2. Plataformas disponibles para Unreal Engine 4.....	26
Tabla 3. Plataformas disponibles para Unity.....	29
Tabla 4. Plataformas disponibles para CryEngine	32
Tabla 5. Tipos de tareas	35
Tabla 6. Estimación de costes de materiales	38
Tabla 7. Estimación de costes de software.....	39
Tabla 8. Estimación de costes de servicios	39
Tabla 9. Estimación de costes de hardware.....	41
Tabla 10. Estimación de costes de personal	42
Tabla 11. Estimación de costes total	42
Tabla 12. Tabla de niveles con los puntos de experiencia necesarios para subir al siguiente nivel	47
Tabla 13. Tabla de equivalencias	74
Tabla 14. Tabla con los atributos que puede tener cada pieza	76
Tabla 15. Ejemplo de tabla de pesos para un enemigo.....	77
Tabla 16. Ejemplo de tabla de pesos para una pieza de armadura	78
Tabla 17. Tabla de probabilidades de atributos perfectos	79
Tabla 18. Tabla definida para el tipo de funcionamiento	80
Tabla 19. Tabla definida para el tipo de uso	80
Tabla 20. Primer nivel de la tabla de pesos para enemigos normales	90
Tabla 21. Tabla anidada de pesos para la pieza de armadura en enemigos normales	90
Tabla 22. Tabla anidada de pesos para las armas de la clase cazador en enemigos normales	90
Tabla 23. Tabla anidada de pesos para las armas de la clase guerrero vikingo en enemigos normales	91
Tabla 24. Primer nivel de la tabla de pesos para enemigos élites	91
Tabla 25. Primer nivel de la tabla de pesos para los jefes finales	91
Tabla 26. Tabla de ventas	93
Tabla 27. Tabla del mercader.....	93
Tabla 28. Tabla con el listado de regiones	110
Tabla 29. Respuestas de la primera pregunta del formulario del prototipo v1.1	211
Tabla 30. Respuestas de la segunda pregunta del formulario del prototipo v1.1	212
Tabla 31. Respuestas de la tercera pregunta del formulario del prototipo v1.1	212
Tabla 32. Respuestas de la cuarta pregunta del formulario del prototipo v1.1	213
Tabla 33. Respuestas de la quinta pregunta del formulario del prototipo v1.1	213
Tabla 34. Respuestas de la sexta pregunta del formulario del prototipo v1.1	214
Tabla 35. Respuestas de la séptima pregunta del formulario del prototipo v1.1	215
Tabla 36. Respuestas de la octava pregunta del formulario del prototipo v1.1	215
Tabla 37. Respuestas de la novena pregunta del formulario del prototipo v1.1	216
Tabla 38. Respuestas de la décima pregunta del formulario del prototipo v1.1	216
Tabla 39. Respuestas de la décima primera pregunta del formulario del prototipo v1.1	217
Tabla 40. Respuestas de la primera pregunta del formulario del prototipo v2.0.....	220
Tabla 41. Respuestas de la segunda pregunta del formulario del prototipo v2.0	220
Tabla 42. Respuestas de la tercera pregunta del formulario del prototipo v2.0	221
Tabla 43. Respuestas de la cuarta pregunta del formulario del prototipo v2.0	221
Tabla 44. Respuestas de la quinta pregunta del formulario del prototipo v2.0	222
Tabla 45. Respuestas de la sexta pregunta del formulario del prototipo v2.0	222
Tabla 46. Respuestas de la séptima pregunta del formulario del prototipo v2.0	223
Tabla 47. Respuestas de la octava pregunta del formulario del prototipo v2.0	224
Tabla 48. Respuestas de la novena pregunta del formulario del prototipo v2.0	224
Tabla 49. Respuestas de la décima pregunta del formulario del prototipo v2.0	225
Tabla 50. Respuestas de la décima primera pregunta del formulario del prototipo v2.0	225
Tabla 51. Respuestas de la décima segunda pregunta del formulario del prototipo v2.0.....	226

Índice de ecuaciones

Ecuación 1. Función para calcular la experiencia necesaria en cada nivel	46
Ecuación 2. Función sumatoria para calcula la experiencia necesaria para subir al nivel máximo.....	47

1. Introducción

Mi trabajo fin de grado consiste en la implementación de un prototipo de videojuego, más concretamente un videojuego con un sistema multijugador del tipo RPG y con un sistema de niveles con diversas componentes procedurales o de un factor suerte.

La elección de este tipo de proyecto ha sido fundamentalmente por dos motivos, en primer lugar considero que es un producto software muy completo, tanto desde el punto de vista de la informática gráfica (animaciones, iluminación, efectos visuales, etc.) como en otros muchos aspectos que intervienen en un videojuego, como puede ser la interfaz con la que interacciona el usuario, bases de datos para almacenar información no volátil, entre otras muchas cosas. En segundo lugar dada mi experiencia como usuario de estos productos, y que he cursado todas las asignaturas de la mención de gráficos, es una buena oportunidad para aprovechar mi faceta de jugador y desarrollador para realizar un prototipo un videojuego.

1.1. Objetivos del proyecto

Los objetivos principales del este trabajo fin de grado son:

- Identificar el juego a desarrollar, destacando las características principales del mismo así como el público objetivo.
- Definir las mecánicas de juego principales.
- Seleccionar el motor de juego más adecuado así como otras herramientas complementarias que se requieran en el desarrollo o interacción con el juego específico.
- Analizar y diseñar la interfaz e interacción con el usuario, así como identificar los dispositivos de interacción más adecuados al tipo de videojuego que se propone.
- Analizar y diseñar el mundo sobre el que se desarrolla, los personajes, animaciones, efectos visuales y de sonido.
- Diseñar e implementar un sistema multijugador en el que sean capaz de jugar varios jugadores a la vez.
- Crear al menos un nivel con componentes procedurales.

- Evaluar el prototipo.

1.2. Motivación

Dentro de un gran abanico de posibilidades la elección de este proyecto ha sido motivada por diversos factores. En las últimas décadas, la industria de los videojuegos ha ido en aumento, hasta tal punto de generar miles de millones de euros anualmente. Pero la creación de un videojuego no implica necesariamente que te reporte unos beneficios. Para ello se debe crear un software en primer lugar, de calidad y que contenga una serie de componentes que resulten únicos y atractivos para el jugador.

Si se analiza con detenimiento aquellos videojuegos que están actualmente en el mercado que tienen una gran popularidad y reportan una gran cantidad de beneficios, se puede observar una serie de componentes comunes:

- Tienen una componente online en la que permite completar contenido junto a otros jugadores, o bien permite enfrentarse a otros jugadores o incluso ambas. También suelen incluir algún modo de comunicación entre los jugadores como por ejemplo un chat.
- Tienen una componente de progresión del jugador a través de niveles e historia que van desbloqueando contenido.
- Tienen una componente de factor suerte en mayor o menor medida para no aburrir con el mismo contenido al usuario.

Por ese mismo motivo he decidido incluir en mi proyecto un sistema multijugador como un componente social y por otro lado, la generación de niveles procedurales. Un nivel procedural es aquel que varía en alguno de sus aspectos y no se mantiene de manera estática cada vez que el usuario decida volver a jugar ese nivel. Estos cambios pueden realizarse en diversos aspectos de nivel: todo lo relacionado con la instanciación de enemigos, generación de terrenos, etc. Estos cambios mantienen siempre alerta a los jugadores, permiten que el jugador no caiga en la monotonía y el aburrimiento, y proporcionan una experiencia única en cada ejecución.

Otros componentes que son dignos de mencionar, pero que no he incluido en la idea del proyecto porque quizás se escape un poco de las dimensiones esperadas para un trabajo fin de grado, son los sistemas de clasificaciones de jugadores. Estos

sistemas fomentan una competitividad y supone un gasto en recursos para llegar a ser los mejores y que se suele regir por algún algoritmo de emparejamiento y puntuación. Por otro lado, está también el modo de pago que suele diferenciarse en dos grandes grupos: el sistema de pago clásico a través de un único pago (Diablo III) o bien pago mensual (World of Warcraft), y el nuevo sistema de pago de juegos gratuitos que permite jugar a cualquier usuario de manera gratuita con el inconveniente de que para avanzar más rápido o desbloquear nuevo contenido tienen que realizar micropagos. El sistema de juegos gratuitos con micropagos es cada vez más común y a la larga proporciona un mayor beneficio que el sistema clásico.

Como ejemplo de juegos con dichas componentes puede ser Clash Royale (ver Ilustración 1), un juego con un sistema multijugador online, con diversas componentes de suerte, como puede ser la apertura de cofres con cartas aleatorias, un sistema de progresión de niveles, clasificación de jugadores y que además, es gratuito con un sistema de micropagos para poder avanzar más rápido y obtener una mejor puntuación en la clasificación global. Supercell, la empresa creadora de este juego, llegó a recaudar 2.3 billones de dólares en 2016 [1].



Ilustración 1. Juego Clash Royale. Foto izquierda con una partida 2 vs 2, foto central con una tabla de clasificaciones y foto derecha con la apertura de un cofre con cartas aleatorias

1.3. Estructura del documento

El documento presenta una estructura secuencial en el que se muestra el progreso del proyecto. Está compuesto de 14 secciones que siguen un orden secuencial del proyecto, y se puede decir que se distingue en cuatro partes:

- Introducción y fase inicial de investigación de tecnología.
- Fase de diseño.
- Fase de implementación.
- Análisis y conclusiones.

En la parte inferior a pie de página, aparecerán anotaciones con información extra y definiciones sobre términos en el mundo de los videojuegos que he considerado relevantes y pueden llevar a confusión. También se ha añadido ilustraciones para ayudar a una mejor compresión de lo que aquí se expone, o mostrar algunos ejemplos.

Se podrá acceder a cualquier sección, ilustración, tabla o ecuación haciendo click en cualquier parte del índice.

1.4. Visión general del videojuego

A continuación se proporciona una visión general del videojuego a través de la siguiente tabla (ver Tabla 1). El sistema de clasificación utilizado ha sido PEGI [2].

Nombre del videojuego	Valhalla
Género	Videojuego de rol (RPG)
Subgénero	Videojuego de rol de acción (ARPG)
Idioma	Español
Plataforma	PC Windows
Modos de juego	Un jugador, multijugador
Similitud con otros videojuegos	Diablo III ¹ , Path of Exile ²
Controles	Teclado y ratón

¹ <https://eu.diablo3.com/es/>

² <https://www.pathofexile.com/game>

Conexión a Internet	Es necesaria la conexión a Internet
Clasificación PEGI	
Otros	3D, top-down
Breve introducción	
<p>El juego dispondrá de una serie de niveles generados proceduralmente que permitirán al jugador sumergirse en la historia de Valhalla a través de ciertas clases de personajes que dispondrán de una serie de habilidades únicas que podrán equiparse los objetos que proporcionan como recompensa los niveles. Está ambientado e inspirado en la mitología nórdica.</p>	

Tabla 1. Visión general del videojuego

1.5. Definición de mecánica

Las mecánicas de un juego son el factor más importante y determinante de éste. Una definición de mecánica podría ser: “Son cualquier acción realizada por el jugador que modifique el estado del juego en un momento preciso del tiempo”. Muchos investigadores en el campo de los videojuegos han dado diversas definiciones de lo que ellos entienden por mecánicas, por ejemplo Elliot Avedon da una definición de mecánica en su libro: *“Una mecánica describe lo que un jugador puede hacer, cómo lo hace y las reglas que gobiernan esas acciones”* [3].

1.6. Géneros de videojuegos

Los géneros de videojuegos es una forma de clasificar los videojuegos a través de una característica fundamental, las mecánicas de las que está compuesto el videojuego. También pueden influir otros factores como puede ser la temática y la estética entre otros, pudiendo estar en varios géneros simultáneamente.

La primera clasificación en géneros fue en 1984 por Chris Crawford, donde hace dos grandes grupos que a su vez tienen una serie de subgrupos [4]. También hace una clara distinción entre un juego y un puzzle. Un puzzle es algo estático e invariante en el tiempo que una vez solucionado se pierde el conflicto y la dificultad para el jugador. En cambio un juego es algo dinámico y que varía cada vez que se

juega. La primera clasificación en dos grandes grupos fue: juegos de habilidad y acción, y juegos de estrategia y cognitivos. Estos dos grandes grupos se subdividen en otros subgéneros:

- Juegos de habilidad y acción:
 - **Juegos de combate:** se presenta un conflicto violento donde el jugador debe derrotar a los enemigos (Missile Command).
 - **Juegos de laberinto:** el jugador debe recorrer una estructura laberíntica, y en ocasiones perseguido por enemigos (Pacman).
 - **Juegos de deportes:** tratan de emular una experiencia de deporte real (Pole Position).
 - **Juegos de paleta:** juegos basados en PONG donde el jugador tiene que mover una paleta para hacer rebotar una pelota.
 - **Juegos de carreras:** el objetivo es mover a un avatar para ganar una carrera (Pole Position, Night Driver).
- Juegos de estrategia y cognitivos:
 - **Juegos de aventuras:** el jugador está situado en un mundo complejo donde puede conseguir herramientas y tesoros para poder conseguir el objetivo final (The Wizard and the Princess).
 - **Juegos de dragones y mazmorras:** el jugador es guiado por un maestro de mazmorras a través de una historia creada por él, donde pone las reglas y las situaciones o eventos que se producen (Dungeons & Dragons).
 - **Juegos de guerra:** juegos complejos con un conflicto entre dos o más jugadores que deben resolver y donde existen muchas variables y modificadores (Computer Napoleonics).
 - **Juegos educacionales:** la intención principal es transmitir algún conocimiento o trabajar alguna habilidad.

Con el paso de los años, la clasificación con los tipos de juegos se ha visto incrementada con la salida de una gran cantidad de videojuegos, generando nuevos grandes grupos más específicos y por tanto, a su vez, nuevos subgrupos [5] [6].

Actualmente, dentro de los grandes grupos de géneros, podemos encontrar: **videojuegos de acción** que requieren del jugador reflejos, puntería y habilidad, normalmente dentro de un contexto de combate o incluso superación de obstáculos y peligros, **videojuegos arcade** que más que un género en sí, se trata de un calificativo que engloba los típicos juegos de máquinas recreativas, **videojuegos de plataformas** donde el jugador controla un personaje que debe esquivar obstáculos para llegar a una meta final, **videojuegos de disparos** considerados una gran subcategoría dentro de los videojuegos de acción y conocidos coloquialmente como *shooters* donde el personaje hace uso de armas de fuego, **videojuegos de estrategia** que se caracterizan por la manipulación de numerosos grupos de personajes, objetos u incluso datos, teniendo como un factor importante la planificación y la inteligencia para lograr los objetivos, **videojuegos de simulación** que recrea situaciones o actividades del mundo real, dejando al jugador tomar el control y tomar decisiones, **videojuegos de deportes** que simulan deportes del mundo real como por ejemplo el fútbol o el tenis, **videojuegos de carreras** consisten en competiciones de 2 o más jugadores, o incluso jugadores artificiales, donde el objetivo es quedar primero, son considerados en ocasiones un subgrupo de los videojuegos de simulación y deportes como puede ser por ejemplo los videojuegos de fórmula 1, **videojuegos de aventura** en los que el protagonista avanza en una trama interactuando con personajes y objetos, **videojuegos de rol** o RPG (Role-Playing Game) que será explicado un poco más en detalle a continuación, y por último otros géneros destacables como sandbox, musical, agilidad mental, party games y educación.

Todos los grandes grupos mencionados con anterioridad contienen a su vez diversos subgéneros que comparten la misma temática pero un poco más especializada, como puede ser el subgénero de estrategia por turnos, que tiene como diferencia la existencia de turnos para poder realizar acciones.

El tipo de género RPG o rol es uno de los géneros principales en los que está basado mi proyecto. Este género está muy relacionado con los de aventura y principalmente, se caracteriza por la interacción con el personaje, una historia profunda y una evolución del personaje a través de la misma historia, toma de decisiones, progreso a través de niveles que desbloquean habilidades, elección de un rol y la mejora del personaje mediante objetos que le proporcionan unas características. Este tipo de género surgió a partir de los juegos de cartas de rol del

estilo de Munchkin³ o Dragones y Mazmorras⁴. El otro género en el que está basado mi proyecto es el subgénero videojuego de rol de acción, y que quizás lo define mejor, que se caracteriza por compartir propiedades comunes con los RPG pero que, a diferencia de éstos, ofrecen combates en tiempo real, con sistema de combate totalmente diferente y que no se centra tanto en la historia pero sí en el progreso del personaje.

2. Estado del arte

En esta sección se incluye la información previa que se ha recopilado antes de comenzar el proyecto. En este caso para profundizar y analizar cada uno de los motores de videojuegos en 3D disponibles y posteriormente elegir la mejor herramienta para poder desarrollarlo.

Por definición técnica, un motor de videojuego es un sistema diseñado para la creación y desarrollo de videojuegos, basado en rutinas de programación que crean las bases para la construcción de un videojuego. Por ende, este tipo de software suele tener ya definidos muchos componentes que nos llevarían bastante tiempo implementarlos por nuestra cuenta como por ejemplo sistema de partículas, animación, colisiones, iluminación, etc.

2.1. Breve reseña histórica

Los primeros motores de videojuegos surgieron como consecuencia del desarrollo de diversos videojuegos a finales de la década de los 80 y durante la década de los 90. Algunos sirvieron para desarrollar nuevos videojuegos, pero la gran mayoría se quedaron obsoletos al poco tiempo debido a los grandes avances que se producían, tanto en hardware como en software. Anteriormente a esto existían videojuegos para consolas recreativas.

Algunos de los motores de videojuegos [7] de la década de los 90, y precursores de los motores actuales, son los siguientes:

- **ZZT**: creado por Tim Sweeney en 1991 a través de su empresa Epic Games. Era un juego y un motor de juegos para DOS. Posteriormente sería el responsable de la creación de la saga del motor Unreal.

³ [https://es.wikipedia.org/wiki/Munchkin_\(juego_de_cartas\)](https://es.wikipedia.org/wiki/Munchkin_(juego_de_cartas))

⁴ https://es.wikipedia.org/wiki/Dungeons_%26_Dragons

- **Wolf3D-Engine:** llegó el 16 de julio de 1992 y fue creado por John Carmack. Este motor surgió de la creación del juego Wolfenstein 3D, que creó el género FPS y además, fue considerado el primer juego en 3D. Las especificaciones del motor eran muy buenas para su época: ray casting, resolución 320x320 a 256 colores, etc.
- **Doom Engine:** surge un año y medio más tarde, de nuevo a manos de John Carmack. Esta nueva versión mejorada no añade nuevas capacidades gráficas significativas: animación en luces, agujeros en paredes, etc. El gran cambio con respecto al anterior motor es el soporte para multijugador. Posteriormente este motor evolucionó en otro motor llamado Build que fue muy popular durante años.
- **Westwood 2D:** vio la luz en 1995, pero fue ampliando y evolucionando, contando en 1999 con variaciones de altura del terreno jugable en tiempo real, iluminación dinámica para pasar del día a la noche, efectos especiales, etc.
- **Quake Engine:** en 1996 vuelve John Carmack con el juego Quake. Esta versión incluye mejoras como el sombreado de Gouraud en iluminación y los lightmaps estáticos para objetos que no se mueven y que se siguen utilizando hoy en día.

En esta época los ordenadores más potentes para el consumo son los Pentium 100 Mhz con tarjetas S3 o Matrox de 1 o 2 MB de memoria. Además, no existen plataformas gráficas realmente estables y prácticamente hay que preparar los videojuegos para cada una de las plataformas gráficas.

2.2. Motores de videojuegos actuales

Actualmente, hay diversos motores de videojuegos, cada uno con un tipo de licencia y requisitos, con diferentes tipos de portabilidad, especializados en juegos 2D o 3D (o ambos) entre otros componentes más específicos como puede ser la iluminación o los sistemas de partículas.

Dadas las características de mi videojuego, he analizado tres motores de videojuegos que permiten desarrollar videojuegos en 3D y al menos dan portabilidad

para ordenador Windows, y por supuesto me permiten desarrollar gratuitamente el proyecto. Los tres motores son: **Unreal Engine 4, Unity y CryEngine**.

2.2.1. Unreal Engine 4

Unreal Engine 4 es un motor gráfico creado por la empresa Epic Games y que ha sido el resultado evolutivo de las anteriores versiones que ha ido sacando dicha empresa en los años 90. El motor Unreal Engine 4 como tal, ha derivado directamente de una serie de versiones hasta por fin llegar a la actual, permitiendo un acceso íntegro al software sin ningún tipo de restricciones a partir de marzo de 2014. El editor que permite el desarrollo de videojuegos en el motor está disponible para Mac OS X, Windows e incluso Linux.



Las plataformas [8] para las que puede dar soporte a cualquier videojuego que se desarrolle en dicho motor queda definido en la siguiente tabla (ver Tabla 2).

Plataforma	¿Está soportado por el motor gráfico?
Windows PC	Sí
Mac OS X	Sí
Linux	Sí
PlayStation 4	Sí

Xbox One	Sí
Nintendo Switch	Sí
Nintendo 3DS	No
PlayStation Vita	No
Wii U	No
VR ⁵	Sí
SteamOS	Sí
HTML 5	Sí
Android	Sí
iOS	Sí
Windows Phone	No
Tizen	No
Android TV	No
Facebook Gameroom	No

Tabla 2. Plataformas disponibles para Unreal Engine 4

Desde el punto de vista económico, este motor es una herramienta totalmente gratuita, todas y cada una de las características están completamente disponibles, pero a partir de los \$3.000 de beneficios la compañía se llevará un 5% de la totalidad que hemos obtenido.

Otro de los factores a tener en cuenta, es la comunidad y la calidad de la documentación. La documentación de Unreal Engine 4 es muy completa y accesible y además, hay una serie de guías y tutoriales para principiantes (incluidos vídeos) totalmente gratuitos que ayudan en la iniciación de dicho motor. Por otro lado, la comunidad todavía está en crecimiento y por tanto, es difícil encontrar soluciones a problemas que vayan surgiendo u obtener respuesta a dudas que se puedan plantear.

⁵ Siglas de *Virtual Reality*. Dentro de las plataformas de realidad virtual están SteamVR/HTC Vive, Oculus Rift, PlayStation VR, Google VR/Daydream, OSVR y Samsung Gear VR.

Esto puede ser debido a su complejidad y también en parte al poco tiempo que lleva como un producto gratuito y libre.

Para empezar con Unreal Engine se recomienda experiencia previa en otros motores. El editor del motor es muy parecido a Unity (ver Ilustración 2), tanto, que en su misma página se encuentra un artículo donde compara todos los elementos del editor de Unity con el de Unreal.

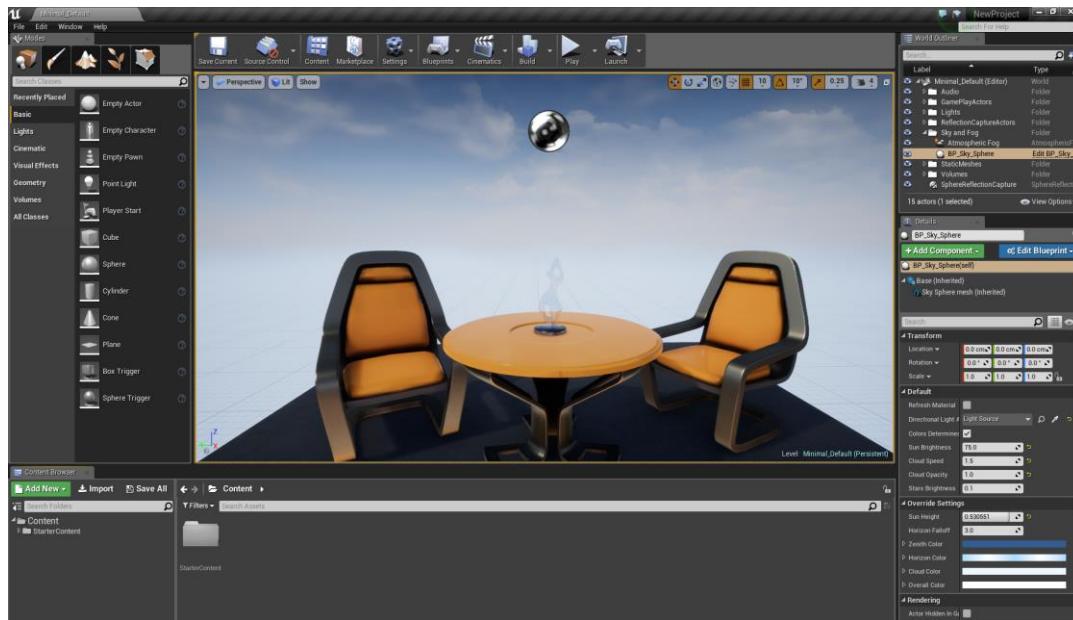


Ilustración 2. Editor de Unreal Engine 4

El lenguaje utilizado en dicho motor es C++, quizás un poco más complejo que otros motores como puede ser Unity con C#. También existe una característica llamada blueprint que permite programar el videojuego pero sin necesidad de ser un programador, todo se realiza a través de una interfaz gráfica sin necesidad de escribir ni una línea de código.

Al igual que los demás motores, incluye los componentes básicos: iluminación, sistema multijugador, cámaras, sistemas de partículas, etc. Cabe destacar que recientemente consiguieron reflejos⁶ (raytracing) en tiempo real.

Este motor ha servido para crear juegos como Batman: Arkham Series, Street Fighter V, Daylight o Fortnite.

⁶ <https://www.youtube.com/watch?v=J3ue35ago3Y>

2.2.2. Unity

Unity es uno de los motores gráficos más usados por desarrolladores de juegos independientes. Su primera versión fue liberada en 2005. Al igual que Unreal está disponible para Mac OS X, Windows y Linux.



Las plataformas [9] para las que puede dar soporte a cualquier videojuego que se desarrolle en dicho motor queda definido en la siguiente tabla (ver Tabla 3).

Plataforma	¿Está soportado por el motor gráfico?
Windows PC	Sí
Mac OS X	Sí
Linux	Sí
PlayStation 4	Sí
Xbox One	Sí
Nintendo Switch	Sí
Nintendo 3DS	Sí
PlayStation Vita	Sí
Wii U	Sí
VR	Sí
SteamOS	Sí
HTML 5	Sí
Android	Sí
iOS	Sí

Windows Phone	Sí
Tizen	Sí
Android TV	Sí
Facebook Gameroom	Sí

Tabla 3. Plataformas disponibles para Unity

A diferencia de Unreal, Unity dispone de una versión gratuita que limita las características de las que dispone este motor. Hay varios tipos de licencias [10]:

- **Unity Personal:** versión gratuita de la herramienta pero con limitaciones.
- **Unity Plus:** licencia que cuesta 35\$/mes.
- **Unity Pro:** licencia que cuesta 125\$/mes, y está más orientada a verdaderos profesionales y estudios.

Aunque hay diferentes tipos de licencias, en función de nuestros ingresos, se deberá actualizar nuestra licencia a una inmediatamente superior. Estas son las restricciones:

- Si nuestros ingresos anuales o totales superan los \$100 mil no se permitirá usar la versión gratuita, tendrá que adquirir la licencia Plus.
- Si nuestros ingresos anuales superan los \$200 mil se deberá comprar la licencia Pro, en cuyo caso no tiene ninguna restricción.

La documentación de Unity es muy completa y está disponible en bastantes idiomas, no sólo inglés. La comunidad, a diferencia de Unreal, es mucho mayor y es un gran apoyo para consultar los problemas que vayan surgiendo o hacer alguna pregunta.

El lenguaje de programación utilizado es C# o Javascript y su editor es realmente intuitivo y sencillo (ver Ilustración 3). Es un motor muy recomendable para empezar en el mundo de los videojuegos sin experiencia.

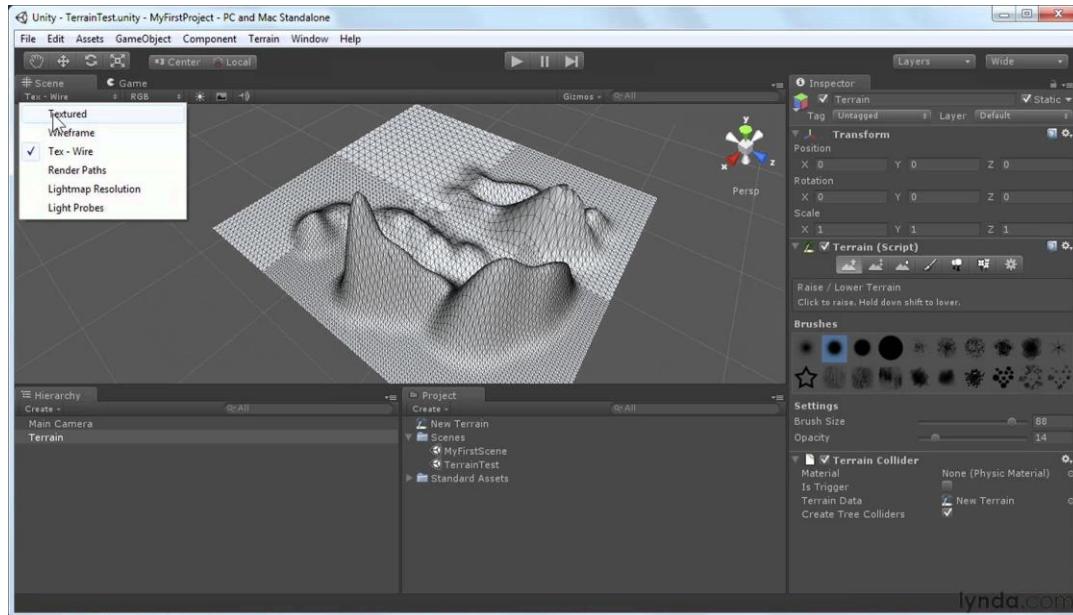


Ilustración 3. Editor de Unity

Unity tiene los componentes básicos de cualquier motor (siempre y cuando se combre la licencia) y además, tiene muchos paquetes externos que facilitan ciertas funcionalidades, como puede ser Vuforia, que te proporciona componentes de realidad aumentada. Hay otros dos paquetes interesantes de empresas externas para Unity: **Photon**, que facilita la implementación de un sistema multijugador y proporciona un sistema de salas y un servidor, y por otro lado, **PlayFab**, que se encarga de la autenticación y el almacenamiento de datos en la nube, dos componentes que van se complementan y pueden facilitar este proyecto.

Entre los miles de juegos que se han creado con este motor, se encuentra Lara Croft GO, Manifold Garden, Rust y StarCrawlers.

2.2.3. CryEngine

Por último, está **CryEngine** que se implementó para la creación del juego FarCry en 2006, y que ha ido evolucionando hasta la versión actual, CryEngine V. Este motor es mucho más restrictivo que los anteriores, y sólo es posible instalar su editor en un sistema operativo Windows.



Las plataformas [11] para las que puede dar soporte a cualquier videojuego que se desarrolle en dicho motor, queda definido en la siguiente tabla (ver Tabla 4).

Plataforma	¿Está soportado por el motor gráfico?
Windows PC	Sí
Mac OS X	No
Linux	Sí
PlayStation 4	Sí
Xbox One	Sí
Nintendo Switch	No
Nintendo 3DS	No
PlayStation Vita	No
Wii U	No
VR	Sí
SteamOS	No
HTML 5	No
Android	No
iOS	No
Windows Phone	No
Tizen	No

Android TV	No
Facebook Gameroom	No

Tabla 4. Plataformas disponibles para CryEngine

Al igual que Unreal Engine 4, es totalmente gratis y tenemos acceso a toda su funcionalidad y además, no deberemos pagar nada a la compañía (independientemente de tus beneficios), a no ser que quieras hacer una donación, o bien quieras ser miembro de la comunidad para recibir cursos [12], podrás hacer uso de la tienda para poner en venta tus paquetes de productos o comprar otros, entre otras muchas cosas.

El aprendizaje en CryEngine es mucho peor que en Unreal Engine 4. Se puede acceder a la documentación pero no hay ningún tutorial gratuito ni se puede acceder a la comunidad a menos que pagues una cuota mensual de miembro.

Este motor ha permitido la creación de juegos como: FarCry, Crysis, Aion Online o Star Citizien.

2.3. Elección del motor de videojuegos

Tras haber analizado e investigado sobre los tres motores anteriores, he llegado a la conclusión de que Unity es la mejor opción para desarrollar el proyecto. Una de las principales razones son los paquetes externos que están disponibles para este motor, ya que facilitarán el sistema online. Por otro lado, la comunidad es mucho mejor y hay una gran cantidad de tutoriales para poder desarrollar el proyecto de un modo sencillo en las 300 horas que debo invertir para desarrollar el trabajo fin de grado.

Desde el punto de vista económico Unity es mucho mejor ya que como máximo gastaremos 125\$ al mes independientemente de la cantidad de beneficios que se obtengan, a diferencia de Unreal.

Unreal puede ser una muy buena opción para grandes empresas que están desarrollando a gran escala, proporciona unos componentes de una calidad superior a cualquier motor de videojuegos disponible en la actualidad y la cantidad de plataformas que soporta es aceptable, aunque la cantidad de dinero que se debe pagar es mucho mayor (a partir de 3.000\$) y la comunidad está todavía en crecimiento.

Por último, CryEngine es la peor opción de las tres, aunque está disponible de manera gratuita, no hay modo de acceder a tutoriales ni a la comunidad de manera gratuita y el número de plataformas para los que está disponible es muy limitado.

3. Planificación del proyecto

La primera fase de un proyecto antes de comenzar es la planificación, es una de las fases más importantes porque se ha de elegir la metodología que ha de seguir el grupo de trabajo (o una única persona como es mi caso), el tiempo que disponemos para terminar el proyecto, cuánto personal necesitamos para llevarlo a cabo en ese tiempo, el presupuesto del que se dispone y otros muchos factores.

3.1. Metodología

La metodología usada será la metodología ágil SCRUM [13], la cual nos permitirá desarrollar el producto a través de un proceso iterativo e incremental. Este tipo de metodología está altamente ligada al desarrollo de videojuegos, ya que es un producto software que necesita ir desarrollando progresivamente prototipos de éste para poder depurarlo y obtener un producto de calidad. Proporciona las siguientes ventajas:

- Etapas de tiempo, denominadas “Sprint”, con una duración de entre una semana o dos en las que el equipo se marca unos objetivos en base a los requisitos del cliente. En este periodo de tiempo se subdivide el problema actual asociando a cada uno de los componentes del grupo una tarea.
- El producto es testeado⁷ al final de todas estas etapas, para comprobar si cumple con los requisitos del cliente, y encontrar posibles problemas o mejoras.
- El nivel de complejidad del proyecto va incrementando gradualmente.

Para llevar a cabo este tipo de metodología se deben seguir una serie de etapas de un modo cíclico, volviendo a la etapa inicial si es necesario. Podemos distinguir varias etapas básicas: analizar y definir los requisitos del cliente, realizar un diseño en

⁷ Pruebas que se realizan en el software para comprobar su calidad.

base a esos requisitos, implementación del diseño, generación de un prototipo y testeo de dicho prototipo, dando lugar esta etapa a un análisis de los resultados obtenidos, y proporcionar una retroalimentación para volver a la etapa inicial si fuera necesario (ver Ilustración 4).

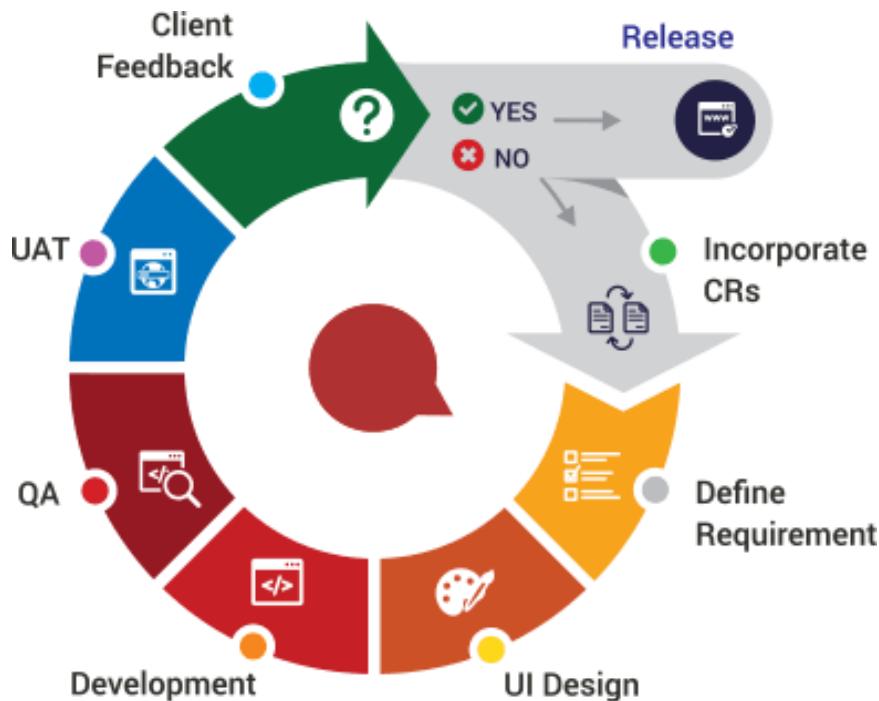


Ilustración 4. Proceso de una metodología ágil

Los grupos de trabajo suelen estar formados por equipos de entre 3 y 9 personas (rol de equipo). En SCRUM hay 3 roles perfectamente diferenciados:

- **Product owner:** encargado de comunicar continuamente al equipo las prioridades y requisitos del cliente.
- **Scrum master:** actúa de intermediario entre el product owner y el equipo.
- **Team:** equipos de entre 3-9 personas que se encargan de completar los objetivos marcados por cada sprint.

En mi caso, al ser un proyecto fin de carrera, el producto es únicamente desarrollado por mí. Por tanto, es innecesario incluirme en un tipo de rol. Los sprints serán periodos de una semana, y en cada uno de ellos me marcaré unos determinados objetivos, que en caso de no completarlos esa semana volverán a aparecer en el siguiente sprint. Además, los objetivos tendrán una prioridad, ya que al estar yo solo deberé completar los objetivos en un orden, dejando los menos prioritarios sin

completar en caso de falta de tiempo. Esta última característica ha sido un añadido incluido por mí para facilitar el modo y el orden de ejecución de tareas y sigue el siguiente esquema de tablas en cada iteración (ver Tabla 5).

Tipo de prioridad de la tarea	Color que la representa
Prioridad alta	Rojo
Prioridad media	Amarillo
Prioridad baja	Verde

Tabla 5. Tipos de tareas

Por otro lado, se deben marcar una serie de directivas o premisas que se deberán cumplir en todo el proceso de desarrollo del proyecto:

- El prototipo final tiene que ser tal y como el cliente quiere.
- El producto debe ser de la mejor calidad.
- Realizar un diseño reusable y un código desacoplado permitiendo en un futuro cambiar componentes modificando lo menos posible del sistema.
- Usar técnicas y patrones de diseño que se ajusten a nuestro problema.

3.2. Cronología

El tiempo en el que se debe desarrollar el proyecto son 300 horas en total distribuidas. En mi caso, a lo largo de poco menos de un año, desde el mes de septiembre de 2017 al mes de junio de 2018, de esas 300 horas hay que descontar 5 horas pertenecientes a las reuniones que se establecen con el tutor, por tanto dispongo de un total de 295 horas totales de trabajo autónomo.

En ese tiempo podemos dividir el proyecto en tres grandes etapas:

- Etapa de revisión de los motores gráficos y diseño del proyecto en general casos de uso, prototipo de la interfaz, definición de las mecánicas con exactitud...
- Etapa de implementación en la que se codifica lo que se ha definido con anterioridad, pero sin entrar en detalle en cuanto a efectos o cualquier diseño gráfico.

- Etapa de diseño gráfico en la que se añaden los efectos, se modelan los personajes u objetos. En definitiva, todo lo relacionado con el aspecto visual.

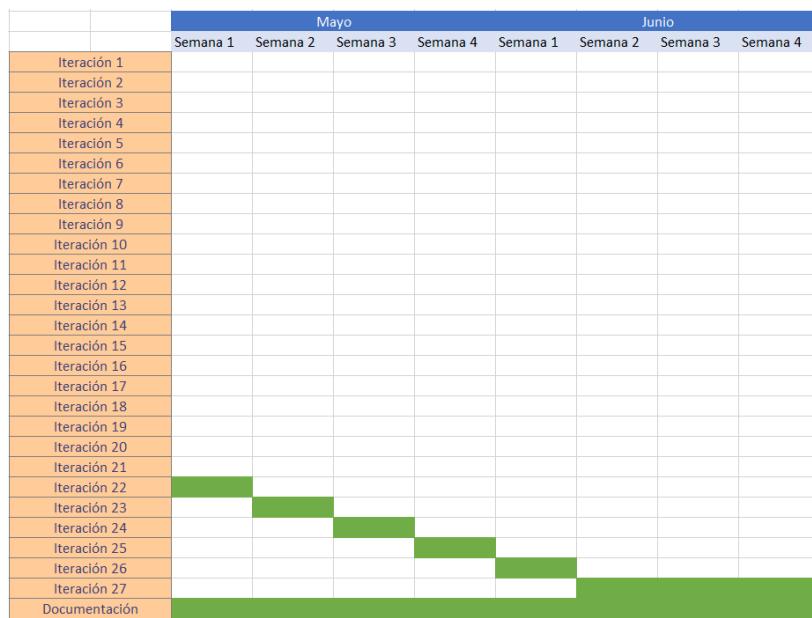
Toda y cada una de estas etapas están divididas en sprints semanales, y que a su vez, están subdivididos en tareas más concretas con un orden de prioridad. Las siguientes tablas muestran el número de total de iteraciones y como se han distribuido desde septiembre de 2017 a Junio de 2018.

Periodo de Septiembre de 2017 a Diciembre de 2017

	Septiembre				Octubre				Noviembre				Diciembre			
	Semana 1	Semana 2	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4
Iteración 1																
Iteración 2																
Iteración 3																
Iteración 4																
Iteración 5																
Iteración 6																
Iteración 7																
Iteración 8																
Iteración 9																
Iteración 10																
Iteración 11																
Iteración 12																
Iteración 13																
Iteración 14																
Iteración 15																
Iteración 16																
Iteración 17																
Iteración 18																
Iteración 19																
Iteración 20																
Iteración 21																
Iteración 22																
Iteración 23																
Iteración 24																
Iteración 25																
Iteración 26																
Iteración 27																
Documentación																

Periodo de Enero de 2018 a Abril de 2018

	Enero				Febrero				Marzo				Abril			
	Semana 1	Semana 2	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4
Iteración 1																
Iteración 2																
Iteración 3																
Iteración 4																
Iteración 5																
Iteración 6																
Iteración 7																
Iteración 8																
Iteración 9																
Iteración 10																
Iteración 11																
Iteración 12																
Iteración 13																
Iteración 14																
Iteración 15																
Iteración 16																
Iteración 17																
Iteración 18																
Iteración 19																
Iteración 20																
Iteración 21																
Iteración 22																
Iteración 23																
Iteración 24																
Iteración 25																
Iteración 26																
Iteración 27																
Documentación																

Periodo de Mayo de 2018 a Junio de 2018

Se adjuntará a esta memoria un documento por iteración donde se muestra más en detalle qué tareas se realizaron en cada iteración o semana.

3.3. Estimación de costes

Otra parte del proyecto es el presupuesto del que se dispone y en base a eso decidir qué se va a necesitar y cuánto es lo que se está dispuesto a pagar por dicho recurso. En mi caso no dispongo de ningún presupuesto, pero sí es necesario hacer un listado de lo que es necesario para el proyecto y lo que estoy dispuesto a pagar por él.

Podemos dividir los costes en cinco tipos:

- Costes de materiales.
- Costes de software.
- Costes de servicios.
- Costes de hardware.
- Costes de personal.

Todos y cada uno de los costes incluidos a continuación tienen aplicados el 21% del impuesto sobre el valor añadido.

En primer lugar, están los costes de índole material, son recursos normalmente de uso general y coste reducido y en los cuales no va incluido el hardware con el que

desarrolla el proyecto. Los productos materiales que se presentan a continuación así como su coste se han sacado de Amazon⁸ España (ver Tabla 6).

Material	Finalidad	Coste estimado
Post-It de colores rojo, amarillo y verde 47.6 mm x 47.6 mm	Se utilizarán para señalar las tareas que se realizarán cada semana	6,00€
5 Libretas Campus A5 (150x210) Grapada mela 48 hojas 90gr	Realización de diseños en papel	5,32€
Pilot BL-G2-7 – Bolígrafo, colorazul (12 unidades)	Realización de diseños en papel	19,20€
Faber-Castell 580096 – Blister de 3 lápices Grip B, goma y afilalápices	Realización de diseños en papel	8,35€
Total		38,87€

Tabla 6. Estimación de costes de materiales

En segundo lugar, están los costes del tipo software donde únicamente y exclusivamente está reflejado el software de terceros del cual será necesario hacer uso. Los períodos de pago mensuales de los productos software para desarrollar el prototipo, serán desde el mes de enero al mes de junio de 2018, es decir, un total de 6 meses. El paquete office se pagará desde el mes de septiembre de 2017 al mes de junio de 2018, haciendo un total de 10 meses (ver Tabla 7).

Software	Finalidad	Coste estimado
Unity licencia Pro	Desarrollo del prototipo de videojuego	106,00€/mes x 6
Blender	Diseño de modelos, importación y exportación de modelos y fracturas	0€

⁸ <http://www.amazon.es>

Adobe Photoshop CC	Realización de texturas y elementos de la interfaz	24,19€/mes x 6
MeshLab	Simplificación de modelos complejos	0€
Audacity	Edición de sonido	0€
Open Broadcaster Software – OBS	Grabación de la pantalla para realización de videos	0€
Licencia de Windows 10	Sistema Operativo del ordenador en el cual se realiza el prototipo	50,82€ por puesto (sólo 1)
Office 365 personal	Realización de la documentación	7,00€/mes x 10
Doxygen	Generación de la documentación asociada al código (formato HTML)	0€
Visual Studio 2017 versión estudiante	Compilación y generación de código	0€
Total		901.96€

Tabla 7. Estimación de costes de software

En tercer lugar, están los costes asociados a servicios externos, como puede ser acceso a internet (ver Tabla 8).

Servicio	Finalidad	Coste estimado
ONO- Vodafone. Fibra simétrica de 50 MB y llamadas ilimitadas ⁹	Acceso a internet para el puesto de trabajo	24,00€
Contratación del servicio Photon ¹⁰	Permitirá testear el servicio Photon que está disponible para Unity	80,77€ (proporciona una licencia de 60 meses)
Total		104,77€

Tabla 8. Estimación de costes de servicios

⁹ El coste de la tarifa se real y ha sido consultado en el mes de Junio de 2018

¹⁰ El coste de la licencia de Photon fue consultada en Junio de 2018

En penúltimo lugar, están los costes hardware en cuanto a equipos necesarios u otros dispositivos. Al ser una única persona trabajando en el proyecto, sólo será necesario la compra de un ordenador. Tantos los componentes del ordenador como su coste, están sacados de una factura real realizada en 2017 en la tienda APP Informática de Jaén (ver Tabla 9).

Hardware	Finalidad	Coste estimado
Intel core i7.7700 3.60 4.20GHz LGA1151 KABY	Procesador	350,80€
Gigabyte B250M Gaming 3 1151 DDR4 HDMI M2M	Placa base	111,20€
Seagate HDD 3.5 1TB 7200rpm 64MB Sata3 Barra	Disco duro de carácter general	47,50€
Crucial DDR4 16GB 2400MHz Ballistix Sport Lt	Memoria RAM	129,00€
Nox Semitorre Hummer ZS S.Fuente USB3.0 negr	Carcasa torre	47,40€
Scythe Refrig. Katana 4 Intel AMD	Ventilación	32,10€
Asus Turbo GTX 1070 8G 8GB GDDR5 PCIE3.0 HDMI	Tarjeta gráfica	522,00€
Asus Grabadora 24X Sata Energy DR W24F1ST	Grabadora	14,80€

Sandisk plus disco SSD 240 GB interno	Disco duro SSD para el Sistema Operativo y el software que queremos que se ejecute de manera más veloz para el proyecto	86,00€
Nox FA 650W ATX 12V 2.2 PFC activo 12cm	Fuente de alimentación	49,50€
Hacer KA240H monitor led24 1920 x 1080 full	Monitor	138,20€
Pack Mars Gaming	Ratón y teclado	15,95€
Montaje y testeo de equipo	Profesional que se encarga del montaje	39,50€
Total		1.583,95€

Tabla 9. Estimación de costes de hardware

Por último, está la estimación de costes de personal en relación a los contratos de las personas que necesitaremos en el proyecto. En este caso, sólo ha habido una persona trabajando en el proyecto pero con atribuciones de desarrollador de videojuegos y diseñador de niveles, texturas y demás aspectos visuales. Dado que el tiempo trabajado ha sido un total de 295 horas, el contrato se definirá en dinero por horas. Para ello, he consultado distintos tipos de ofertas de trabajo con atribuciones similares en páginas como InfoJobs [14] o Domestika [15], y el sueldo bruto mensual para trabajos a tiempo completo suele rondar los 1.400-3.000€. Eso quiere decir que, si mensualmente trabaja 140 horas (7 horas diarias) y suponemos que el sueldo medio es de unos 2.200€ brutos mensuales, el precio por hora será de 15,70€ (ver Tabla 10).

Tipo de contrato	Finalidad	Coste estimado
Desarrollador de videojuegos	Diseño de mecánicas e implementación a nivel de código del videojuego	15,70€/hora x 295 (solo 1 persona)
Diseñador gráfico	Diseño de niveles y aspectos visuales	15,70€/hora x 295 (solo 1 persona)

Total	9.420€
--------------	--------

Tabla 10. Estimación de costes de personal

Si juntamos todos los costes asociados a cada uno de los tipos de gasto, tendremos el coste total del proyecto que viene reflejado en la siguiente tabla (ver Tabla 11).

Tipo de coste	Coste
Materiales	38,87€
Software	901,96€
Servicios	104,77€
Hardware	1.583,95€
Personal	9.420€
Total	12.049,55€

Tabla 11. Estimación de costes total

4. Diseño del juego

Una vez planificado el proyecto la primera fase de éste es el diseño. Es importante obtener los requisitos de la aplicación y en base a estos requisitos realizar un diseño que se ajuste al problema. En esta fase es importante definir las componentes del proyecto con exactitud para poder implementar el prototipo posteriormente.

4.1. Diagrama de casos de uso. Requisitos del sistema

Antes de comenzar con el diseño, se deben definir los requisitos de la aplicación, que en este caso es un videojuego. Los requisitos son aquellos que el cliente quiere que la aplicación tenga, en este caso no hay ningún cliente. Por tanto, yo definiré qué podrá hacer y qué no.

Dado que es un videojuego RPG, voy a definir como requisitos los componentes fundamentales de cualquier videojuego de este tipo de género, así como uno de los objetivos de este trabajo fin de grado: la implantación de un sistema multijugador con niveles con ciertas componentes procedurales. Los requisitos son:

- Dado que será un videojuego multijugador, sería recomendable tener guardados datos relacionados con cada jugador y sus personajes en la nube y por tanto, necesario un sistema de autenticación.
- Cada jugador podrá crear (eliminar) un determinado número de personajes para seguir la historia de un modo u otro.
- Dentro del juego podrá acceder a una serie de hechizos y elegir cuáles quiere usar y cuáles no, y por supuesto lanzarlos.
- Dentro del juego podrá recoger armadura o armas y equipárselas o tirarlas.
- Dentro del juego podrá acceder a los diferentes niveles.
- Dentro del juego podrá vender y comprar objetos.
- Dentro del juego podrá interactuar con NPCs¹¹ que le proporcionarán información sobre la historia de Valhalla.

Para esta fase, he utilizado para definir las acciones del videojuego el diagrama de caso de uso [16] (capítulo 8). Se ha dividido en dos diagramas diferentes, uno para las acciones disponibles en el menú principal y otro para las disponibles dentro del juego. Como actor principal, está única y exclusivamente el jugador. También se podría añadir otro actor principal, como puede ser el de administrador, encargado de gestionar cuentas y posibles abusos dentro del juego, pero se ha visto innecesario. Por tanto, todos los usuarios que tengan acceso a la aplicación tendrán el mismo rol.

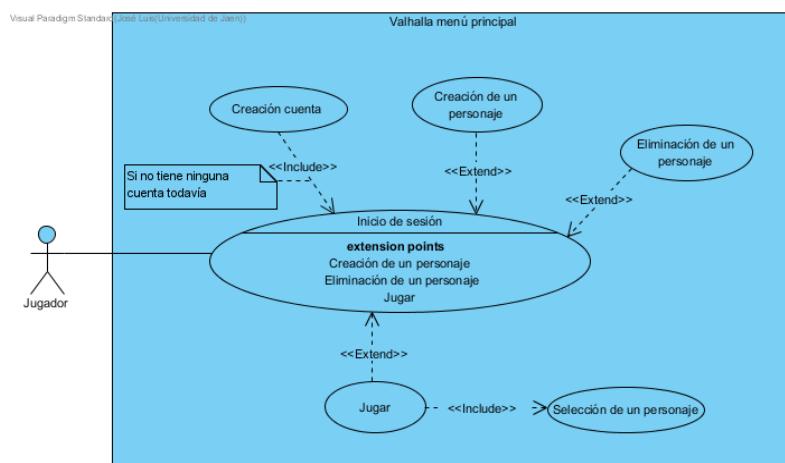
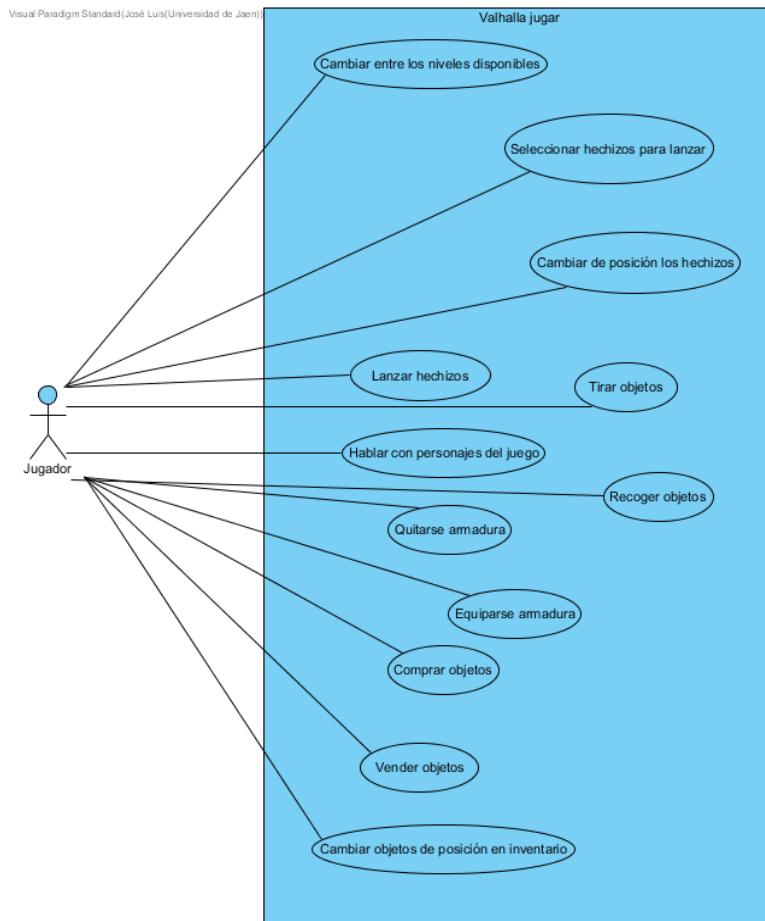


Ilustración 5. Caso de uso para el menú principal

¹¹ NPC: Non-Player Character

**Ilustración 6.** Caso de uso jugar

Para el diagrama de caso de uso del menú principal (ver Ilustración 5), se ha contemplado una única acción principal, el inicio de sesión. Para poder iniciar sesión es obligatorio haber creado una cuenta con anterioridad, la cual almacenará información sensible relativa al jugador. Por otro lado, una vez el usuario ha iniciado sesión se le permite crear un personaje, eliminarlo y jugar siempre y cuando ya haya seleccionado uno de los personajes creados.

El segundo diagrama de caso de uso es para la parte del juego (ver Ilustración 6), donde el jugador interacciona con el videojuego una vez ha iniciado sesión y ha cargado la información relacionada con el personaje seleccionado con anterioridad. Las acciones son bastante sencillas y se han obtenido de los requisitos. Se pueden dividir en tres grandes grupos: manipulación de hechizos, manipulación de objetos e interacción con NPCs.

4.2. Diseño de las mecánicas

Antes de comenzar con el diseño de clases, prototipos de interfaz..., es necesario definir el núcleo del videojuego, las mecánicas. Como se definió en la introducción, las mecánicas son la base y le proporcionan una entidad que lo define dentro de un género, en este caso RPG.

A continuación se explica cada una de las mecánicas por separado y más en detalle, justificando su uso dentro del juego y definiéndolas con exactitud para que en la fase de implementación solo sea necesario codificarlo.

4.2.1. Mecánica de niveles

La mecánica de niveles proporciona un progreso para uno de los personajes. Estos niveles permiten una mejora en su equipo y hechizos, y desbloquean nuevas características del videojuego. Cada personaje cuenta con un número entero que define su nivel y cada nivel tendrá unos puntos de experiencia concretos que te permiten subir al siguiente nivel cuando se han alcanzado dichos puntos o se han superado. Este tipo de mecánica es muy común en la mayoría de juegos, no solo los del tipo RPG.

Ventajas que proporciona subir de nivel:

- Por cada nivel, el personaje aumentará sus atributos principales o secundarios o ambos. Cada vez que el personaje consiga subir de nivel incrementa el valor de sus atributos en una cantidad fija e independiente del nivel (la cantidad en la que se ven incrementados dependerá de su rol o clase), así se puede personalizar cada una de las clases para equilibrarlas¹² debidamente.
- En ciertos niveles el sistema desbloquea habilidades en el libro de hechizos y si el personaje alcanza el nivel máximo consigue desbloquear todas las habilidades.
- A nivel máximo se desbloquea el modo carrera.
- El nivel del personaje limitará el máximo valor que un equipo puede proporcionarle a un personaje al ser equipado.

¹² Uno de los problemas principales de la mayoría de videojuegos es que los diferentes roles o clases no están compensados y hace imposible o no viable el uso de algunos de éstos

Para poder subir de nivel es necesario ganar experiencia a través de:

- Derrotando enemigos. Cada enemigo proporcionará unos puntos de experiencia al ser derrotado.
- Completando los objetivos de la historia.

El nivel de experiencia que se necesita para cada nivel queda definido por la siguiente función (ver Ecuación 1), donde la variable x representa el nivel siendo $1 \leq x \leq 30$, e y es la experiencia necesaria para subir al siguiente nivel:

$$y = \left(\frac{x}{0.1}\right)^2$$

Ecuación 1. Función para calcular la experiencia necesaria en cada nivel

En un comienzo, cuando se crea el personaje, tiene nivel 1, y el nivel máximo de éste es nivel 30. Haciendo uso de la función anterior se puede calcular los puntos de experiencia necesarios para subir en cada nivel (ver Tabla 12).

Nivel	Puntos de experiencia necesarios
1	100
2	400
3	900
4	1600
5	2500
6	3600
7	4900
8	6400
9	8100
10	10000
11	12100
12	14400
13	16900
14	19600
15	22500
16	25600
17	28900
18	32400
19	36100

20	40000
21	44100
22	48400
23	52900
24	57600
25	62500
26	67600
27	72900
28	78400
29	84100
30	90000

Tabla 12. Tabla de niveles con los puntos de experiencia necesarios para subir al siguiente nivel

La experiencia necesaria para subir desde nivel 1 a nivel 30 viene definida por la siguiente ecuación (ver Ecuación 2), donde x son los puntos de experiencia totales para subir un personaje hasta el nivel máximo.

$$x = \sum_{x=1}^{30} \left(\frac{x}{0.1}\right)^2 = 945500$$

Ecuación 2. Función sumatoria para calcular la experiencia necesaria para subir al nivel máximo

**Ilustración 7.** Gráfico que representa la función

Esta función garantiza un incremento en la experiencia necesaria en cada nivel, aumentando la dificultad para subir conforme se va acercando al nivel máximo. Otro factor complementario y que también facilitará o perjudicará la facilidad en la subida de nivel, son los puntos de experiencia que proporcionarán los enemigos y misiones. Estos dos factores deben estar en equilibrio para proporcionar la complejidad adecuada y deseada.

4.2.2. Mecánica libro de hechizos

Es un paradigma muy utilizado en videojuegos en los que el jugador puede elegir entre una gran variedad de hechizos, pero sólo puede elegir unos cuantos. Para ello, se suelen presentar los hechizos en un libro desde el cual se pueden arrastrar los hechizos disponibles para poder usarlos, o bien mostrar información de éstos: qué hacen, qué daño realizan, bajo qué condiciones, etc.

Un claro ejemplo de este tipo de mecánica es cualquier videojuego del género RPG. Como en casos anteriores veces se usará como ejemplo World of Warcraft (ver Ilustración 8).



Ilustración 8. Libro de hechizos de World of Warcraft

4.2.3. Mecánicas de clase o rol

Los personajes que controlarán los jugadores estarán divididos en diferentes tipos de clases o roles, que a su vez estarán definidas por unos atributos¹³ principales y secundarios, así como una serie de habilidades que varían en función de la clase definida.

Este tipo de mecánica es el modo de progresión dentro del juego, el cual te permite identificarte con uno de los roles y progresar como un héroe o un villano. Este recurso es necesario en los tipos de videojuego RPG, y un ejemplo de ello es Diablo 3 (ver Ilustración 9).

Cada clase dispondrá de al menos 12 habilidades, y sólo podrá usar 6 habilidades simultáneamente y además, tendrá una serie de pasivas que de igual modo que los hechizos sólo podrán tener una pasiva¹⁴ activa a la vez.



Ilustración 9. Clases de diablo 3. De izquierda a derecha: guerrero, cazador de demonios, monje, médico brujo y mago

4.2.3.1. Atributos comunes a todas las clases

A continuación, se hará un listado de los atributos que serán comunes a todas las clases o roles, o incluso a los enemigos. Se han hecho dos divisiones: atributos principales y atributos secundarios.

Atributos principales: son los atributos más importantes y que por tanto, todas las clases tienen. Muchos de estos atributos suelen ser comunes en la mayoría de videojuegos, por ejemplo en World of Warcraft [17]. Los atributos principales son los siguientes:

¹³ Atributo: calidad o característica propia de una persona o cosa, especialmente algo que lo caracteriza y es esencial en su naturaleza

¹⁴ Pasiva: beneficio que recibe de manera permanente o bajo las condiciones establecidas pero sin necesidad de que el jugador las active

- Vida: atributo que debe gestionar el jugador para que su personaje no muera, si su vida llega a 0 muere.
- Recurso: atributo necesario para poder usar las habilidades del personaje.
- Armadura: reduce el daño recibido.
- Daño: indica los puntos de daño que es capaz de infligir en un ataque. Las habilidades usarán estos puntos como base para calcular el daño que provocan en función porcentajes.
- Probabilidad de crítico: indica la probabilidad que tiene el personaje de que el siguiente ataque sea crítico.

Atributos secundarios: son atributos que una clase puede tener o no, en función de cómo se quiera orientar. Los atributos secundarios son los siguientes:

- Regeneración de vida: es el encargado de regenerar al usuario una cantidad de vida cada cierto tiempo.
- Regeneración de recurso: atributo que regenera el recurso del que dispone el personaje, y al igual que la vida, también se regenera cada cierto tiempo.
- Velocidad de movimiento: indica la velocidad del personaje.

Estos atributos pueden aplicarse también a los compañeros¹⁵: mascotas, otros guerreros invocados, etc. También pueden tener algún atributo más que esté específicamente definido para ese compañero.

4.2.3.2. Definiendo en detalle los atributos

Vida: es uno de los atributos más sencillos e importantes del juego. Cada personaje contará con unos puntos de vida, si el personaje llega a 0 puntos de vida, éste morirá. El valor de la vida nunca puede llegar a ser negativo, y tendrá un valor máximo en función del nivel u otros factores que modifiquen este límite.

¹⁵ Compañero: inteligencia artificial que acompaña al personaje de algún modo, de manera permanente o durante un tiempo determinado, y ataca los enemigos o ayuda al jugador de alguna forma

Recurso: es otro de los atributos esenciales. A diferencia de la vida, si el recurso llega a 0, el jugador no muere pero no podrá ejecutar habilidades. Por tanto, es un recurso que debe gestionar el usuario con cautela. Con lo cual, se puede englobar este atributo, así como la vida, en una mecánica de economía. Por otro lado, al igual que la vida, nunca será negativo, aunque sí un valor máximo, pero con la diferencia de que este máximo puede ser modificado al subir de nivel o no.

Armadura: cada punto de armadura reducirá un 0.05% el daño recibido, sin hacer distinciones (en principio) en tipos de daño, y como máximo podrá reducirse un 30% el daño recibido. Esto quiere decir que los puntos necesarios para alcanzar este máximo son: $\frac{30}{0.05} = 600$ puntos de armadura. Se puede superar el umbral de los 600 puntos, pero el porcentaje no pasa de ese máximo.

Daño: el cuarto atributo principal, y no menos importante, es el atributo que le proporcionará daño al usuario, y se añadirá como daño base a los ataques básicos y en una proporción a las habilidades a modo de porcentaje.

Probabilidad de crítico: como último atributo principal está la probabilidad de crítico. Este atributo será el encargado de incrementar el daño de otro modo. El valor de éste estará en un intervalo de [0, 100] %: con 0 nunca habrá un ataque crítico, y con 100 siempre será crítico.

¿Qué es un ataque crítico?: es un ataque que ve aumentado su daño normal, y ocurre con más o menos frecuencia en función de la probabilidad que tenga asociada el personaje, en este caso el daño se verá triplicado.

Regeneración de vida: se encarga de regenerar una cantidad de vida en un tiempo determinado. Este atributo está dividido en dos atributos secundarios:

- Cantidad de vida: puntos de vida que regenera. No puede ser negativo, y puede ser modificado por el nivel o por otros factores como el equipo.
- Tiempo: cada cierto tiempo regenera la cantidad de vida que se ha especificado. No puede ser negativo, está definido en segundos y será un valor fijo que varía en función de la clase.

Se regenerará vida siempre y cuando el personaje no tenga la vida máxima y no esté muerto.

Regeneración de recurso: se encarga de regenerar una cantidad del recurso del personaje en un tiempo determinado. Este atributo está dividido en dos atributos secundarios al igual que la vida:

- Cantidad de recurso: puntos de recurso que regenera. No puede ser negativo, y puede ser alterado por el nivel o por otros modificadores, aunque no es obligatorio que lo modifiquen. Por lo tanto puede ser un valor fijo.
- Tiempo: cada cuanto tiempo regenera la cantidad de recurso que se ha especificado. No puede ser negativo, está definido en segundos y será un valor fijo que viene definido por la clase.

Se regenerará recurso siempre y cuando el personaje no llegue al máximo valor de recurso y no esté muerto.

Velocidad de movimiento: La velocidad de movimiento indica las unidades por segundo que es capaz de moverse el personaje. Todas las clases tendrán una velocidad de movimiento base que será de 1.5 metros/segundo.

4.2.4. Mecánica de clase o rol: Cazador

Es una clase a distancia, se distingue por su sigilo y velocidad debido a que no tiene mucha armadura, asesta grandes golpes a distancia con su ballesta o su arco, y suele ir acompañado de su fiel lobo Sköll.

El compañero del cazador tendrá el mismo nivel que posea el cazador y además, los mismos atributos pero con diferentes valore. Por cada nivel que suba también verá aumentados los valores de éstos.

4.2.4.1. Valores iniciales de los atributos

Definición de los atributos del cazador a nivel inicial¹⁶:

- **Vida:** 125 puntos de vida.
- **Recurso:** dispondrá de un recurso llamado energía cuyo rango de valores estará en el intervalo [0, 100].
- **Daño:** 40 puntos de ataque.

¹⁶ Nivel con el comienza un personaje cuando es creado, en el caso de Valhalla nivel 1

- **Armadura:** 30 puntos de armadura.
- **Probabilidad de crítico:** 0 %.
- **Regeneración de vida:** 5 puntos de vida cada 5 segundos.
- **Regeneración de energía:** 10 puntos de energía cada 2 segundos.
- **Velocidad de movimiento:** la velocidad básica será de un 125%.

Definición de los atributos del lobo Sköll a nivel inicial:

- **Vida:** 75 puntos de vida.
- **Recurso:** ninguno.
- **Daño:** 15 puntos de ataque.
- **Armadura:** 310 puntos de armadura.
- **Probabilidad de crítico:** 10 %.
- **Regeneración de vida:** 15 puntos de vida cada 5 segundos.
- **Velocidad de movimiento:** la velocidad básica será de un 185%.
- **Velocidad de ataque¹⁷:** 1 ataque por segundo.
- **Rango de ataque¹⁸:** 0.5 metros de radio.
- **Tiempo de resurrección¹⁹:** 10 segundos.

4.2.4.2. Incremento de los atributos al subir de nivel

Cada vez que el cazador suba de nivel aumentará de manera constante sus atributos del siguiente modo:

- **Vida:** aumenta 25 puntos de vida.
- **Recurso energía:** no aumenta.
- **Daño:** aumenta 10 puntos de ataque.
- **Armadura:** no aumenta.
- **Probabilidad de crítico:** no aumenta.

¹⁷ Atributo específico de los compañeros que indica cuántos ataques básicos por segundo realiza

¹⁸ Distancia máxima en la que es efectiva una habilidad o un ataque básico

¹⁹ Tiempo que tarda en volver a reaparecer una vez haya muerto

- **Regeneración de vida:** aumenta 10 puntos de vida, y el tiempo se queda igual.
- **Regeneración de energía:** no aumenta.
- **Velocidad de movimiento:** no aumenta.

En el caso del lobo, aumentará sus atributos del siguiente modo:

- **Vida:** aumenta 10 puntos de vida.
- **Daño:** aumenta 5 puntos de ataque.
- **Armadura:** aumenta 10 puntos de armadura.
- **Probabilidad de crítico:** aumenta un 1%.
- **Regeneración de vida:** aumenta 5 puntos de vida, y el tiempo se queda igual.
- **Velocidad de ataque:** no aumenta.
- **Rango de ataque:** no aumenta.
- **Tiempo de resurrección:** no disminuye.

4.2.4.3. Habilidades de clase

Por último, se define las habilidades asociadas al cazador. El compañero no dispone de ninguna habilidad, pero podrá ser manipulado a través de habilidades de clase. El daño de los hechizos o habilidades de cada una de las clases vienen dadas a modo de porcentajes, ejemplo: si posee 45 puntos de daño base y el hechizo realiza un 120% de daño, los puntos de vida que quita son: $1.2 * 45 = 54$ puntos de vida.

Cada habilidad que necesite ser explicada por su complejidad, incluirá una imagen explicativa de su definición. Inicialmente fueron bocetos hechos en papel y pasados a ordenador.

Ataque básico:

- Definición: lanza un proyectil en línea recta y que únicamente afecta a un enemigo (ver Ilustración 10).
- Daño: 100% del daño.
- Rango de ataque: 10 metros.

- Coste de energía: ninguno.
- Tiempo de reutilización²⁰: ninguno.
- Nivel requerido²¹: 1.

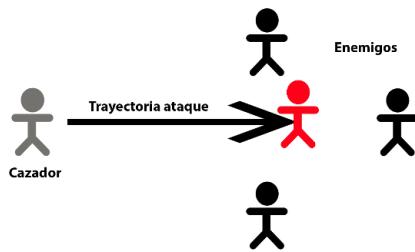


Ilustración 10. Ataque básico

Lanzamiento explosivo:

- Definición: lanza un proyectil en línea recta que sólo afecta a un enemigo y cuando colisiona con éste genera una explosión que afecta a los enemigos cercanos (ver Ilustración 11).
- Daño: 120% de daño.
- Rango de ataque: 10 metros.
- Daño de la explosión: 50% del daño.
- Radio explosión: 2.5 metros.
- Coste de energía: 15 puntos de energía.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 5.

²⁰ Tiempo necesario que hay que esperar entre dos lanzamientos consecutivos de una misma habilidad

²¹ Nivel necesario para desbloquearla

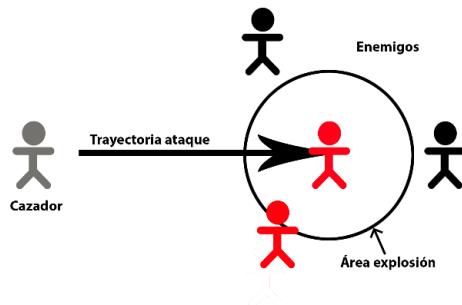


Ilustración 11. Ataque explosivo

Lanzamiento potenciado:

- Definición: lanza un proyectil en línea recta que solo afecta a un enemigo, y aumenta el daño que recibe durante un tiempo (ver Ilustración 10).
- Daño: 150% de daño.
- Rango de ataque: 10 metros.
- Coste de energía: ninguno.
- Aumento de daño: aumenta un 10% el daño recibido de todas las fuentes al enemigo golpeado durante 5 segundos.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 2.

Marcado para morir:

- Definición: marca a un enemigo para que Sköll lo ataque, en ese momento Sköll deja de hacer lo que estaba haciendo y carga contra el enemigo marcado.
- Daño: aumenta el daño base de Sköll un 20%.
- Rango de ataque: 10 metros.
- Coste de energía: 10 puntos de energía.

- Duración: 3 segundos.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 7.

Multidisparo:

- Definición: realiza un lanzamiento de proyectil múltiple que golpea a todos los enemigos situados dentro del cono que define el ataque (ver Ilustración 12).
- Daño: 150% de daño.
- Rango de ataque: 10 metros.
- Ángulo de apertura: 45 grados.
- Coste de energía: 20 de energía.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 10.

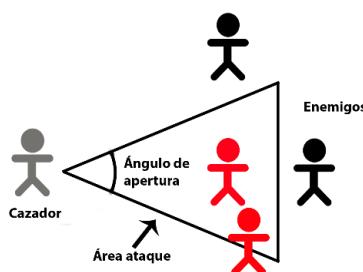


Ilustración 12. Multidisparo

Lanzamiento envenenado:

- Descripción: lanza una botella que impacta contra el suelo o contra un enemigo u otro objeto, y explota envenenando a todos los enemigos, produciendo un daño en el tiempo y aumentando todo el daño recibido durante ese periodo (ver Ilustración 13).

- Daño: 20% de daño cada segundo.
- Aumento de daño: las unidades afectadas reciben un 5% más del daño recibido.
- Duración: 10 segundos.
- Rango de ataque: 10 metros.
- Radio de explosión: 5 metros.
- Coste de energía: ninguno.
- Tiempo de reutilización: 10 segundos.
- Nivel requerido: 13.

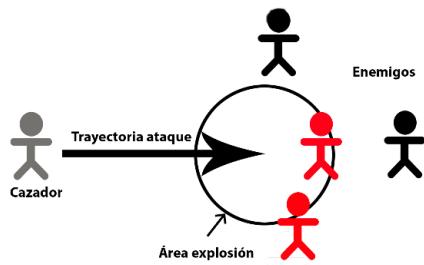


Ilustración 13. Lanzamiento envenenado

Trampa congelante:

- Descripción: el cazador coloca una trampa congelante a sus pies que cuando es activada ralentiza²² en un área a todos los enemigos durante un tiempo (ver Ilustración 14).
- Daño: ninguno.
- Radio de activación: 1 metro.
- Radio de ralentización: 5 metros.

²² Reducción de la velocidad de movimiento

- Ralentización: reducen un 50% la velocidad de movimiento a los enemigos que están dentro del área de ralentización.
- Coste de energía: 10 de energía.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 15.

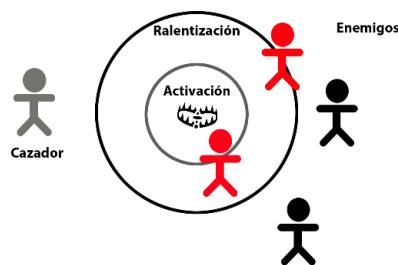


Ilustración 14. Trampa congelante

Salto enérgico:

- Descripción: el cazador salta en una dirección determinada una cantidad de metros (ver Ilustración 15).
- Daño: ninguno.
- Distancia de salto: 15 metros.
- Coste de energía: 20 puntos de energía.
- Tiempo de reutilización: 5 segundos.
- Nivel requerido: 17.

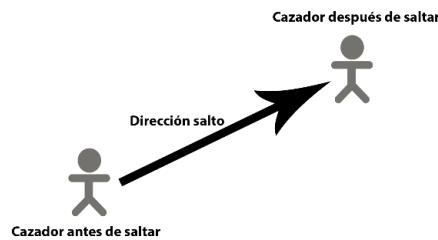


Ilustración 15. Salto enérgico

Rugido salvaje:

- Descripción: Sköll ruge asustando a todos los enemigos en un radio determinado alrededor suyo, reduciendo su daño durante un tiempo (ver Ilustración 16).
- Daño: ninguno.
- Radio de acción: 5 metros.
- Coste de energía: 20 puntos de energía.
- Tiempo de reutilización: 5 segundos.
- Nivel requerido: 20.

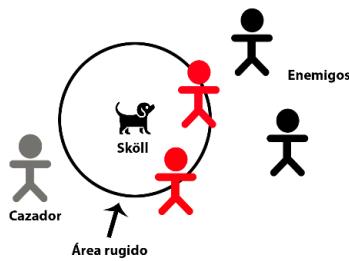


Ilustración 16. Rugido salvaje

Regeneración salvaje:

- Descripción: Sköll se cura un porcentaje de su vida cada cierto tiempo.
- Daño: ninguno.
- Regeneración de vida: se cura un 10% de su vida cada segundo durante 5 segundos.
- Coste de energía: ninguno.
- Tiempo de reutilización: 15 segundos.
- Nivel requerido: 24.

Inyección de adrenalina:

- Descripción: el cazador se inyecta una sustancia que le permite regenerar energía y aumenta su daño durante un periodo de tiempo.
- Daño: ninguno.
- Aumento de daño: aumenta el daño que realiza un 10% durante 5 segundos.
- Coste de energía: ninguno.
- Regeneración de energía: recupera toda la energía faltante.
- Tiempo de reutilización: 20 segundos.
- Nivel requerido: 27.

Cólera salvaje:

- Descripción: Sköll entra en cólera aumentando su ataque, su velocidad de ataque y reduce el daño que recibe. Si estaba muerto ²³revive.
- Daño: ninguno.
- Duración de la habilidad: 10 segundos.
- Aumento de daño: aumenta el daño que realiza un 120%.
- Aumento velocidad de ataque: aumenta un 300% la velocidad de ataque.
- Disminución del daño recibido: reduce un 100% el daño recibido.
- Coste de energía: 50 puntos de energía.
- Tiempo de reutilización: 60 segundos.
- Nivel requerido: 30.

4.2.4.4. Pasivas de clase

Las pasivas de las que dispone son las siguientes:

- **Ataques potenciados:** aumenta el daño del cazador un 20% a los enemigos que estén a 8 metros o más. Nivel requerido: 10.
- **Entrenador de bestias:** aumenta la vida de Sköll un 50% y su daño un 25%. Nivel requerido: 20.
- **Equilibrio natural:** aumenta la probabilidad de crítico del cazador un 15% y de Sköll un 10%. Nivel requerido: 30.

4.2.5. Mecánica de clase o rol: Guerrero vikingo

Es una clase que pelea cuerpo a cuerpo y posee una gran resistencia. Además, al ser fuerte es capaz de llevar cualquier armadura y siempre va armado con sus hachas o espadas, e incluso un escudo.

4.2.5.1. Valores iniciales de los atributos

Definición de los atributos del cazador a nivel inicial:

- **Vida:** 250 puntos de vida.

²³ Volver a la vida

- **Recurso:** dispone de un recurso llamado hamingja, la suerte o fortuna concedida por los Dioses. Tendrá a nivel inicial 200 puntos de hamingja.
- **Daño:** 75 puntos de ataque.
- **Armadura:** 75 puntos de armadura.
- **Probabilidad de crítico:** 0 %.
- **Regeneración de vida:** 10 puntos de vida cada 5 segundos.
- **Regeneración de hamingja:** 30 puntos de hamingja cada 3 segundos.
- **Velocidad de movimiento:** la velocidad básica será de un 105%.

4.2.5.2. Incremento de los atributos al subir de nivel

Cada vez que el guerrero vikingo suba de nivel aumentará de manera constante sus atributos del siguiente modo:

- **Vida:** aumenta 50 puntos de vida.
- **Recurso hamingja:** 25 puntos de hamingja.
- **Daño:** aumenta 15 puntos de ataque.
- **Armadura:** no aumenta.
- **Probabilidad de crítico:** no aumenta.
- **Regeneración de vida:** aumenta 20 puntos de vida y el tiempo se queda igual.
- **Regeneración de hamingja:** 10 puntos de hamingja y el tiempo se queda igual.
- **Velocidad de movimiento:** no aumenta.

4.2.5.3. Habilidades de clase

Al igual que el cazador tiene una serie de habilidades o hechizos y son las que vienen a continuación.

Ataque básico:

- Definición: golpea al enemigo más cercano en la dirección en la que apunta su arma (ver Ilustración 17).

- Daño: 100% del daño.
- Rango de ataque: 0.5 metros.
- Coste de hamingja: ninguno.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 1.

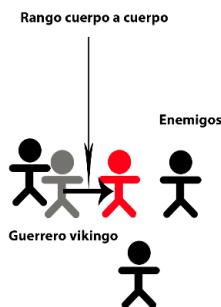


Ilustración 17. Ataque básico

Rezo a Odín:

- Definición: imbuye el arma con el poder de Odín proporcionándole un aumento del daño en los ataques que realice.
- Daño: aumenta en un 200% el daño de cualquier fuente mientras esté activo.
- Duración: 10 minutos.
- Coste de hamingja: 5 puntos de hamingja.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 1.

Rezo a Thor:

- Definición: imbuye el arma con el poder de Thor proporcionándole daño en área, desencadenando otro ataque secundario en la misma dirección que el ataque lanzado (ver Ilustración 18).

- Ataque en área: por cada ataque hará otro más en área mientras esté activo, y está definido a su vez por los siguientes atributos:
 - Rango del ataque: 4 metros.
 - Ángulo de apertura: 45 grados.
 - Daño: el daño de ese ataque secundario en área supone un 50% del daño de la habilidad que encadena el ataque en área.
- Duración: 10 minutos.
- Coste de hamingja: 5 puntos de hamingja.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 7.

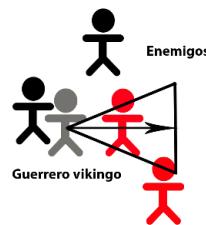


Ilustración 18. Ataque secundario del rezo de Thor

Rezo a Freya:

- Definición: imbuye el arma con el poder de Freya proporcionándole curación por cada enemigo golpeado y un aumento de la regeneración de vida y de hamingja.
- Daño: ninguno.
- Curación: se cura un 3% de la vida por cada enemigo golpeado mientras esté activo.

- Regeneración de vida: aumenta un 50% la regeneración de vida mientras está activo.
- Regeneración de hamingja: aumenta un 50% la regeneración de hamingja mientras está activo.
- Duración: 10 minutos.
- Coste de hamibgja: 5 puntos de hamingja.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 16.

Tajo profundo:

- Definición: asesta un golpe tremendo con un arma a un único enemigo en la dirección que apunta, dejándolo sangrando y aumentando todo el daño que recibe (ver Ilustración 17).
- Daño: 250% de daño.
- Duración sangrado: 5 segundos.
- Daño del sangrado: 10% de daño cada segundo.
- Aumento del daño: el objetivo afectado sufre un 10% más de daño de todas las fuentes durante la duración del sangrado.
- Rango de ataque: 0.5 metros.
- Coste de hamingja: 15 puntos de hamingja.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 5.

Torbellino:

- Definición: los dioses imbuyen sus armas con fuego de la fragua de Valhalla y el guerrero gira a su alrededor golpeando a todos los enemigos cercanos (ver Ilustración 19).
- Daño: 200% de daño.
- Rango de ataque: 0.5 metros.

- Coste de hamingja: 5 puntos de energía de hamingja cada segundo que está activa la habilidad.
- Tiempo de reutilización: ninguno.
- Nivel requerido: 10.



Ilustración 19. Ataque torbellino

Grito ensordecedor:

- Definición: grita y provoca terror en los enemigos cercanos reduciendo su daño y su velocidad de movimiento (ver Ilustración 20).
- Daño: ninguno.
- Duración: los enemigos se ven afectados durante 10 segundos.
- Reducción de daño: reduce un 25% el daño de los enemigos.
- Reducción de velocidad de movimiento: reduce un 30% la velocidad de movimiento de los enemigos afectados.
- Radio de ataque: 5 metros.
- Coste de hamingja: ninguno.
- Tiempo de reutilización: 15 segundos.
- Nivel requerido: 12.

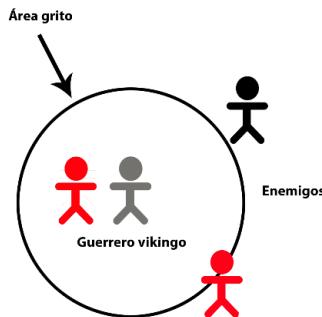


Ilustración 20. Grito ensordecedor

Grito potenciador:

- Definición: grita, potenciando sus ataques y aumentando su velocidad de movimiento.
- Duración: 10 segundos.
- Daño aumentado: aumenta todo el daño del personaje un 15% durante ese periodo de tiempo.
- Incremento de velocidad de movimiento: aumenta un 15% la velocidad de movimiento durante ese periodo de tiempo.
- Coste de hamingja: ninguno.
- Tiempo de reutilización: 25 segundos.
- Nivel requerido: 20.

Resistencia de Jötunheim:

- Descripción: pide ayuda al rey gigante de Jötunheim para que le preste su resistencia durante un breve periodo de tiempo.
- Duración: 10 segundos.
- Reducción de daño: reduce todo el daño recibido un 50% durante ese tiempo.

- Otros efectos: se vuelve imparable²⁴ durante ese tiempo y además, por cada golpe que recibe, reduce en 1 segundo el tiempo de reutilización.
- Coste de hamingja: ninguno.
- Tiempo de reutilización: 30 segundos.
- Nivel requerido: 23

Lanzamiento heroico:

- Definición: lanza un hacha en una dirección atravesando a todos los enemigos hasta un rango máximo. Esta habilidad puede acumular hasta un número de cargas²⁵ máximo (ver Ilustración 21).
- Daño: 300% de daño.
- Número de cargas máximo: 3 cargas.
- Coste de hamingja: 10 puntos de hamingja.
- Tiempo de reutilización: 3 segundos.
- Nivel requerido: 25.

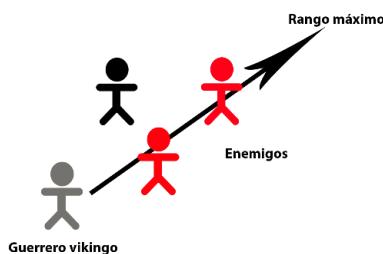


Ilustración 21. Lanzamiento heroico

²⁴ No puede ser ralentizado ni sometido a otro efecto de pérdida de control

²⁵ La habilidad tiene un contador con el número de lanzamiento consecutivos sin tiempo de reutilización

¡A las armas!

- Definición: invoca a 5 guerreros del Valhalla para que luchen por él.
- Atributos de los guerreros:
 - Daño: cada guerrero tiene un 50% del daño del guerrero vikingo.
 - Velocidad de ataque: 2 ataques por segundo.
 - Velocidad de movimiento: 100% de la velocidad básica.
 - Vida: no tienen, por tanto no pueden morir.
- Duración: 20 segundos.
- Coste de hamingja: ninguno.
- Tiempo de reutilización: 60 segundos.
- Nivel requerido: 26.

Terremoto:

- Definición: provoca un seísmo con su fuerza en la posición en la que se encuentra, ralentizando a los enemigos e infligiéndoles daño (ver Ilustración 22).
- Daño: provoca un 500% de daño cuando es activada.
- Duración: 10 segundos.
- Daño en el tiempo: después del primer golpe el terremoto provoca 100% del daño del guerrero vikingo cada segundo.
- Radio de acción: 10 metros.
- Coste de hamingja: ninguno.
- Tiempo de reutilización: 1 minuto y 15 segundos.
- Nivel requerido: 30.

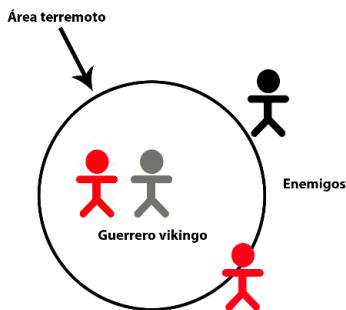


Ilustración 22. Terremoto

4.2.5.4. Pasivas de clase

Las pasivas de las que dispone son las siguientes:

- **Regeneración iracunda:** por cada enemigo asesinado, el guerrero vikingo recupera un 10% de hamingja. Nivel requerido: 10.
- **Gran defensor:** si el guerrero tiene equipado un escudo, reduce el daño recibido un 40% de manera permanente. Nivel requerido: 20.
- **Maestro de batalla:** aumenta el daño un 20% y la probabilidad de crítico aumenta un 15% de manera permanente, siempre y cuando sólo lleve armas equipadas. Nivel requerido: 30.

4.2.6. Mecánicas de compañeros

Ciertos tipos de clases y enemigos podrán acompañarlos a su lado. Éstos estarán caracterizados por unos atributos y también estarán definidos por la duración de su invocación y por último, si pueden ser eliminados o no.

4.2.6.1. Atributos del compañero

Los atributos del compañero están definidos en las mecánicas de clase. Éstos tienen los mismos tipos de atributos que cualquier otra clase, o incluso, nuevos atributos más específicos. Además, pueden tener sus propios valores para cada

atributo o bien, definir sus valores en función de algún atributo de la clase o enemigo que lo invoca²⁶.

4.2.6.2. Duración de la invocación

En función del tiempo que permanecen en el juego, se pueden hacer dos tipos de distinciones:

- **Compañeros permanentes:** son aquellos que no tienen un tiempo de desaparición, solo desaparecen si mueren.
- **Compañeros temporales:** son invocaciones asociadas a ciertas habilidades, o incluso pasivas, que permanecen durante un determinado periodo de tiempo, o bien hasta que mueren.

4.2.6.3. Tipo de combate

Por último, se pueden diferenciar en el modo de combatirlos. Hay dos tipos:

- **No pueden ser atacados:** no tienen el atributo vida y por tanto, no pueden ser atacados ni morir. Sólo desaparecen si se acaba el tiempo de invocación o desaparece el invocador.
- **Pueden ser atacados:** tendrán el atributo vida y podrán ser atacados. Si mueren y son del tipo permanente, revivirán pasado un tiempo, como por ejemplo, en el caso del lobo del cazador.

4.2.7. Mecánicas de objetos equipables

Cada personaje podrá tener equipado²⁷ una serie de objetos de manera simultánea, y cada uno de estos objetos se identificará con una parte del cuerpo y le aportará una mejora en ciertos atributos. Los atributos que aportan cambiarán de una clase a otra. Por ejemplo, en el caso del recurso para el cazador, debe aportar energía y para el guerrero vikingo hamingja. Estas partes no modificarán el aspecto de los personajes, y se podrán conseguir mediante un mercader o tendrán una probabilidad de soltarlos los enemigos al ser derrotados, o también mediante la apertura de cofres.

Este tipo de mecánica es también fundamental en cualquier videojuego RPG, ya que permite un progreso en el que el personaje es cada vez más fuerte y el jugador

²⁶ Llamar a algo o alguien, en especial a un poder superior, para pedir auxilio o protección

²⁷ Proveer a alguien de las cosas necesarias para su uso particular, especialmente de ropa o en este caso armadura

puede tomar decisiones en algunos aspectos. Por ejemplo, si quiere llevar dos hachas y ser mucho más ofensivo, o llevar un hacha y un escudo y ser más defensivo. El rol de la clase y la toma de este tipo de decisiones suele ser determinante para el rol secundario dentro de un grupo. Estos roles secundarios suelen ser:

- **Tanque:** personaje que por lo general recibe todo el daño y permite a los demás jugadores realizar daño sin morir.
- **Atacante:** también conocido coloquialmente en este contexto como 'DPS'. Son los encargados de realizar todo el daño posible.
- **Sanador:** encargados de sanar el daño recibido de todo el grupo.
- También hay otros tipos de roles como el de apoyo o mitigador, o incluso se hacen subdivisiones en el rol de atacantes. Esto varía en función del videojuego y las posibilidades que ofrezca.

Las partes que se pueden equipar se dividen en dos grandes grupos: equipables comunes a todas las clases, y los equipables que varían en función del tipo de clase.



Ilustración 23. Objetos equipados en el videojuego Diablo 3. A la izquierda aparece información acerca de los atributos

4.2.7.1. Atributos

Todas y cada una de las partes proporcionarán una mejora en los atributos del personaje que los lleve equipados. Para ello, usarán atributos propios que equivalen

a atributos de clase, de este modo facilitamos la visión general de éstos. Los atributos son (ver Tabla 13):

- **Vitalidad:** aumenta el atributo vida del personaje. Cada punto de vitalidad equivale a 500 puntos de vida y 50 puntos de regeneración de vida.
- **Fuerza:** aumenta el atributo ataque del personaje. Cada punto de potencia equivale a 50 puntos de ataque.
- **Inteligencia:** aumenta el atributo recurso del personaje. Cada punto de vitalidad equivale a 250 puntos de recurso y 25 puntos de regeneración de recurso. No todas las clases hacen uso de este atributo debido a que su valor máximo es constante, por ejemplo en el caso del cazador.
- **Índice de golpe crítico:** aumenta el atributo de probabilidad de crítico del personaje. Cada punto de índice de golpe crítico equivale a un 1% de probabilidad de crítico.
- **Resistencia:** aumenta el atributo armadura del personaje. Cada punto de armadura equivale a 15 puntos de armadura.
- **Agilidad:** aumenta el atributo velocidad de movimiento del personaje. Cada punto de agilidad equivale a 0.25% de velocidad de movimiento.

Atributo	Equivalencia
Vitalidad	500 puntos de vida y 50 de regeneración
Fuerza	50 puntos de ataque
Inteligencia	250 puntos de recurso y 25 puntos de regeneración
Índice de golpe crítico	1% de golpe crítico
Resistencia	15 puntos de armadura
Agilidad	0.25% velocidad de movimiento

Tabla 13. Tabla de equivalencias

4.2.7.2. Equipables comunes

Los equipables comunes serán iguales para todas las clases. Sólo difieren en los atributos que pueden subir al ser equipados, pero podrán obtenerlos todas las clases y sólo podrán tener equipado un objeto del mismo tipo a la vez.

Los objetos equipables son:

- **Casco**: parte que se coloca en la cabeza.
- **Hombreras**: parte que se coloca en los hombros.
- **Pecho**: parte que se coloca en el torso.
- **Guantes**: parte que se coloca en las manos.
- **Pantalones**: parte que se coloca en las piernas.
- **Botas**: parte que se coloca en los pies.

4.2.7.3. Equipables no comunes

Los equipables no comunes varían en función de las clases. Por tanto se deben definir a nivel de clase.

Para la clase cazador se tendrán los siguientes equipables, y sólo se podrá equipar uno de los siguientes, a menos que se diga lo contrario:

- **Arco**: arma que lanza como proyectiles flechas.
- **Ballesta de dos manos**: arma que lanza como proyectiles flechas o bodoques²⁸.
- **Ballesta de una mano**: pueden equiparse hasta dos al mismo tiempo y son armas que lanzan lo mismo que la ballesta de dos manos.

Para la clase guerrero vikingo, se tendrán los siguientes equipables y se podrán equipar dos objetos cualesquiera, siempre y cuando no estén repetidos:

- **Espada de una mano**: arma con la que puede golpear cuerpo a cuerpo.
- **Hacha de una mano**: arma con la que puede golpear cuerpo a cuerpo.
- **Escudo**: arma defensiva para protegerse.

4.2.7.4. Asignación de recursos

Cada parte aportará una serie de atributos en concreto, y la asignación a cada parte quedará definida por la siguiente tabla (ver Tabla 14).

²⁸ Bola de barro endurecida

Atributo	Partes
Vitalidad	Casco, hombreras, pecho, guantes, pantalones, botas, escudo
Inteligencia	Casco, hombreras, pecho, guantes, pantalones, botas, escudo
Fuerza	Arco, ballesta de dos manos, ballesta de una mano, hacha de una mano, espada de una mano
Índice de golpe crítico	Arco, ballesta de dos manos, ballesta de una mano, hacha de una mano, espada de una mano, guantes
Resistencia	Pecho, pantalones, escudo
Agilidad	Botas

Tabla 14. Tabla con los atributos que puede tener cada pieza

Cada uno de los atributos tienen como intervalo de valores [1, nivel actual] (ambos inclusive), es decir, si por ejemplo tenemos nivel 23 y somos un cazador, si un enemigo tira un casco podrá tener hasta un máximo de 23 puntos en el atributo de vitalidad, y no tendrá el atributo inteligencia, ya que esta clase no varía su recurso (a no ser que sea por habilidades). En este caso, el valor del atributo vitalidad podrá variar entre [1, 23], y la manera en que se asignará vendrá definido en las mecánicas de suerte.

Como caso excepcional, en el escudo, su intervalo de valores en el atributo resistencia es de [1, nivel actual + 20] (ambos inclusive), y las armas de dos manos tendrán como intervalo en los atributos [1, nivel actual * 2].

4.2.8. Mecánicas de suerte

Son mecánicas que utilizan el factor aleatorio para evitar comportamientos predecibles. Las mecánicas de suerte estarán presentes de muy diversas maneras en el videojuego.

4.2.8.1. Factor suerte en los objetos equipables

Los objetos equipables son afectados por el factor suerte en dos sentidos: en primer lugar, en la probabilidad de que un enemigo suelte un objeto, y en segundo lugar, una vez lo haya soltado que valores tendrán los atributos de éste.

La probabilidad de los objetos que dejan caer los enemigos, vendrá definida por una tabla de objetos que pueden soltar, y cada uno de ellos tendrá un peso, que representará una probabilidad. Veámoslo con un ejemplo: suponemos que un enemigo tiene la siguiente tabla (ver Tabla 15).

Objeto	Peso
Nada	70
Casco	20
Arma de tu clase	10

Tabla 15. Ejemplo de tabla de pesos para un enemigo

En este caso, la suma de pesos totales es de: $70 + 20 + 10 = 100$. Por tanto la probabilidad de que no tire nada sería de: $\frac{70}{100} * 100 = 70\%$ de probabilidad. ¿Cuál será el procedimiento para saber que tirará el monstruo? Muy sencillo, los pasos son los siguientes:

1. Se tira un número aleatorio entre [1, suma de pesos total] (ambos incluidos).
2. Comprobamos en que rango de los objetos se encuentra.

El rango de cada objeto vendrá definido por (valor inicial, valor final], y éstos quedarán definidos como:

- **Valor inicial del rango:** suma de los pesos anteriores. En caso de no haber ningún intervalo anterior, el valor será el inicial, es decir 0.
- **Valor final del rango:** al valor inicial le sumamos el peso de ese objeto.

Con lo cual, retomando el ejemplo anterior, el rango de valores para esa tabla sería de:

- **Rango para nada:** (0, 70] que también sería equivalente a [1, 70].
- **Rango para casco:** (70, 90] que también será equivalente a [71, 90].

- **Rango para el arma de clase:** (90, 100] que también será equivalente a [91, 100].

Vamos a suponer que como resultado de una tirada, el enemigo debe tirar una pieza de armadura ¿pero cuál? Para ello se usarán tablas anidadas en casos en los que pueda haber más de una opción. Por ejemplo, el caso de la **pieza de armadura**, se puede definir otra tabla de pesos como la siguiente (ver Tabla 16).

Pieza de armadura	Peso
Casco	25
Hombreras	25
Guantes	25
Pecho	25

Tabla 16. Ejemplo de tabla de pesos para una pieza de armadura

Para este caso, se seguirá el mismo procedimiento volviendo a hacer una nueva tirada y siguiendo el mismo método de selección.

Por otro lado, está el factor suerte asociado a los valores de los atributos. Recordemos que los valores de los atributos varían entre un intervalo bien definido [valor mínimo, valor máximo], y estos valores están definidos en las mecánicas de objetos equipables. Por tanto, se debe tener un modo de asignar un valor para cada atributo. A la hora de asignar un valor hay que diferenciar dos tipos de asignación (aunque muy parecidos entre ellos): asignación de valores de objetos que tiran los monstruos y cofres, o asignación de valores de objetos que obtenemos de los mercaderes.

En primer lugar, se va a definir el factor suerte de los objetos asociados a los mercaderes. Para esta asignación de valores se tendrá en cuenta qué abalorio ha usado el jugador para comprar ese objeto:

- **Objeto comprado con abalorio de bronce:** todos los atributos se calculan obteniendo un número aleatorio entre su rango de valores (del mismo modo que para los enemigos).
- **Objeto comprado con abalorio de plata:** se elige aleatoriamente uno de los atributos del objeto para que tenga al menos la mitad de su valor máximo. Para ello, se hace una tirada entre el intervalo [mitad valor

máximo, valor máximo], y para el resto de atributos se calculan igual que en el abalorio de bronce.

- **Objeto comprado con abalorio de oro:** se elige aleatoriamente uno de los atributos del objeto para que tenga el valor máximo, y el resto de atributos se calculan igual que en el abalorio de bronce.

Por otro lado, están los objetos que los enemigos y los cofres dejan caer. Éstos se calculan usando la técnica anterior y el método de la tabla de probabilidades. En primer lugar se usa la siguiente tabla para elegir qué atributos serán perfectos (ver Tabla 17).

Beneficio	Peso
Nada	95
Un atributo será perfecto ²⁹	4.75
Todos los atributos serán perfectos	0.25

Tabla 17. Tabla de probabilidades de atributos perfectos

El método será el mismo que en la tabla de loteo, se sacará un número aleatorio y en función del rango en el que esté, se hará una cosa u otra.

El siguiente paso dependerá de lo que haya salido anteriormente:

1. Si ha salido que no ocurre nada, entonces se calcula aleatoriamente los valores de los atributos, siempre en el intervalo del atributo.
2. Si ha salido que un atributo será perfecto, primero se elige aleatoriamente qué atributo será perfecto y se le pondrá como valor el máximo del intervalo, y por último, para los atributos restantes se hace el mismo procedimiento que en el caso de no haber ningún beneficio.
3. Si ha salido que todos los atributos serán perfectos, se ponen todos los atributos a su valor máximo.

4.2.8.2. Factor suerte en las habilidades de los enemigos elites

En las mecánicas de enemigos (definido más adelante), los enemigos raros pueden tener simultáneamente hasta 3 habilidades que serán seleccionadas siguiendo estos pasos para cada habilidad:

²⁹ El atributo tiene el valor máximo del intervalo

1. En primer lugar, sigue el método de la tabla de pesos y tira aleatoriamente para saber si la habilidad será pasiva o activa.
2. Una vez seleccionado el tipo anterior, se decrementa su peso en 10 en la tabla para que en la siguiente tirada sea menos probable que sea seleccionada. Si el peso llega a 0 o un número negativo, se elimina de la tabla.
3. El siguiente paso será elegir si será ofensiva o defensiva (del mismo modo que antes).
4. Una vez seleccionado el tipo anterior, se decrementa su peso en 10 en la tabla para que en la siguiente tirada sea menos probable que sea seleccionada. Si el peso llega a 0 o un número negativo se elimina de la tabla.
5. Por último, selecciona una habilidad aleatoria que cumpla las restricciones de los dos tipos elegidos anteriormente, y la habilidad que sea seleccionada (si la hay) debe ser eliminada del registro para que no vuelva a salir.
6. Volver al paso 1 hasta que se tengan 3 habilidades.

A continuación se definen las tablas que permiten dicho funcionamiento: en primer lugar, la tabla del tipo de funcionamiento (ver Tabla 18) y por último, la tabla del tipo de uso (ver Tabla 19).

Tipo de funcionamiento	Peso
Activa	20
Pasiva	20

Tabla 18. Tabla definida para el tipo de funcionamiento

Tipo de uso	Peso
Ofensiva	20
Defensiva	20

Tabla 19. Tabla definida para el tipo de uso

Para las habilidades no hace falta definir una tabla de pesos, ya que se elige aleatoriamente de entre las que hay definidas. Por ejemplo, tirando un número aleatorio entre en el intervalo [0, número total de habilidades).

4.2.9. Mecánicas de enemigos

El videojuego contará con enemigos o monstruos que deberán ser eliminados para conseguir experiencia, objetos o algún tipo de objetivo.

Hay varios tipos de enemigos, y tienen una serie de atributos comunes entre ellos que difieren en el valor de éstos. Los atributos son similares a los que tienen los roles.

4.2.9.1. Atributos comunes

Los atributos que tienen definidos los enemigos será un subconjunto de los atributos que han sido definidos para las clases de personajes. Los atributos son:

- Nivel.
- Vida.
- Armadura.
- Daño.
- Probabilidad de crítico.
- Velocidad de movimiento.

Estos atributos ya han sido explicados anteriormente y tienen el mismo significado y utilidad. El atributo nivel servirá para calcular los valores de los demás atributos en función de los valores base a excepción de los atributos velocidad de movimiento y probabilidad de crítico, que serán constantes, a no ser que alguna habilidad los modifiquen. Por ejemplo, si tenemos un enemigo con los siguientes valores base:

- Vida: 200 puntos de vida.
- Armadura: 10 puntos de armadura.
- Daño: 20 puntos de ataque.
- Probabilidad de crítico: 5%.
- Velocidad de movimiento: 1.5 metros/segundo.

Si el nivel del enemigo es 30 se multiplicarán los atributos modificables por el nivel del enemigo, y sus valores pasarán a ser:

- Vida: 6.000 puntos de vida.
- Armadura: 300 puntos de armadura.
- Daño: 600 puntos de ataque.
- Probabilidad de crítico: 5%.
- Velocidad de movimiento: 1.5 metros/segundo.

4.2.9.2. Tipos de enemigos

Hay tres tipos de enemigos:

- **Enemigos frecuentes:** son los enemigos más comunes, más débiles y frecuentes.
- **Enemigos élitres:** son enemigos del tipo frecuente, pero que son mucho más fuertes y que tienen 3 habilidades aleatorias (definido en mecánicas de suerte).
- **Enemigos finales o jefes:** son los enemigos más fuertes y únicamente habrá cuatro jefes finales, uno por cada nivel, y tendrán una serie de habilidades.

4.2.9.3. Definiendo los enemigos

A continuación vienen definidos los enemigos que se encontrarán en el videojuego. En cada uno se indicará los valores base de los atributos, el intervalo de niveles que tendrán, y el tipo que son con sus respectivas especificaciones, y por supuesto sus habilidades.

Las habilidades definidas como ataques básicos causarán un daño igual al daño que tiene definido el enemigo, es decir de un 100%. El nivel de los enemigos será el mismo nivel que el del jugador que crea la partida, a excepción de enemigos élitres o jefes finales.

Guerreros caídos: son guerreros que han caído en combate y han subido al Valhalla, y ahora han sido invocados por los dioses para luchar. Definición:

- Tipo: enemigos frecuentes/raros.
- Atributos base:
 - Vida: 200 puntos de vida.

- Armadura: 2 puntos de armadura.
- Daño: 25 puntos de ataque.
- Probabilidad de crítico: 5%.
- Velocidad de movimiento: 1.5 metros/s.
- Nivel:
 - Como enemigo frecuente, tendrá el nivel del jugador que creó la partida.
 - Como enemigo raro, tendrá 8 niveles más que como enemigo frecuente.
- Habilidades:
 - Ataques básicos con una velocidad de un 1 ataque/segundo y con un rango de 0.5 metros.

Berseker o ulfhednar: son guerreros que han sido modificados por alucinógenos, no tienen miedo a nada y son los primeros en la batalla, además no sienten el dolor y crean terror entre sus enemigos. Definición:

- Tipo: enemigos frecuentes/raros.
- Atributos base:
 - Vida: 100 puntos de vida.
 - Armadura: 10 puntos de armadura.
 - Daño: 50 puntos de ataque.
 - Probabilidad de crítico: 15%.
 - Velocidad de movimiento: 1.5 metros/s.
- Nivel:
 - Como enemigo frecuente tendrá el nivel del jugador que creó la partida.
 - Como enemigo raro tendrá 8 niveles más que como enemigo frecuente.
- Habilidades:

- Ataques básicos con una velocidad de 2 ataques/s y con un rango de ataque 0.5 metros.
- Lanza un ataque en área cada 15 segundos que hace sangrar a los jugadores y compañeros que estén en un radio de 10 metros y con un daño del 20% y una duración de 3 segundos.

Suplantador del Dios Thor: un cazador de dioses ha conseguido robarle parte de su magia al Dios del trueno Thor y suplantarle, y como consecuencia puede usar todos y cada uno de sus poderes y manejar a su hueste de guerreros del Valhalla.

Definición:

- Tipo: jefe final.
- Atributos base:
 - Vida: 2000 puntos de vida.
 - Armadura: 5 puntos de armadura.
 - Daño: 40 puntos de ataque.
 - Probabilidad de crítico: 5%.
 - Velocidad de movimiento: 1 metros/s.
- Nivel: tendrá 8 niveles más que el jugador que creó la partida.
- Habilidades:
 - Ataques básicos con una velocidad de 1 ataque/segundo y con un rango de ataque de 0.5 metros.
 - Tormenta furiosa: lanza rayos en un radio de acción definido por la posición del jefe. Definición de la habilidad:
 - Radio de acción: 10 metros.
 - Daño: cada rayo hará un 300% del daño del enemigo.
 - Tiempo de reutilización: 15 segundos.
 - Invocación de guerreros: invoca dos guerreros caídos del Valhalla. Definición de la habilidad:
 - Número de guerreros: 2.

- Tipo: guerreros caídos que pueden morir y no tienen tiempo límite de vida.
- Tiempo de reutilización: 15 segundos.

Suplantador del Dios Odín: es otro de los suplantadores, y en este caso, le ha robado la magia al Dios de dioses, Odín, y por tanto, también ha adquirido sus poderes, y al igual que los demás dioses tendrá control sobre una hueste de guerreros del Valhalla. Definición:

- Tipo: jefe final.
- Atributos base:
 - Vida: 5000 puntos de vida.
 - Armadura: 10 puntos de armadura.
 - Daño: 20 puntos de ataque.
 - Probabilidad de crítico: 10%.
 - Velocidad de movimiento: 2 metros/s.
- Nivel: tendrá 8 niveles más que el jugador que creó la partida.
- Habilidades:
 - Ataques básicos con una velocidad de 1.5 ataques/segundo y 0.5 metros de rango de ataque.
 - Cría cuervos: cada cierto tiempo invocará a un determinado número de cuervos que no podrán ser golpeados ni morir de ningún modo, sólo desaparecen si el suplantador muere.Definición de la habilidad:
 - Daño: 20% del daño de Odín.
 - Número de cuervos: 2 cuervos.
 - Tiempo de invocación: 3 segundos.
- Potenciador: recibe una bonificación de ataque cuando alcanza ciertos porcentajes en la vida. Criterios de potenciación:

- 50% la vida: cuando pierde un 50% de la vida aumenta su ataque un 200%, y en consecuencia se actualiza el daño de los cuervos.
- 75% de vida: cuando pierde un 75% de la vida aumenta su ataque otra vez en otro 200%, y en consecuencia se actualiza el daño de los cuervos.

Suplantador del Dios Loki: es otro de los suplantadores y en este caso le ha robado la magia al Dios Loki, el embaucador, y por tanto, también ha adquirido sus poderes, y al igual que los demás dioses tendrá control sobre una hueste de guerreros del Valhalla. Definición:

- Tipo: jefe final.
 - Atributos base:
 - Vida: 1500 puntos de vida.
 - Armadura: 2 puntos de armadura.
 - Daño: 50 puntos de ataque.
 - Probabilidad de crítico: 10%.
 - Velocidad de movimiento: 1.5 metros/s.
 - Nivel: tendrá 8 niveles más que el jugador que creó la partida.
 - Habilidades:
 - Ataques básicos con una velocidad de 1 ataque/segundo y 10 metros de rango de ataque.
 - Escudo de espinas: el jefe se envuelve en un escudo que devuelve el daño recibido al personaje que origina el daño.
- Definición de la habilidad:
- Duración del escudo: 5 segundos.
 - Tiempo de reutilización: 15 segundos.
 - Reflejo de daño: refleja el daño provocando un 200% del daño recibido.

- Maestro del engaño: realiza una serie de copias de sí mismo con un porcentaje de su daño y de su vida, sólo desaparecen si son eliminadas y solo tienen como habilidad el ataque básico.

Definición de habilidades:

- Número de copias: 3.
- Daño de cada copia: un 20% de su daño cada una.
- Vida de cada copia: un 10% de su vida.
- Tiempo de reutilización: 20 segundos.

Suplantadora de la Diosa Freya: es el último de los suplantadores y en este caso le ha robado la magia a la Diosa Freya y por tanto, también ha adquirido sus poderes, y al igual que los demás dioses tendrá control sobre una hueste de guerreros del Valhalla. Definición:

- Tipo: jefe final.
 - Atributos base:
 - Vida: 3000 puntos de vida.
 - Armadura: 2 puntos de armadura.
 - Daño: 40 puntos de ataque.
 - Probabilidad de crítico: 5%.
 - Velocidad de movimiento: 1.5 metros/s.
 - Nivel: tendrá 8 niveles más que el jugador que creó la partida.
 - Habilidades:
 - Ataques básicos con una velocidad de 2 ataques/segundo y con un rango de ataque de 10 metros.
 - Charco venenoso: crea un charco de veneno teniendo como centro la posición aleatoria de un jugador, este veneno produce un daño en el tiempo y ralentiza el jugador durante su duración.
- Definición de la habilidad:
- Radio del charco: 2.5 metros.

- Daño: provoca un 200% de su daño cada segundo mientras esté dentro del charco.
 - Ralentización: reduce la velocidad un 30% mientras esté dentro del charco.
 - Duración: 10 segundos.
 - Tiempo de reutilización: 12.5 segundos.
- Curación iracunda: se cura un porcentaje de su vida. Definición de la habilidad:
 - Curación: 20% de su vida.
 - Tiempo de reutilización: 20 segundos.

4.2.9.4. Habilidades que pueden tener los enemigos raros

Cada enemigo raro tendrá 3 habilidades como máximo de manera simultánea. El método de asignación de estas habilidades viene definido en las mecánicas de suerte. A continuación, se define a los tipos de habilidades que pueden tener.

Habilidades defensivas

Son aquellas habilidades que ayudan al enemigo a sobrevivir y dificulta matarlo.

Entre estas habilidades están:

- **Armadura drakar:** habilidad pasiva que reduce todo el daño un 50%.
- **Descendencia de gigantes:** habilidad pasiva que aumenta su vida un 30% de manera permanente.
- **Escudo reflector:** habilidad activa que devuelve un porcentaje del daño al causante de éste. Definición de la habilidad:
 - Duración del escudo: 5 segundos.
 - Daño reflejado: devuelve un 200% del daño recibido al causante.
 - Tiempo de reutilización: 15 segundos.
- **Inmovilización:** habilidad activa que provoca que un jugador no se mueva durante unos segundos y no pueda atacar. Definición de la habilidad:

- Duración del prejuicio: 3 segundos.
- Tiempo de reutilización: 15 segundos.

Habilidades ofensivas

Son aquellas habilidades que ayudan al enemigo a enfrentarse al jugador y dificulta la supervivencia del jugador. Entre estas habilidades están:

- **Potenciador ofensivo:** habilidad pasiva que aumenta un 30% el daño de manera permanente.
- **Asesino sin piedad:** habilidad pasiva que aumenta su probabilidad de crítico en un 30%.
- **Granadero:** habilidad activa que coloca una trampa explosiva en la posición en la que se encuentra con una duración determinada y que provoca un daño en un radio al ser activada. Definición de la habilidad:
 - Duración: 30 segundos.
 - Radio activación: 1 metro.
 - Radio explosión: 4 metros.
 - Daño: 300% del daño del enemigo.
 - Tiempo de reutilización: 5 segundos.
- **Invocación:** habilidad activa que invoca guerreros del mismo tipo que el suyo pero normales, para que desaparezcan deben morir. Definición de habilidad:
 - Número de invocaciones: invoca 2 guerreros simultáneamente.
 - Tiempo de reutilización: 10 segundos.

4.2.9.5. Tablas de probabilidad de objetos equipables

Cada enemigo tiene una tabla de probabilidad (ir a mecánicas de suerte para su explicación). En este caso, se hace una diferencia de tablas entre tipo de enemigo. A continuación, se definen las tablas.

Tabla de probabilidad para enemigos normales

La siguiente tabla de pesos es utilizada por todos los enemigos del tipo normal (ver Tabla 20).

Objeto	Peso
Nada	80
Pieza de armadura	15
Arma de tu clase	5

Tabla 20. Primer nivel de la tabla de pesos para enemigos normales

La siguiente tabla es el siguiente nivel de la tabla anterior para el caso de la pieza de armadura (ver Tabla 21).

Pieza de armadura	Peso
Casco	20
Pecho	20
Pantalones	20
Hombreras	20
Guantes	20
Botas	20

Tabla 21. Tabla anidada de pesos para la pieza de armadura en enemigos normales

La siguiente tabla es el siguiente nivel de la primera tabla para el caso del arma de clase cazador (ver Tabla 22).

Arma de clase	Peso
Ballesta	30
Arco	30
Ballesta de una mano	30

Tabla 22. Tabla anidada de pesos para las armas de la clase cazador en enemigos normales

La siguiente tabla es el siguiente nivel de la primera tabla para el caso del arma de clase guerrero vikingo (ver Tabla 23).

Arma de clase	Peso
Espada de una mano	30
Hacha de una mano	30
Escudo	30

Tabla 23. Tabla anidada de pesos para las armas de la clase guerrero vikingo en enemigos normales**Tabla de probabilidades para enemigos élite**

La siguiente tabla de pesos es utilizada por todos los enemigos del tipo élite (ver Tabla 24).

Objeto	Peso
Nada	50
Pieza de armadura	35
Arma de tu clase	15

Tabla 24. Primer nivel de la tabla de pesos para enemigos élitess

Las demás tablas anidadas son iguales que para el caso de los enemigos normales.

Tabla de probabilidades para jefes finales

La siguiente tabla de pesos es utilizada por todos los enemigos del tipo jefe final (ver Tabla 25).

Objeto	Peso
Nada	10
Pieza de armadura	30
Arma de tu clase	60

Tabla 25. Primer nivel de la tabla de pesos para los jefes finales**4.2.10. Mecánicas de economía**

Las mecánicas de economía son aquellas que permiten al usuario auto gestionar un determinado recurso, en función de la gestión que se hagan de ellas se obtendrán unos beneficios o prejuicios determinados, y limitarán las acciones futuras del jugador. Por ejemplo, el atributo vida o recurso se pueden calificar, en cierto modo, como una mecánica de economía debido a que el jugador debe administrarse el recurso de tal modo que pueda lanzar habilidades en los momentos oportunos y debe también administrarse la vida para no morir.

4.2.10.1. Abalorios para obtener reliquias

Aunque la vida y el recurso son un recurso que necesita de una gestión y por tanto, una mecánica de economía de una manera implícita, hay otro recurso más

explícito. En el juego habrá una “moneda” de cambio, que serán los abalorios. Habrá abalorios de varios tipos:

- Abalorio de bronce.
- Abalorio de plata.
- Abalorio de oro.

Cada tipo de abalorio lo soltará un tipo específico de enemigo al morir, y se utilizarán para obtener objetos en los mercaderes que habrá en la zona neutral que recompensarán en base a su valía. También se podrán obtener abalorios vendiendo equipo sobrante.

4.2.10.2. Mercaderes de la zona neutral

En la zona neutral hay tres tipos de mercaderes, un mercader para los cazadores, otro para los guerreros vikingos, y otro para vender equipo sobrante.

Váli: es el dios de los arqueros, hijo de Odín y la giganta Rind, tiene una maestría y una puntería insuperable con el arco y proporciona partes equipables por abalorios únicamente para los cazadores.

Ull: es el dios de los guerreros, es el más diestro con las espada, el hacha y el escudo, y nadie puede superarlo en duelo y proporciona partes equipables a los guerreros vikingos.

Tabla de ventas

Todas y cada una de las partes que venden los mercaderes definidos anteriormente, tendrán el coste definido en la siguiente tabla (ver Tabla 26).

Parte	Coste
Casco	150 abalorios de bronce
	50 abalorios de plata
	20 abalorios de oro
Hombreras	150 abalorios de bronce
	50 abalorios de plata
	20 abalorios de oro
Pecho	250 abalorios de bronce

	75 abalorios de plata
	25 abalorios de oro
Guantes	150 abalorios de bronce
	50 abalorios de plata
	20 abalorios de oro
Botas	150 abalorios de bronce
	50 abalorios de plata
	20 abalorios de oro
Pantalones	250 abalorios de bronce
	75 abalorios de plata
	25 abalorios de oro
Armas de una mano	200 abalorios de bronce
	60 abalorios de plata
	22 abalorios de oro
Armas de dos manos	350 abalorios de bronce
	80 abalorios de plata
	30 abalorios de oro

Tabla 26. Tabla de ventas

Mercader de la taberna

Este mercader será el responsable de comprar al jugador los objetos equipables que no le interesen, y a cambio le dará abalorios. Los abalorios que proporciona quedan definidos en la siguiente tabla (ver Tabla 27).

Tipo abalorio	Condiciones
Abalorio de bronce	Lo da por cualquier objeto
Abalorio de plata	Lo da si alguno de los valores de los atributos, es mayor o igual que la mitad del máximo valor de ese atributo
Abalorio de oro	Lo da si alguno de los atributos tiene el valor máximo

Tabla 27. Tabla del mercader

Si se cumplen las tres condiciones, dará el abalorio de mayor calidad.

4.3. Modos de juego

La mayoría de videojuegos no se basan en un único modo de juego, tienen una gran variedad y el número varía en función de la complejidad de éste.

En mi caso, constará de tres modos de juego:

- Modo historia.
- Modo Core.
- Modo carrera.

Todos estos modos de juego tienen un sistema de cámara común del tipo “*Top-down*”, similar a juegos como **Diablo 3** o **Path of Exile** (ver Ilustración 24, Ilustración 25).



Ilustración 24. Vista Top-down en el videojuego Diablo 3

También en cada uno de estos modos de juego podrán jugar entre 1 y 4 jugadores a la vez, siempre y cuando estén en el mismo punto de la historia y además, los objetos que tiren los enemigos serán distintos para cada jugador. Excepto en el modo carrera, donde los enemigos no sueltan objetos.



Ilustración 25. Vista Top-down en el videojuego Path of Exile

Otro de los elementos comunes a todos los modos de juego, es el tipo de habilidades o acciones de las que dispone el jugador. El jugador dispone de las siguientes acciones:

- Moverse por el mundo 3D a través del ratón.
- Volver al punto neutral.
- Interactuar con el entorno: puntos de control, personajes que nos den objetivos, mercaderes, etc.
- Podrá tener 6 habilidades de clase simultáneas que podrá accionar mediante ratón, teclado o una combinación de ambos.
- Acceder al menú del juego con las siguientes opciones:
 - Salir del juego.
 - Guardar partida.
 - Opciones de interfaz y teclado.

Por ahora, no es importante definir qué botón del teclado o ratón acciona cada una de las acciones descritas anteriormente. Solamente es importante asociar qué acciones son permitidas dentro de cada modo de juego.

Por último, se puede decir que tienen un carácter cooperativo PvE³⁰, o bien en solitario en caso de que no haya jugadores.

4.3.1. Modo historia

Es uno de los modos principales junto con el modo Core, donde el jugador podrá completar la historia del videojuego. El jugador se sumerge en una historia que avanza conforme complete objetivos, aunque el verdadero objetivo es progresar a nivel de rol.

En este modo, tendrá acceso a la interfaz para poder cambiar de habilidades (libro de hechizos), a la interfaz para cambiar de armadura/armas (objetos equipados) y a la interfaz del inventario.

4.3.2. Modo Core

Es el otro modo principal, es idéntico al modo historia. Salvo por la diferencia de que si tu personaje muere, no podrás proseguir y tu personaje desaparecerá, con lo cual para poder progresar no podrá morir ni una sola vez.

La interfaz es la misma que en el modo anterior, pero indicando que es un personaje de Core.

4.3.3. Modo carrera

Es un modo que se activa desde cualquiera de los otros modos anteriores. Al activarse, lleva a los jugadores a un mapa pequeño con multitud de enemigos, y deberán derrotarlos en un tiempo predeterminado para poder ganar. Si derrotan a todos los enemigos antes del límite de tiempo, entrarán a formar parte en un ranking general ordenado desde el menor tiempo hasta el mayor.

Además de las reglas de tiempo, una vez dentro no se podrá acceder al libro de hechizos y al inventario, y tampoco se podrá modificar la armadura, y aunque se acceda a este modo desde el modo core, si el personaje muere no desaparecerá.

4.4. Diseño de la base de datos

Para poder almacenar la información relativa a cada uno de los jugadores, es necesario algún tipo de base de datos. En principio, una base de datos remota para

³⁰ PvE/JcE (Player Versus Environment/Jugador contra Entorno): término muy utilizado en los videojuegos de rol para designar una modalidad en la que uno o más jugadores se enfrentan a la máquina, no entre ellos.

que nadie pueda acceder a ningún tipo de información ni modificarla, sólo tendrá acceso a través del videojuego y unas credenciales.

En primer lugar, hay que analizar qué datos hay que almacenar y cómo se deberían almacenar sin tener en cuenta qué tecnología se va a utilizar para implementarla. Por un lado, se deberían almacenar datos con información sensible del jugador, este tipo de información sería parte de una cuenta de usuario. Al estar trabajando con información sensible, es importante registrar dichos ficheros de la base de datos en la Agencia Española de Protección de Datos.

Los datos de cada **cuenta de usuario** necesarios podrían ser:

- Correo electrónico como identificador único de cada cuenta.
- Nombre de usuario que no tiene porqué ser único.
- Contraseña almacenada de manera encriptada.
- Lista de personajes.
- Algún dato más, como fecha de creación de la cuenta, o dinero virtual, podría ser información relevante a tener en cuenta para posibles prototipos futuros.

Por otro lado, se debe guardar información de cada uno de los personajes. Los campos necesarios serían:

- Nombre del personaje (único en la lista de personajes asociado a una cuenta).
- Nivel del personaje.
- Puntos de experiencia en el nivel actual.
- Número de abalorios de oro.
- Número de abalorios de plata.
- Número de calaveras de bronce.
- Lista con objetos del inventario.
- Lista con los objetos armadura que tiene equipados.
- Habilidad asociada a cada tecla.

Por último, para los objetos del inventario se deben hacer distinciones para poder almacenar cualquier tipo de objeto, armadura, paciones, etc. Aunque por ahora en este diseño sólo contempla objetos del tipo armadura, es muy recomendable realizar el diseño pensando en futuros objetos. Los datos que debe incluir **cualquier objeto** son:

- Identificador del objeto.
- Posición en el inventario.

La **armadura** es un tipo de dato más concreto, que además de incluir los datos de cualquier objeto, contiene datos específicos relacionados con el valor de cada atributo (generados en el momento de la obtención):

- Vitalidad.
- Inteligencia.
- Fuerza.
- Velocidad.
- Resistencia.
- Índice de golpe crítico.

A continuación, se plantea un posible diseño haciendo uso del modelo entidad-relación [18] de una base de datos tradicional (ver Ilustración 26). Cuenta con un total de 8 tablas: tabla para almacenar información relativa a la cuenta de un usuario, tabla para almacenar información de cada personaje, tabla con los hechizos que mantiene en uso un personaje, otra tabla para almacenar los objetos generales de un personaje y los objetos de armadura, y por último, otras 4 tablas para mantener relaciones entre ellas.

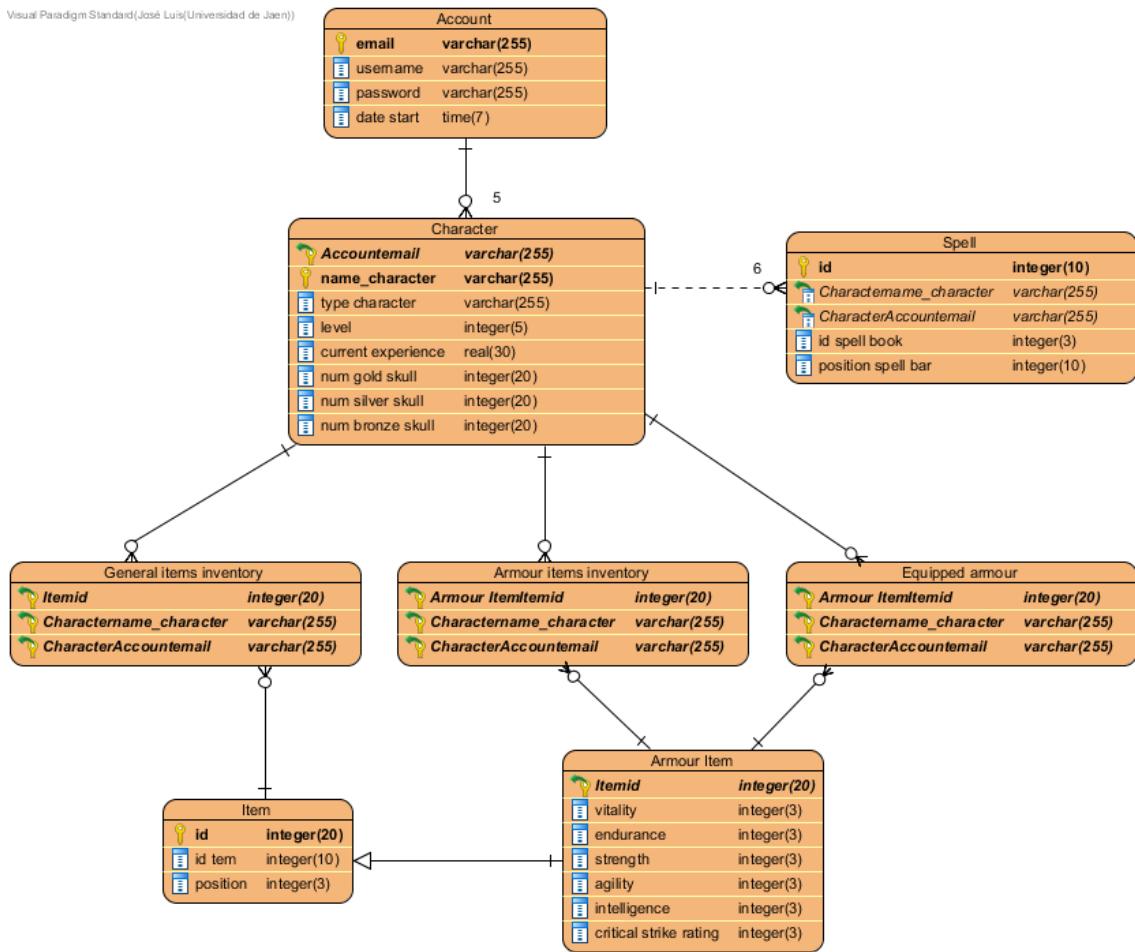


Ilustración 26. Modelo entidad-relación de la base de datos

En primer lugar, la tabla de cuentas almacena el correo, el nombre de usuario y la contraseña cifrada con algún algoritmo de cifrado, como por ejemplo, AES³¹ (también llamado Rijndael), y también se ha añadido el atributo de la fecha de creación, a modo de ejemplo, de otro tipo de dato que se puede almacenar sobre el usuario, pero que no será incluido en la versión final. La clave primaria será el mismo correo, de tal un correo sólo se podrá asociar a una cuenta.

En segundo lugar, está la tabla de personajes que contiene como clave primaria el nombre del personaje y la cuenta de correo a la que pertenece, de este modo no se podrá repetir el mismo nombre de personaje en una misma cuenta. Por otro lado, también se almacena la información mínima necesaria para reconstruir al personaje. Para ello, es necesario el tipo del personaje (deberá tener una restricción con los valores de los tipos que puede almacenar), el nivel del personaje, la experiencia

³¹ Advanced Encryption Standard

acumulada en el nivel actual, y el número de abalorios de bronce, plata y oro. El número de personajes por cuenta estará limitado a 5.

En tercer lugar, está la tabla de hechizos, encargada de almacenar que hechizos tenía asignados el jugador en un personaje en la barra de lanzamiento la última vez que jugó. Para ello, como clave primaria tiene un identificador entero auto incremental y por otro lado, está el nombre del personaje y la cuenta asociada al personaje, ambos definidos como clave foránea. De este modo, identifica a quién pertenece la información del hechizo, y por último, está el identificador del hechizo dentro del libro de hechizos del tipo de personaje y la posición que ocupa en la barra de hechizos.

En último lugar, está la tabla de los objetos almacenados en el inventario del personaje. Por un lado, está la tabla de objetos genéricos que almacena como clave primaria un identificador numérico auto incremental, el identificador del objeto en cuestión en el juego y la posición en el inventario, y por otro lado, la tabla objetos armadura, que es un subtipo de la tabla anterior que mantiene como clave primaria un identificador numérico y que además, es clave foránea de la clave primaria de la tabla objeto (no el identificador de objeto dentro del juego, por la nomenclatura de Visual Paradigm puede llevar a confusión) y también guarda todos los atributos de la armadura que fueron generados proceduralmente dentro del juego: vitalidad, resistencia, fuerza, inteligencia, agilidad e índice de golpe crítico.

El resto de tablas son el resultado de las relaciones entre un personaje y los objetos. De este modo, se puede mantener una lista de objetos que tiene almacenados en el inventario y los objetos armadura equipados en el personaje. Cabe destacar, que la suma de tuplas de objetos generales almacenados en el inventario, y por otro lado, los de objetos armadura, debe ser el número máximo de huecos del inventario. También el número de tuplas de objetos equipados, debe ser la suma total de los distintos tipos de armadura.

4.5. Prototipo de la interfaz

Para el diseño de la interfaz, se ha realizado un prototipo en Power Point utilizando el método de prototipo en papel, indicando qué consecuencias tiene para el usuario pinchar en un sitio u otro a través de números. Se adjuntará un pdf con el prototipo completo por si las ilustraciones no son suficientes.

En primer lugar, está el diseño del menú principal (ver Ilustración 27). Aquí el jugador puede crear personajes nuevos, eliminarlos, seleccionarlos y jugar, y un menú de opciones para cambiar los atajos de los hechizos o habilidades y cambiar el idioma. Se ha considerado que el usuario ya ha iniciado sesión para simplificar el diseño.



Ilustración 27. Prototipo de la interfaz. Menú principal

A continuación, está el diseño de la zona de juego. Es decir, una vez que el jugador pulsa comenzar partida. Tanto el diseño del modo historia (ver Ilustración 28)

como el diseño del modo Core (ver Ilustración 30) tienen los mismos componentes, con la diferencia de que en modo Core se indica con un elemento visual que está en dicho modo. En esta parte, el usuario tiene elementos visuales a modo informativo, como puede ser la vida, la barra de experiencia, el recurso o la barra de hechizos con el tiempo de reutilización, el mini mapa, y también elementos para interactuar.

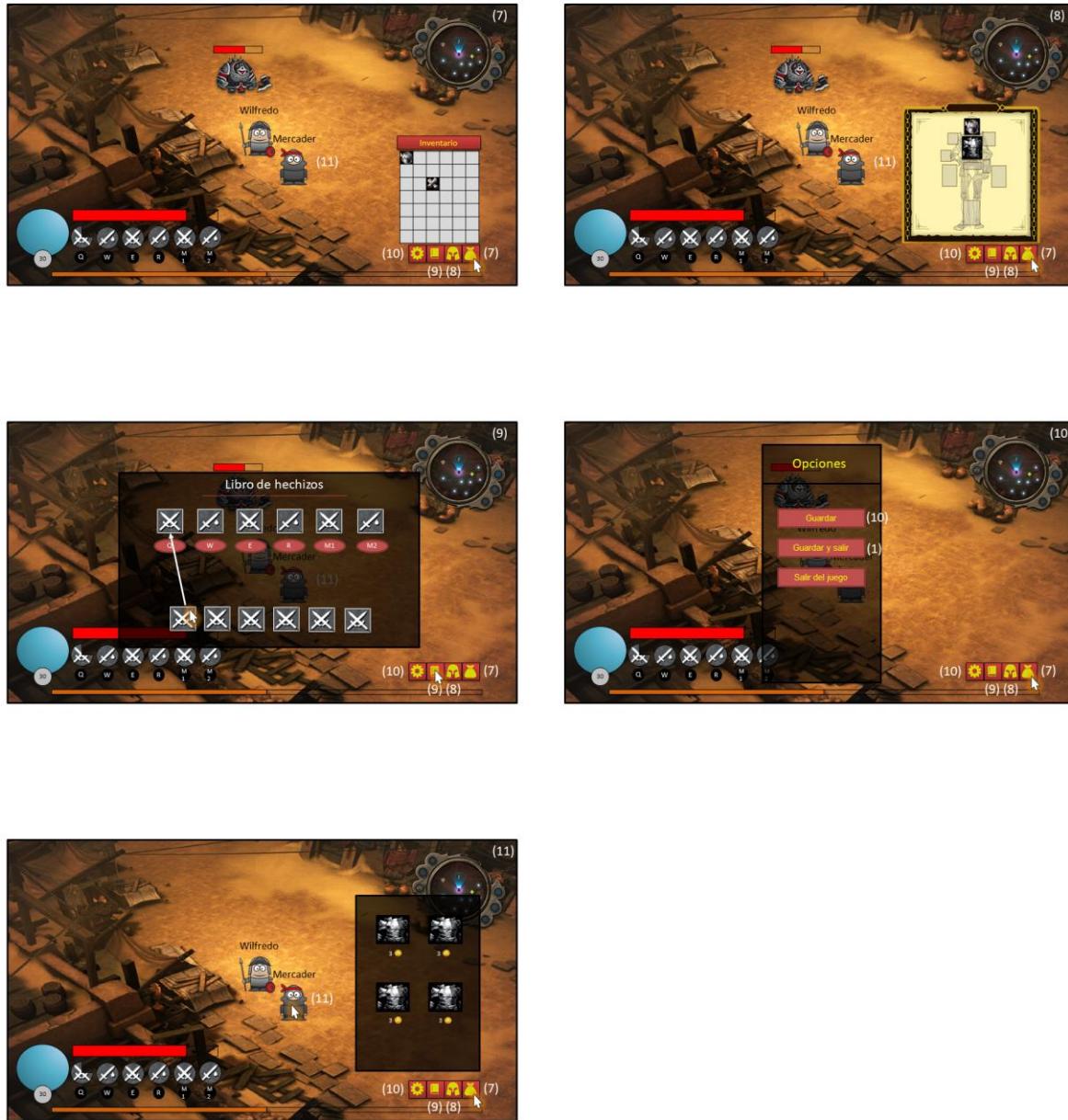


Ilustración 28. Prototipo de la interfaz. Dentro del juego en modo historia

Hay dos zonas importantes de interacción:

- **Barra inferior derecha con las opciones principales:** apertura del libro de hechizos, apertura del menú del juego y la apertura del inventario y de los objetos equipados.
- **Menú con las opciones:** guardar, guardar y salir (menú principal) y salir del juego.

Para el diseño de los elementos de la interfaz, he seguido los modelos de cualquier juego RPG (como se ha podido ver anteriormente en las mecánicas).

Por último, está el diseño del modo carrera (ver Ilustración 29) que elimina la barra inferior derecha y añade dos nuevos componentes, uno que indica el tiempo restante, y otro con un contador de los enemigos que quedan por matar.



Ilustración 29. Prototipo de la interfaz. Dentro del juego en modo carrera

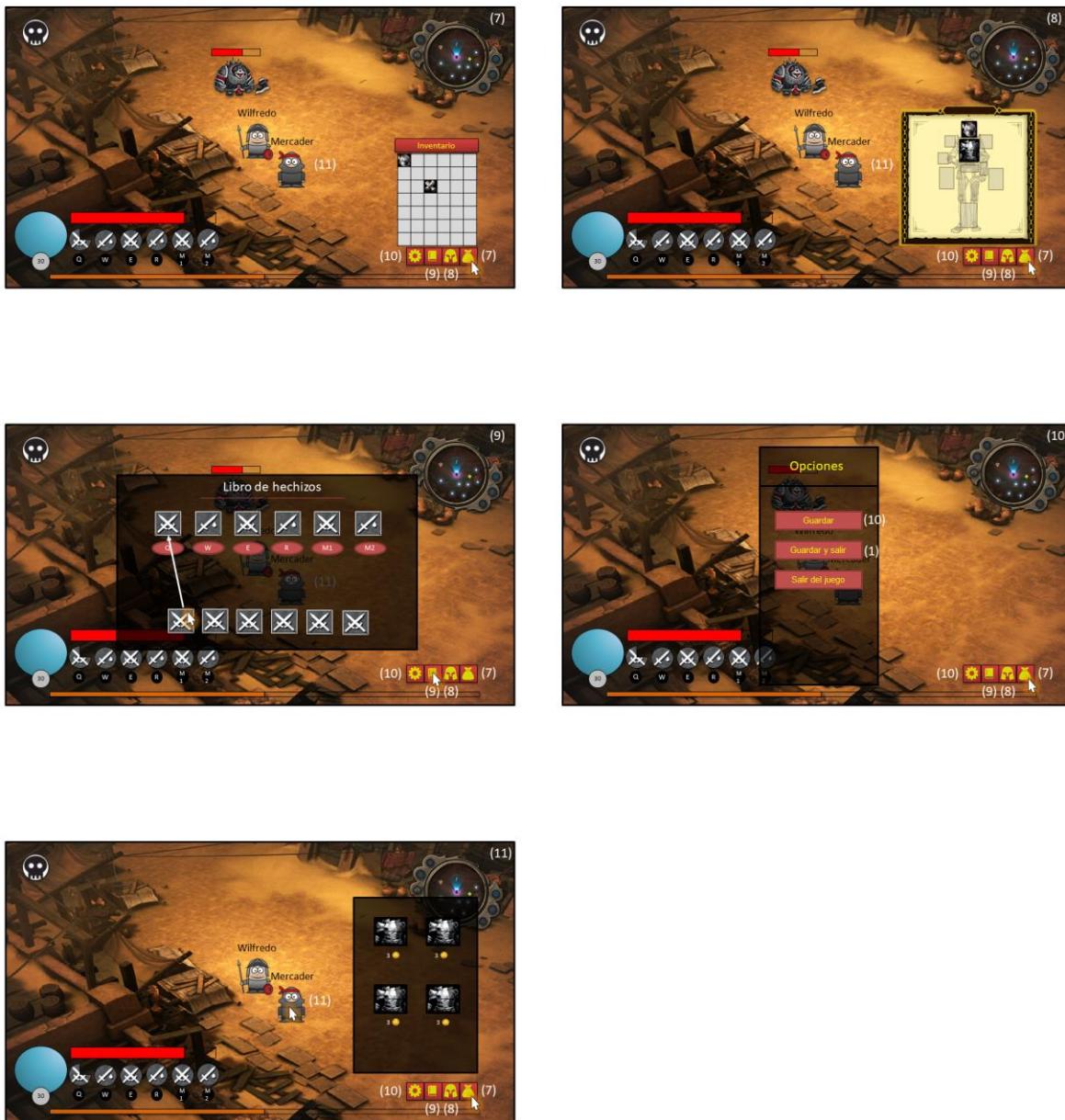


Ilustración 30. Prototipo de la interfaz. Dentro del juego en modo core

4.6. Flowboard

Flowboard es un gráfico [19] que mezcla dos tipos de diagramas: el diagrama de flujo (flow-chart), y la historia de usuario (storyboard). Este diagrama organiza los distintos modos de experiencia de juego, es decir, proporciona una estructura al juego. Cada elemento final será un modo o menú. Las flechas indicarán cómo pasar de un elemento a otro, de un modo a otro o a un menú.

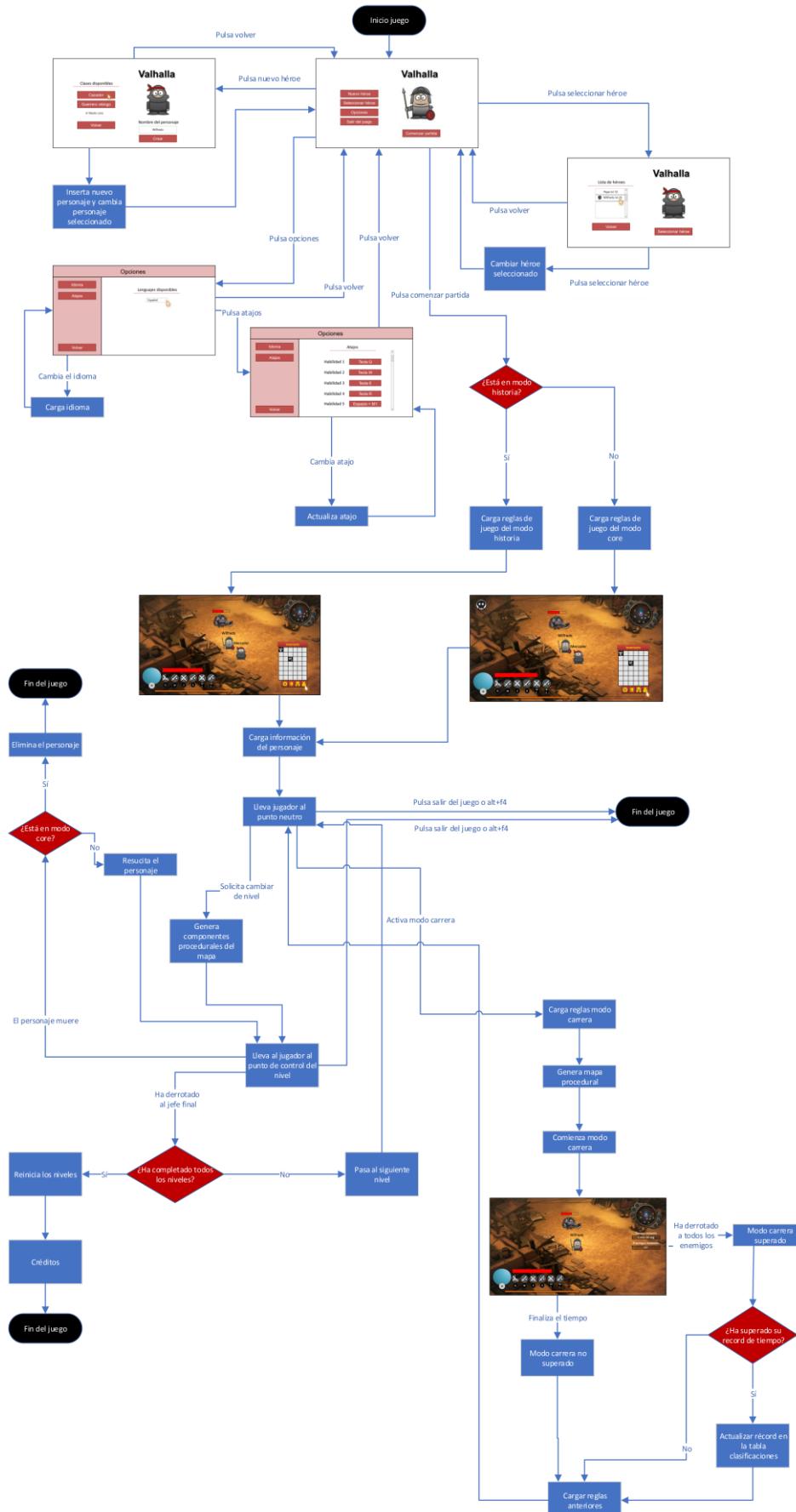


Ilustración 31. Flowboard

En el diagrama queda reflejado todo lo definido anteriormente, mostrando el flujo del videojuego. Se adjuntará un pdf para poder ver mejor en detalle el diagrama.

5. Sistema multijugador: Photon Network

Una de las características de este proyecto es el componente multijugador. Para poder implementar este tipo de sistema es necesario evaluar las posibilidades que proporciona Unity y elegir la más apropiada. Las soluciones para un sistema multijugador que proporciona Unity son:

- Utilizar la API³² de bajo nivel que proporciona Unity con NetworkManager [20].
- Utilizar la API de alto nivel que proporciona Unity (High-Level-API) [21].
- Utilizar el framework PUN de la empresa Exit Games [22].

La primera solución, es la mejor para crear un sistema multijugador bajo nivel, y además, sin ninguna limitación, pero esto implicaría una gran inversión de tiempo para crear todo el sistema desde cero. Como por ejemplo, implementar la sincronización de animaciones a nivel de bytes. También supone una inversión económica para mantener un servidor o servidores donde se alojará un proceso con la parte servidora del videojuego (en el caso de que uno de los clientes sea el servidor esta parte no haría falta) y también una base de datos en caso de necesitar almacenar información sobre cada jugador. Sin duda, si no hay ninguna opción que se ajuste a tu problema, y se desea una sincronización mucho más fluida, es la mejor opción.

La segunda solución, te abstrae de la comunicación entre los clientes y del paso de mensajes, nos proporciona una serie de clases que permiten hacer conexión a un servidor y sincronizar la información que necesita verse en los diferentes clientes. Como por ejemplo, las animaciones, algún elemento de la interfaz que comparten como la vida de los enemigos y la posición de cualquier objeto que sea capaz de moverse. Con esta API se ahorra mucho tiempo en la implementación de los componentes necesarios para sincronizar y mantener una conexión entre varios clientes, pero también se necesita hacer una inversión en un servidor o servidores.

³² Interfaz de programación de aplicaciones

Por último, PUN, Photon Unity Network, es similar a la anterior solución con el añadido de que proporciona un modo de conexión entre clientes sin importar la red en la que se encuentren, siempre y cuando el cliente tenga salida a Internet, y además, proporciona una versión gratuita con algunas limitaciones, pero más que suficientes para poder probar la API y la calidad de la conexión de los servidores.

Dada la magnitud del TFG y el poco tiempo del que dispongo, he optado por la elección de PUN para desarrollar el sistema multijugador debido a los siguientes factores:

- Proporciona una API a alto nivel que permite conectar y sincronizar clientes sin implementar nada.
- Permite la conexión entre jugadores sin importar dónde ni cuándo se conecten siempre que tengan conexión a Internet.
- No es necesario invertir en un servidor y proporciona una licencia básica y gratuita suficiente para desarrollar el prototipo.
- Se puede usar junto con PlayFab para almacenar información relativa a cada jugador: información de sus personajes, inventario, etc.
- También hay otras características interesantes como la sincronización de terrenos.

5.1. Arquitectura Photon Unity 3D Networking Framework

PUN es un framework para Unity que permite diseñar y desarrollar un sistema multijugador de un modo muy sencillo a través la creación de una cuenta y la instalación de un SDK en Unity. El sistema multijugador es exportable a cualquier tipo de plataforma que esté disponible en Unity, facilitando el desarrollo de cualquier tipo de videojuego (ver Ilustración 32).



Photon proporciona diversos servicios desde la conexión y sincronización de multijugador en tiempo real PUN, como otros servicios relacionados con la comunicación entre jugadores haciendo uso de un chat mediante mensajes de texto o voz.



Ilustración 32. Información general de Photon

Para tener acceso a cualquier servicio, es necesario crearse una cuenta totalmente gratuita que nos permite usar sus servicios e integrarlos en nuestro proyecto de un modo muy sencillo. En este caso, yo he hecho uso del servicio PUN para conectar jugadores y sincronizarlos en tiempo real. Photon nos permite usar todos sus servicios de un modo gratuito, pero con algunas limitaciones [23]. En el caso de PUN, son:

- Conexión máxima de 20 usuarios simultáneamente.
- Límite de envío de 500 mensajes por segundo.
- Límite de conexión de 8.000 jugadores activos mensualmente.

Aunque estas limitaciones quizás no permiten lanzar al mercado el software para un volumen de jugadores grande, sí permite en sus comienzos las pruebas pertinentes para desarrollar el proyecto y comprobar que el sistema es de calidad, para posteriormente ampliar el plan según se vaya necesitando. Los planes se pueden pagar o mensualmente o anualmente, y son los siguientes:

▪ Primer plan:

- Conexión máxima de 100 usuarios simultáneamente.
- Límite de envío de 500 mensajes por segundo.
- Límite de conexión de 40.000 jugadores activos mensualmente.
- Coste: 95\$ cada 60 meses.

▪ Segundo plan:

- Conexión máxima de 500 usuarios simultáneamente.
- Límite de envío de 500 mensajes por segundo.
- Límite de conexión de 200.000 jugadores activos mensualmente.
- Característica CCU Burst incluida.
- Coste: 95\$ mensualmente.

▪ Tercer plan:

- Conexión máxima de 1000 usuarios simultáneamente.
- Coste: 185\$ mensualmente por cada paquete de 1000.

Photon es usado por muchos videojuegos, como Time of Dragons, Modern Strike Online, Robocraft, Family Guy Online (ver Ilustración 33) entre otros muchos [24].



Ilustración 33. Juego de padre de familia realizado con unity y photon

A continuación, se van a describir algunas de las características del servicio multijugador de Photon más importantes y que serán necesarias en el proyecto.

5.2. Regiones

Photon Cloud proporciona un sistema de conexión por regiones [25], permitiendo conectar al cliente a la región más cercana que le garantiza la latencia más baja. La selección de región puede ser implementada por el desarrollador, o bien elegir el método de elección automática. Estas regiones se llaman Master Servers y son gestionados por Photon Cloud (ver Ilustración 34).

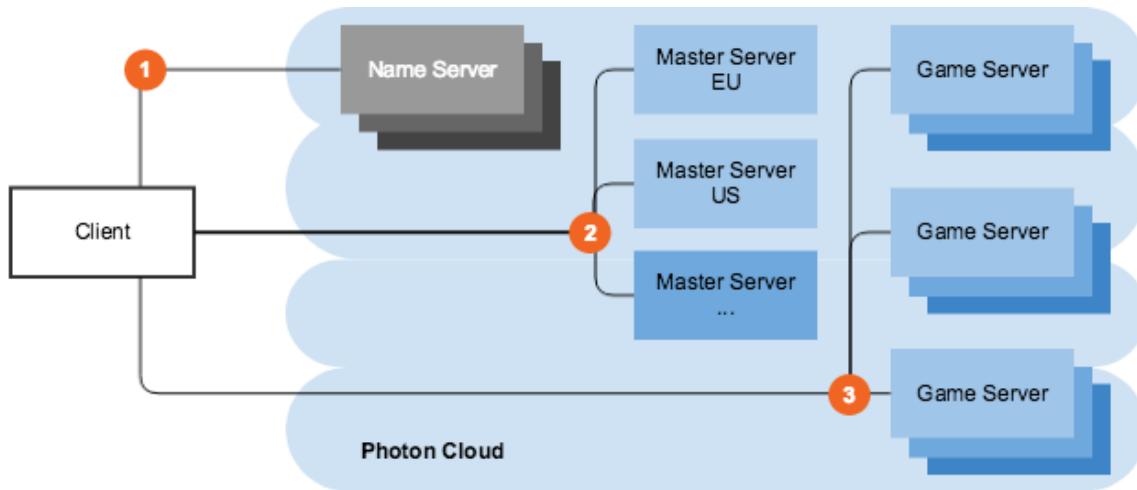


Ilustración 34. Arquitectura de la conexión con regiones

La lista de regiones disponibles viene definida en la siguiente tabla (ver Tabla 28).

Región	Lugar del servidor	Token
Asia	Singapore	asia
Australia	Melbourne	au
Canada, Este	Montreal	cae
China	Shanghai	cn
Europa	Amsterdam	eu
India	Chennai	in
Japón	Tokyo	jp
Rusia	Moscú	ru
Rusia, Este	Khabarovsk	rue
América del Sur	Sao Paulo	sa
Korea del Sur	Seoul	kr
USA, Este	Washington	us
USA, Oeste	San José	usw

Tabla 28. Tabla con el listado de regiones

Aunque en principio el cliente conecta a la región más cercana, se pueden limitar las regiones desde nuestra cuenta, permitiendo escalar poco a poco el proyecto en las primeras fases. Por ejemplo, en una fase beta o alpha del proyecto, se puede limitar a regiones muy específicas para ver cómo responde el plan seleccionado e incrementarlo si fuera necesario.

De todas las regiones, China puede dar problemas debido a la gran cantidad de bloqueos en red de este país, y se debe tratar de un modo independiente, como viene explicado en su documentación.

5.3. Lobby y salas

Una vez se ha hecho conexión con la región más cercana, el sistema pasa al modo de emparejamiento [26] a través de salas y lobbies (ver Ilustración 35).

Lobbies: photon organiza las salas entorno a un “lobby”. Por defecto hay un lobby, pero los clientes pueden crear nuevos lobbies. En un lobby los clientes son capaces de obtener las salas que están disponibles y visibles y unirse a ellas, si no se desea proporcionar este tipo de lista al jugador, es recomendable no incluir ningún lobby y hacer un emparejamiento aleatorio con un lobby por defecto para todas las partidas, como se explicará en las secciones posteriores. Hay varios tipos de lobbies: por defecto, SQL y asíncronos.

Salas: por otro lado, están las salas donde los usuarios pueden jugar y tener comunicación entre ellos. Las salas pueden ser creadas por los usuarios o unirse a una sala ya creada explícitamente, o unirse/crear una sala de un modo aleatorio sin necesidad de mostrar las salas disponibles. Las salas distinguen dos tipos de clientes:

- **Master Client:** será el cliente que hará de servidor y será el primero que entre en la sala, o dicho de otra forma, será el creador de la sala.
- **Cliente:** cualquier otro cliente que se conecte a la sala después de su creación.

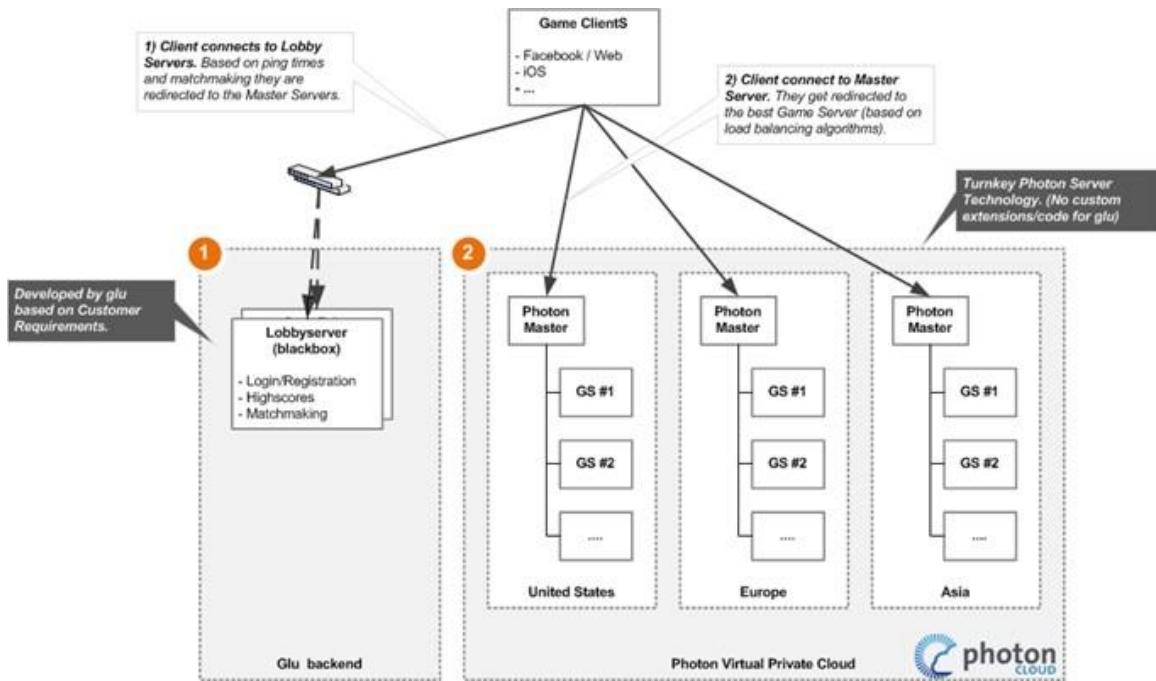


Ilustración 35. Sistema de salas y lobbies

5.4. Emparejamiento aleatorio

Es uno de los modos que tiene Photon de emparejamiento, y que permite unirte a cualquier sala de las disponibles de manera aleatoria, o bien a las que cumplan un determinado filtro que se especifique. Funciona como cualquier sala normal descrita anteriormente.

Este tipo de emparejamiento se suele usar en juegos como Clash Royale, donde se realiza un emparejamiento aleatorio pero con ciertos filtros, como por ejemplo, que los jugadores tengan un nivel similar.

5.5. Emparejamiento no aleatorio

Es el otro modo de emparejamiento y asociado a las listas de las salas de los lobbies. Es un emparejamiento que depende más del jugador, dándole la posibilidad de entrar en las salas que estén disponibles en la lista o crear una nueva.

Las salas con un emparejamiento no aleatorio tienen una serie de atributos configurables a la hora de crearlas:

- **Nombre** de la sala. Debe ser único.
- **Capacidad máxima**: máximo de jugadores que tendrá dicha sala.

- **Propiedades customizadas** con una hash table para añadir propiedades por nuestra propia cuenta.
- **Visibilidad de la sala:** se puede ocultar la sala en la lista del lobby.
- **Contraseña.**

También permite la búsqueda de amigos e invitación a salas.

5.6. Instanciación

En todos los juegos, es necesario instanciar uno o más objetos por jugador que deben ser replicados a su vez en los diferentes clientes, y sincronizados si tienen algún tipo de movimiento o interacción. Por ejemplo, si un jugador lanza una flecha se debe mantener sincronizada su posición, su rotación y las animaciones que tenga asociadas.

PUN permite la instanciación [27] de cualquier objeto a través de una función donde se le indica el nombre del objeto que debe estar alojado en la dirección “Assets/Resources”, la posición en el mundo y su rotación. Para poder instanciarlo es necesario que el GameObject tenga asociado el script PhotonView, que proporciona un identificador único asociado al propietario. Este identificador del objeto oscila entre el siguiente intervalo: [owner_id * 1000 + 1, owner_id * 1000 + 999], siendo owner_id un identificador único entero perteneciente a un jugador.

Por otro lado, se debe sincronizar [28] el movimiento, rotación, animaciones, entre otros. Photon ya proporciona unos scripts personalizados para mantener sincronizados posición, rotación y animaciones. Para ello, se deben asociar dichos scripts a PhotonView, así como cualquier otro que se utilice para sincronizar algún tipo de dato que se repite con mucha frecuencia en cortos periodos de tiempo.

5.7. RPC

Un RPC [29] es otro modo de sincronizar datos, pero que no ocurren con mucha frecuencia en un corto periodo de tiempo. Un RPC (Remote Procedural Call) es una llamada a un procedimiento remoto, proporcionando llamadas a métodos de instancias que se encuentran en diferentes clientes. Para hacer este tipo de llamadas es necesario tener acceso al elemento PhotonView, que proporciona la opción de llamar a un método de ese objeto en los diferentes clientes. Este tipo de llamadas permite sincronizar datos que no necesitan una actualización constante, como por

ejemplo el envío de una notificación a todos los jugadores para aceptar algún tipo de petición o informar sobre algo.

6. Sistema de autentificación y almacenamiento: PlayFab

En anteriores secciones, se propuso un diseño de base datos ante la necesidad de almacenar información relativa al jugador. En primer lugar, está la información relativa a la cuenta con la información básica del usuario (necesario para autenticar al usuario), y con conexión a la información de sus personajes y objetos conseguidos. Además del diseño, es necesario hacer un análisis y búsqueda de las posibles soluciones a efectos de implementación. Las posibles soluciones son las siguientes:

- Mantenimiento de un servidor propio o alquilado a alguna empresa, como OVH, para almacenar exclusivamente la base de datos.
- Servicio que ofrece Google con Firebase.
- Servicio PlayFab para Unity que incluye integración con Photon.

En la primera solución, implicaría gastos materiales para mantener un servidor, como una inversión de tiempo para efectos de implementación. Si se opta por un servidor propio, implicaría gastos en los equipos que hacen de servidor, los gastos de luz y alquiler del sitio en el que se encuentren los equipos, conexión a internet, obtención de una IP pública para tener acceso desde Internet a la base de datos, entre otros muchos factores en función de la magnitud del proyecto. Por otro lado, está disponible la opción de alquilar un servidor en empresas que proporcionan esta clase de servicios, como por ejemplo OVH [30], que permitiría alojar la base datos en un servidor personalizado y con acceso remoto a través de ssh u OpenSSH [31], o incluso a través de una interfaz gráfica, dando la posibilidad de cambiar de plan a lo largo del tiempo, permitiendo una adaptación a las necesidades del proyecto. En OVH, el plan más barato de servidor dedicado sería de 44.99€ [32]. Con respecto a la inversión de tiempo, en los dos tipos de servidores sería necesario establecer un sistema de autentificación seguro prácticamente de cero, pero tienen la ventaja de poder elegir la tecnología de base datos más apropiada, optar por una base de datos tradicional Oracle o quizás optar por una base de datos NoSQL, como puede ser MongoDB.

La siguiente solución, es el servicio Firebase [33] creado por Google. Es un servicio al estilo de Photon y PlayFab (se explicará a continuación) con un plan gratuito con algunas limitaciones, pero suficientes para probarlo a fondo. Firebase proporciona, entre otros, un sistema de autentificación muy completo, permitiendo acceso a través de cuentas de Google, Facebook o propias, y una base de datos en tiempo real con formato JSON, que nos permitiría autentificación y almacenamiento de datos sin necesidad de implementar estos dos componentes. Tiene soporte para Android, C++, Web, y Unity, pero con el inconveniente que sólo está disponible en Unity para plataformas móviles y tablets, tanto en Android como en IOs.

Como última solución, está PlayFab, y que al igual que Firebase proporciona un sistema de autentificación y base de datos sin necesidad de implementar nada y ni de mantener un servidor, con la diferencia de que está diseñado especialmente para videojuegos y ofrece la posibilidad de integrarlo con Photon, y por supuesto, sin limitaciones en el tipo de plataforma. Por tanto, ésta es la mejor solución, ya que ahorra el mantenimiento de un servidor, proporciona un sistema de autentificación sin necesidad de implementarlo por ti mismo, tiene un plan gratuito más que suficiente para hacer las primeras pruebas, y no limita la exportación del software a ninguna plataforma.

A continuación, se va a explicar brevemente PlayFab y cómo adaptar el diseño de la base datos a esta tecnología.

6.1. Arquitectura de PlayFab

PlayFab es una plataforma backend diseñada y construida exclusivamente para videojuegos. Puede ser integrada en los principales motores de videojuegos, como Unity y Unreal Engine 4, a través de un SDK³³ y exportado a cualquier plataforma.



³³ Software Development Kit: reúne un grupo de herramientas que permiten la programación de aplicaciones

PlayFab [34] proporciona múltiples servicios, como un sistema de autenticación, almacenamiento y gestión de información relativa a un jugador, torneos y clasificaciones globales, comercio, análisis de datos, etc. Las características principales y de las cuales se sacará partido son el sistema de autenticación y el almacenaje y gestión de la información del jugador, aunque también hay otros servicios interesantes como las clasificaciones globales, un componente atractivo para implementaciones futuras en el modo de juego carrera que se propuso en el diseño inicial del proyecto.

Por un lado, está el sistema de autenticación que permite la conexión a los demás servicios de PlayFab. El sistema permite la autenticación de un jugador a través de unas credenciales, que son el correo electrónico y una contraseña. Las cuentas usadas para autenticar al usuario pueden ser creadas con PlayFab o usar cuentas ya creadas en un sistema de autenticación externo [35], como Google, Steam, Facebook, etc.

Con este tipo de servicio, libera al desarrollador de preocuparse de cómo y dónde almacena la información de cada jugador, simplemente proporciona una serie de métodos para crear cuenta o iniciar sesión a través de cualquier tipo de cuenta mencionada anteriormente. Los métodos (al igual que en el sistema de la base de datos) tienen el formato siguiente:

- Petición al servidor con la información que debe suministrarse.
- Callback que se activa cuando la operación se ha completado con éxito.
- Callback que se activa cuando la operación no se ha podido completar.

La información devuelta por las peticiones que se realizan a PlayFab, se encuentra en formato JSON [36].

Para poder tener acceso a PlayFab, es necesario crearse una cuenta, de igual modo que Photon, a través de la cual se puede acceder a un panel (ver Ilustración 36) para gestionar la sincronización con el servicio Photon, gestionar las cuentas de cada jugador, comprobar la información almacenada en cada una de ellas, fechas de conexión, etc.

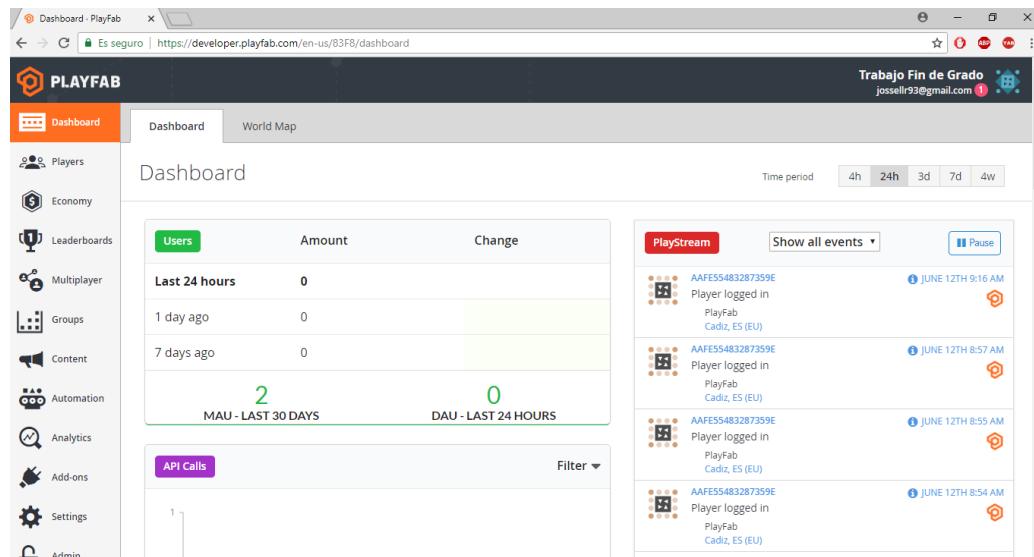


Ilustración 36. Panel de información de una cuenta de PlayFab

Cada cuenta tiene una serie de información principal muy parecida al diseño que se había hecho previamente. La información principal de cada cuenta es la siguiente:

- Identificador interno de la cuenta.
- Nombre de usuario.
- Cuenta de correo electrónico.
- Contraseña.
- Información de las horas que hizo conexión entre muchos otros.

Además de la información principal, la plataforma proporciona diversos modos de almacenamiento asociados a cada cuenta, como puede ser, información de personajes, su inventario, información general del jugador, amigos, estadísticas, posibles expulsiones (o conocido coloquialmente en el mundo de los videojuegos como ‘baneo’) y compras del jugador. La mayoría de estos servicios guardan la información y son actualizados a través de archivos con formato JSON.

6.2. Adaptación del diseño a PlayFab

El diseño de la base de datos debe de ser adaptado al sistema de la plataforma PlayFab. En primer lugar, no será necesario almacenar información de cada cuenta. PlayFab ya se encarga del sistema de autenticación, solamente se guardará información de los personajes que haya creado el jugador y la información de éstos.

Aunque el propio sistema proporciona una zona de inventario y personajes para almacenar la información, limita bastante la cantidad de objetos y personajes almacenados. Por tanto, he optado por almacenar la información en un diccionario de almacenamiento general de hasta 20.000 bytes como máximo de capacidad. El diccionario tiene como clave el nombre del personaje, único por cuenta, y como datos el resto de información relacionada con el personaje.

Para el diseño de la nueva base de datos es necesario tener en cuenta que usará el formato JSON propio de bases de datos NoSQL. El nuevo diseño de la base de datos con formato JSON sería como el siguiente ejemplo para una posible cuenta de jugador (ver Ilustración 37).

```
{  
    "name_character": "Wilfredo el Belloso",  
    "level": "7",  
    "current_experience": "2000",  
    "num_gold_skull": "0",  
    "num_silver_skull": "1",  
    "num_bronze_skull": "200",  
    "inventory": {  
        "general_items": [ ],  
        "armour_items": [  
            {  
                "id": "7",  
                "position_bag": "0",  
                "vitality": "5",  
                "strength": "10",  
                "intelligence": "10",  
                "agility": "0",  
                "endurance": "100",  
                "critical_strike": "0.05"  
            }  
        ],  
        "equipped_items": [  
            {  
                "id": "5",  
                "type": "0",  
                "vitality": "5",  
                "strength": "10",  
                "intelligence": "10",  
                "agility": "0",  
                "endurance": "100",  
                "critical_strike": "0.05"  
            }  
        ]  
    }  
}
```

Ilustración 37. Ejemplo de información guardada sobre un usuario en formato JSON

Para cada jugador se almacena:

- **Lista de personajes.**
- Para cada personaje se almacena:

- **name_character:** nombre del personaje y clave en el diccionario. Es necesario para poder actualizar y borrar el personaje.
- **level:** nivel actual del personaje a través del cual se calcula los valores de los atributos base, los hechizos desbloqueados y la experiencia necesaria para subir de nivel.
- **current_experience:** experiencia actual obtenida en el nivel actual.
- **num_gold_skull:** número de calaveras o abalorios de oro conseguidos.
- **num_silver_skull:** número de calaveras o abalorios de plata conseguidos.
- **num_bronze_skull:** número de calaveras o abalorios de bronce conseguidos.
- **inventory:** almacena a su vez otro nivel de atributos. En este caso, atributos que representan listas o arrays. Son los siguientes:
 - **general_items:** almacena objetos que no tienen información más específica asociada. Siempre almacenarán el identificador del objeto y la posición en el inventario del personaje. Actualmente estará vacío porque no hay ningún objeto de este estilo, pero se deja contemplado para un futuro.
 - **armour_items:** almacena objetos armadura donde además de guardar el identificador y la posición en la mochila, guarda todos los valores de los atributos que fueron generados proceduralmente en el videojuego.
 - **equipped_items:** almacena objetos armadura que tiene equipados, y se guarda por objeto lo mismo que en el caso anterior, con la diferencia que en vez de guardar la posición en la mochila se guarda el tipo de hueco donde va alojado en la armadura.

○

7. Sistema de niveles procedurales

El sistema de niveles procedurales es uno de los objetivos del trabajo fin de grado junto con el sistema multijugador. En primer lugar, se debe establecer la definición de nivel dentro de un videojuego. Una posible definición podría ser la siguiente: “*Un nivel, mapa, área, etapa, mundo, tablero, pantalla o zona en un videojuego es el espacio total disponible para el jugador a la hora de completar un objetivo específico. Los niveles suelen estar ordenados por dificultad*”. Además, se puede hacer distinción en dos tipos de sistema de niveles:

- Sistema de niveles estáticos.
- Sistema de niveles dinámicos y cambiantes.
- Una mezcla de ambos.

Por un lado, los **sistemas de niveles estáticos** son aquellos que no varían de una ejecución a otra, siempre es el mismo mapa sin ningún tipo de cambio o modificación de los elementos de éste. Un ejemplo es el World of Warcraft, el cual proporciona un gran mundo abierto con diferentes zonas en las que el jugador va progresando y cambiando de éstas cuando tiene nivel suficiente (ver Ilustración 38).

Por otro lado, están los **sistemas de niveles dinámicos** que varían de una ejecución a otra. Para conseguir este tipo de niveles, es necesario hacer uso de componentes aleatorias. Las componentes para conseguir esto son muy diversas, y depende mucho del tipo de juego y cómo se quiera orientar. Algunas de ellas pueden ser el modo de instanciación de los enemigos del nivel al entrar el jugador o jugadores por primera vez, variar los distintos accesos y salidas del nivel, sistema de enemigos con habilidades completamente aleatorias e impredecibles, o incluso hasta la generación completa del nivel que va desde generación del terreno hasta la generación de las texturas de éste, etc. Algunos ejemplos de videojuegos con este tipo de niveles son: Diablo 3, con instanciación aleatoria de enemigos, habilidades aleatorias de enemigos, misiones aleatorias, zonas de botín aleatoria o incluso los atributos de los objetos también son generados de forma aleatoria, entre muchos otros componentes. O Minecraft, con generación de mapas procedurales compuestos incluso con sistemas de cuevas, agua y lava.



Ilustración 38. Mapa de los niveles o zonas del juego World of Warcraft. En el mapa hay un grafo con el orden que siguen las zonas por complejidad

Por último, se pueden hacer una mezcla de **ambos tipos**, incluir un sistema de niveles en el que algunos de ellos estarán generados proceduralmente y otros no. Como por ejemplo, la inclusión de ciertos niveles neutrales en Diablo 3 que permiten el suministro de nuevos objetos y la adquisición de nuevos objetivos o misiones.

Dada la gran cantidad de componentes que permiten un sistema de niveles dinámico, la complejidad de algunos de ellos como puede ser la generación procedural completa de un terreno y sus texturas, y los demás objetivos del proyecto, he decidido implementar sólo los componentes que considero más imprescindibles. Por tanto, el modo carrera quedará propuesto para futuros prototipos.

Las componentes aleatorias de las cuales haré uso servirán para el sistema de niveles del modo historia. El terreno siempre será estático.

A continuación, se describen cada uno de los componentes.

7.1. Instanciación de enemigos

La instancia de enemigos en una zona concreta es un componente fundamental para mantener al jugador en continuo estado de alerta. Con un sistema

aleatorio de colocación de enemigos a lo largo de una zona se evita que el nivel siempre tenga la misma complejidad, que sea predecible, y que obligue al jugador a improvisar en cada momento sus acciones.

Una vez que se ha decidido el uso de este componente, hay que buscar posibles soluciones, y elegir la solución más eficiente y la que mejor resultados proporcione.

Como primera solución, se puede pensar en un sistema aleatorio sencillo en el que se instancia a los enemigos usando un punto 3D aleatorio, el cual representa la posición de éste, y que se obtendrá haciendo una tirada dentro de unos rangos permitidos, en este caso por ejemplo, del terreno. Para la obtención de este punto, se puede calcular la caja en 2D (la coordenada Z, que en terrenos suele representar la altura de éste, es irrelevante por ahora) que envuelve al terreno en el que se instancian los enemigos. Toda caja envolvente está formada por dos puntos, uno de ellos con sus coordenadas mínimas, y otro con sus coordenadas máximas, y por tanto, con estos dos puntos se pueden definir dos intervalos, uno por coordenada, donde el intervalo sería:

[min_coordenada_envolvente, max_coordenada_envolvente]

Por último, una vez se elige al azar un valor dentro del rango de posibles valores, tanto en la coordenada X como en la Y, hay que obtener el valor en la coordenada Z, ya que es un videojuego en 3D. Para ello se debe preguntar al terreno qué altura tiene en dicha coordenada (x, y) y situar al enemigo debidamente en dicho lugar.

Esta solución permite instanciarlos de manera muy sencilla y rápida, pero esto provocará distribuciones de enemigos muy dispares, dejando concentraciones de éstos en algunas zonas y otras demasiado vacías. Por así decirlo es una solución demasiado aleatoria, pero puede servir como paso inicial como parte de otro algoritmo más complejo. Una distribución no uniforme y tan aleatoria puede provocar:

- Posibles zonas muy densas de enemigos, en las cuales el jugador le resultaría casi imposible vencer y produciría una frustración en él.
- Posibles zonas muy vacías que haría el nivel mucho más fácil.

Para solucionar el problema de una distribución no uniforme, se puede hacer uso de una serie de restricciones que permiten o no instanciar al enemigo en dicho punto. Una posible restricción puede ser que no haya más de 5 enemigos en el área de influencia del enemigo en la posición prevista (ver Ilustración 39). Con la incorporación de este nuevo paso en el algoritmo, se puede obtener un terreno con una distribución uniforme con las restricciones adecuadas. En mi caso sería necesario establecer dos tipos de restricciones:

- Restricciones para los enemigos normales.
- Restricciones para los enemigos élites.

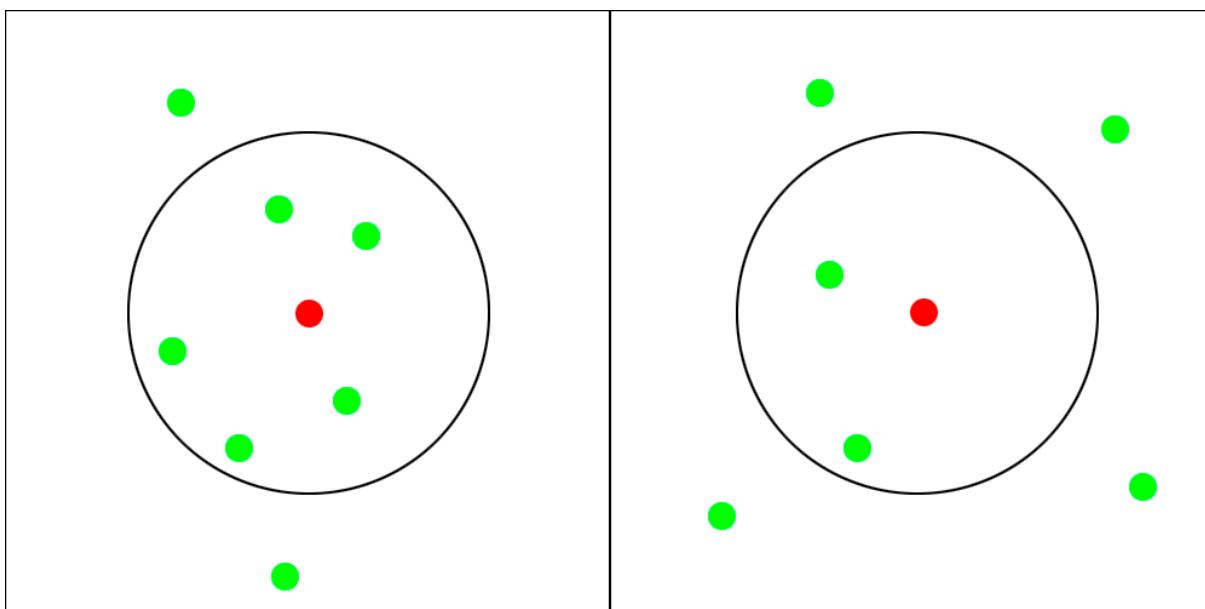


Ilustración 39. Ejemplo de restricción de 4 enemigos como máximo en la zona de influencia. Los puntos rojos son los enemigos que se están instanciando y los verdes los enemigos instanciados. La imagen de la izquierda no cumple la restricción y la derecha sí.

Por último, es necesario tener en cuenta un factor a la hora de colocar los enemigos en cualquier terreno de Unity (u otro motor gráfico). Tanto los personajes como los enemigos, se podrán mover a través de una característica de dicho motor gráfico, las mallas de navegación. Por tanto, es estrictamente necesario que los enemigos sean instanciados en una posición que esté directamente conectada con la posición donde el jugador empieza el nivel, es decir, si está dentro de la zona de acción del jugador.

También sería conveniente tener en cuenta la prohibición de ciertas zonas del terreno, donde de ningún modo debe de ser instanciado un enemigo, como por

ejemplo, la zona donde empieza el jugador o pueblos. Para el cálculo de estas zonas será necesario el punto central y el radio de acción de ésta.

7.1.1. Estructuras auxiliares para resolver el problema

Para poder llevar a cabo el algoritmo, es necesario hacer uso de una estructura auxiliar para comprobar si la posición de un nuevo enemigo cumple las restricciones definidas. Para ello, será necesario obtener el número de enemigos en una zona determinada alrededor de éste. Por tanto, es necesario una estructura espacial que obtenga los enemigos dentro de un rango de manera eficiente.

Como posibles estructuras espaciales para búsquedas eficientes en rangos están las siguientes:

- Estructuras arbóreas:
 - 2D-tree.
 - Range-tree.
 - R*-tree (R-tree balanceado).
- Malla regular en 2D (siempre y cuando los datos estén lo suficientemente distribuidos).

En términos de eficiencia, las estructuras arbóreas son la mejor opción, debido a que en la malla regular para obtener todas las casillas de un rango alrededor del enemigo es $O(n * m)$, siendo n el número de casillas a lo ancho y m a lo alto (para simplificar la zona de influencia será rectangular), y por ejemplo, una búsqueda por rango en un Range-tree es de orden $O(\log_2 n+k)$ para k puntos en $[x_1,y_1][x_2,y_2]$.

Dado que ese tipo de estructuras son más complejas, y también es necesario marcar ciertas zonas como prohibidas, he tomado la decisión de usar la **malla regular** en 2D. Pero sin duda, si se desea mejorar el tiempo del algoritmo en un futuro sería recomendable usar cualquiera de las estructuras arbóreas anteriores, aportando una solución para marcar con exactitud las zonas prohibidas.

Para marcar las zonas prohibidas en una malla regular, es semejante a obtener los enemigos dentro de un área, se obtienen todas las casillas dentro de esa área y se marcan como prohibidas a través de un booleano, por ejemplo.

Tal y como se dijo antes, la tercera componente es innecesaria, por lo que una malla en 2D será más que suficiente, ahorrando una gran cantidad de espacio y hueco completamente vacíos (principal problema de las mallas regulares en 3D).

7.1.2. Pseudocódigo del algoritmo

El pseudocódigo del algoritmo que se ha definido anteriormente es el siguiente:

```
function InstanciarEnemigosNivel ()
{
    grid2D ← CrearGrid2D(envolvente_terreno.min, envolvente_terreno.max, num_divisiones)
    enemigo_no_instanciado ← true

    for i = 0 hasta num_total_enemigos
    {
        while enemigo_no_instanciado
        {
            posicion_2D ← VectorAleatorio(envolvente_terreno.min, envolvente_terreno.max)

            if grid2D.NoEstaProhibida(posicion_2D) && grid2D.CumpleRestriccion(posicion_2D) &&
                EstaConectado(posicion_2D, punto_comienzo_jugador)
            {
                InstanciarEnemigo(Vector3D(posicion_2D.x, AlturaTerreno(posicion_2D), posicion_2D.y))
                grid2D.ColocarNuevoEnemigo(posicion_2D)
                enemigo_no_instanciado ← false
            }

            intento_actual++

            if intento_actual >= max_intento
            {
                enemigo_no_instanciado ← false
            }
        }

        intento_actual ← 0
        enemigo_no_encontrado ← true
    }
}
```

7.1.3. Resultados

Antes de usarlo en cualquier nivel del videojuego, se probó en un terreno cualquiera, en el cual se colocó una posición de comienzo del jugador que marca la zona transitable del nivel, y se definió un número de enemigos normales, un porcentaje de estos del tipo élite, y unas restricciones. Por otro lado, se dividió el terreno en dos partes, zona conectada con la posición de comienzo de cualquier jugador, y la zona que no está conectada con la anterior, de este modo los enemigos solo deben instanciarse en la primera zona.

Los valores eran:

- **Número de enemigos normales:** 300.

- **Número de enemigos élite:** 5% de la población de enemigos normales.
- **Restricciones:**
 - Enemigos normales: 10 radio de influencia, 5 enemigos como máximo.
 - Enemigos élite: 30 radio de influencia, 1 enemigo como máximo.

Las ejecuciones del algoritmo, se completaron en un intervalo de 1 o 2 segundos, y consiguieron esparcir de manera uniforme los dos tipos de enemigos. A continuación se muestran 3 ejecuciones (ver Ilustración 40, Ilustración 41, Ilustración 42), y el modo de conexión de la malla de navegación del terreno (ver Ilustración 43, Ilustración 44).

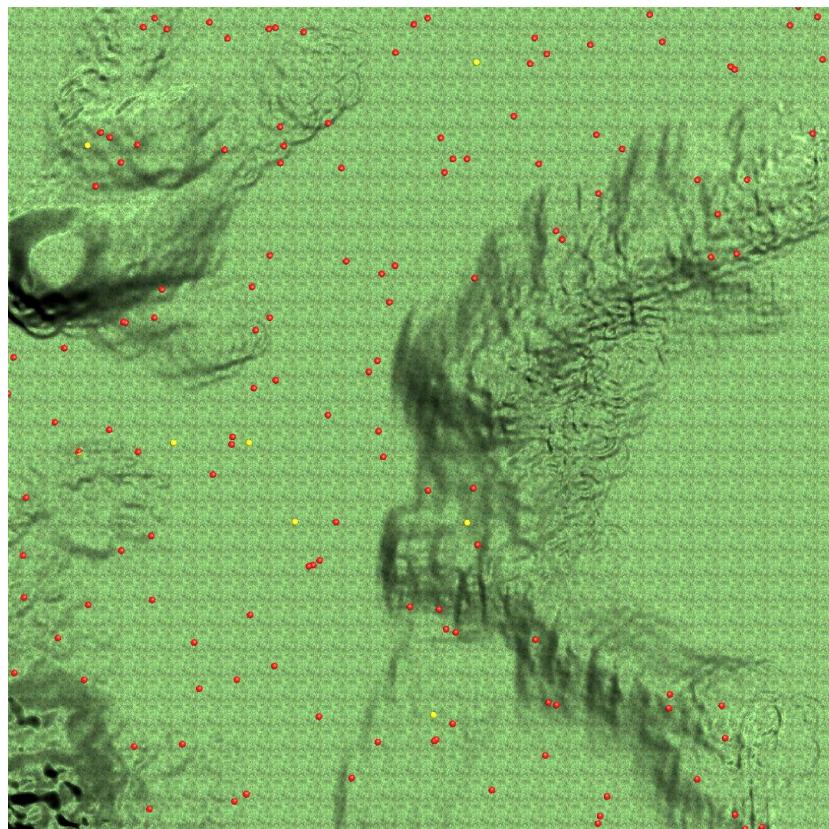


Ilustración 40. Primera instanciación procedural

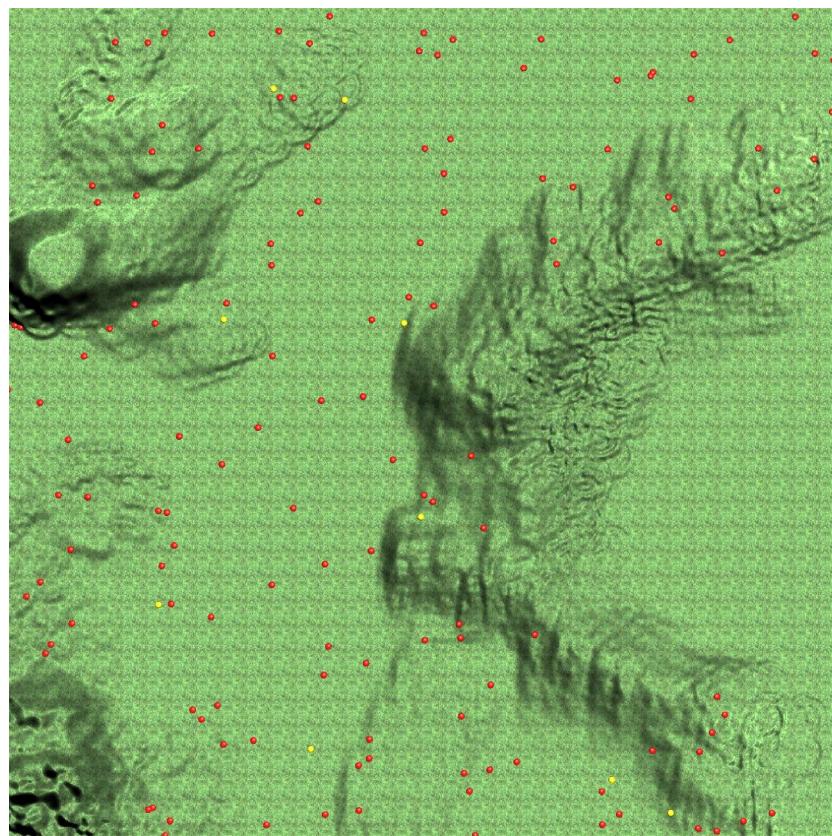


Ilustración 41. Segunda instanciación procedural

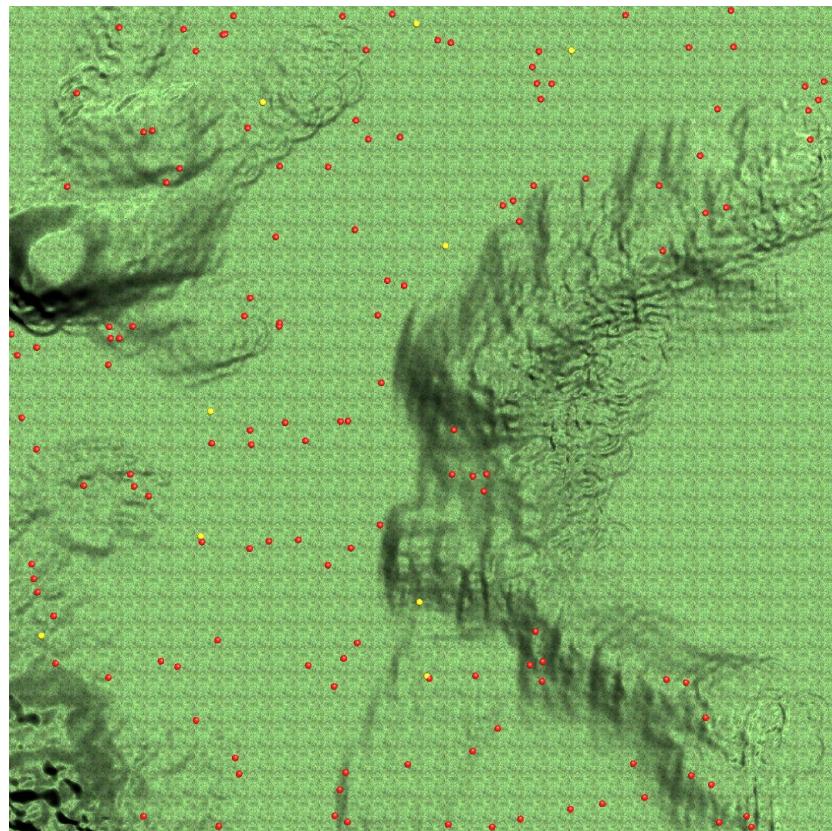


Ilustración 42. Tercera instanciación procedural

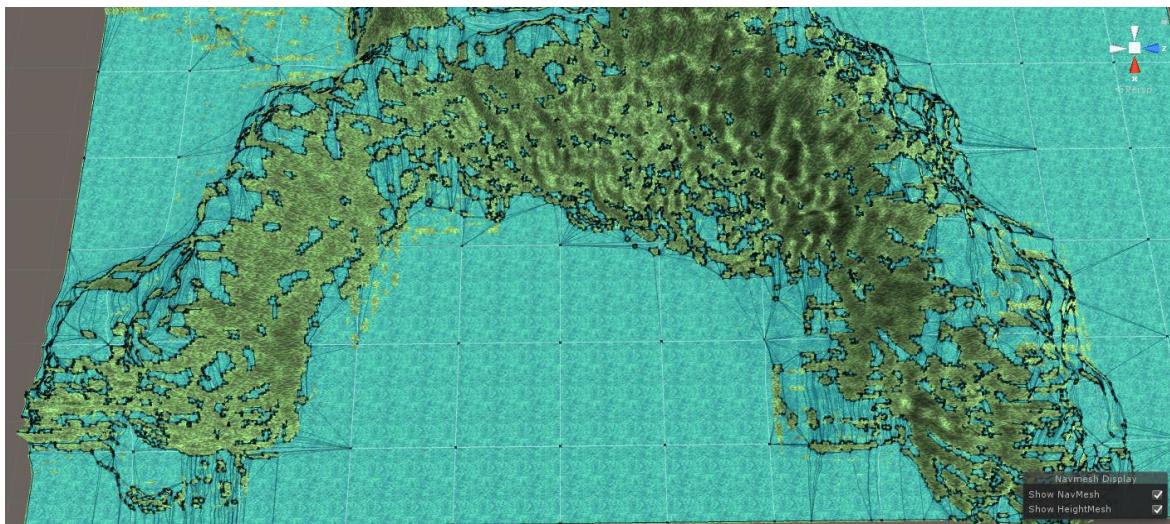


Ilustración 43. Conexiones del lateral izquierdo del mapa donde se realizó la instanciación

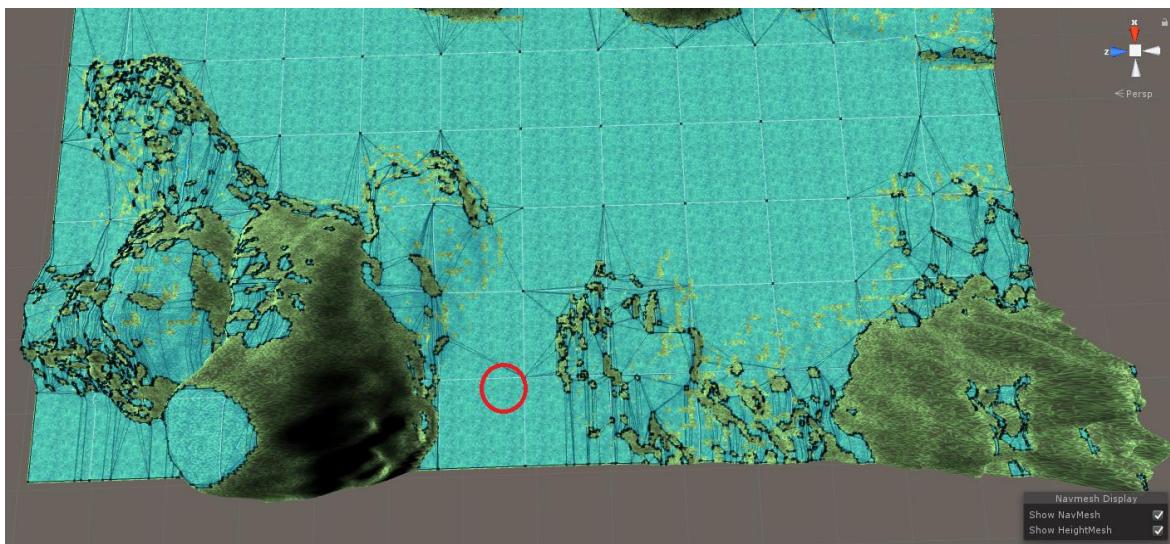


Ilustración 44. Conexiones del lateral derecho del mapa donde se realizó la instanciación. El círculo rojo es el punto de respawn del jugador

7.2. Sistema de cofres

Una de las mecánicas que permite la obtención de objetos son los cofres. Para la colocación de los cofres se realizan los siguientes pasos:

- Se coloca un GameObject³⁴ [37] vacío que tiene como hijos todas las posibles posiciones de los cofres en cada nivel (también son GameObjects vacíos, sólo tienen posición). En definitiva es una lista de posibles posiciones.

³⁴ Objeto básico en cualquier videojuego en el motor Unity.

- Se le proporciona al script encargado de la instanciación de cofres la lista de cofres y una probabilidad de crear el cofre. Una vez el nivel comienza, se realizan tiradas en un intervalo de $[0, 1]$ en cada posición, y se colocará un cofre si y sólo si la tirada es menor o igual que la probabilidad proporcionada.

Antes de crear la lista, hay que pensar dónde podrá haber un cofre, y distribuirlos de la mejor manera posible. A continuación se muestra las posibles posiciones de cofres en el primer nivel del videojuego (ver Ilustración 45).

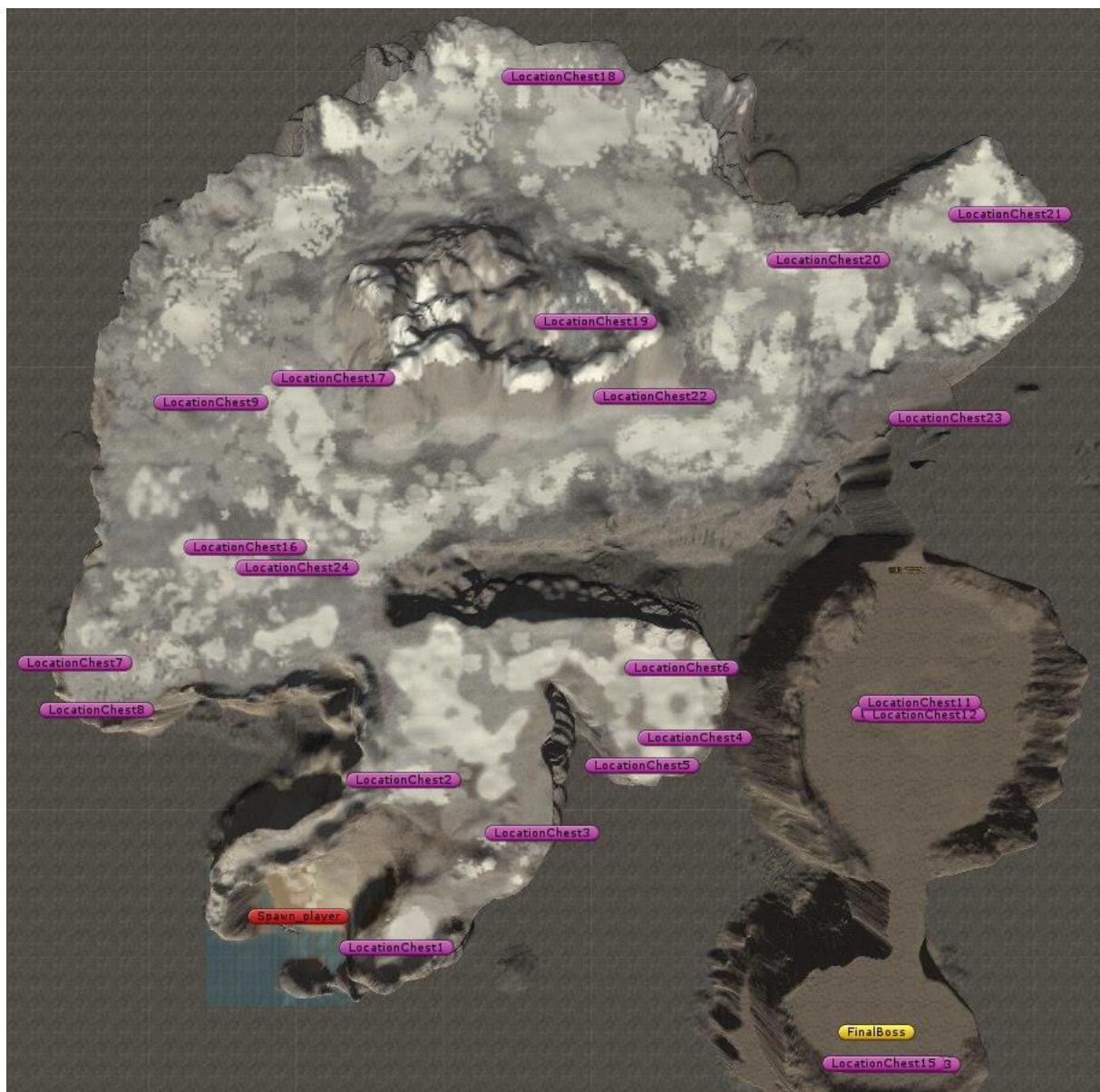


Ilustración 45. Posibles posiciones de los cofres en el nivel uno

7.3. Habilidades aleatorias de los enemigos elites y atributos de objetos aleatorios

Las habilidades aleatorias de los enemigos de un nivel, o los objetos equipables que dejen caer con atributos aleatorios también son parte del sistema aleatorio o procedural del sistema de niveles. Éstos han sido definidos ya en las mecánicas, y en la sección del prototipo final se explicará cómo ha quedado finalmente.

8. Arquitectura y diseño software

En esta sección, se tratarán los diseños y arquitectura software del proyecto. Se trata en una sección diferente a la cuarta debido a que aquí se presenta el diseño final total de todo el software implementado. Aunque a continuación se muestra la solución final a efectos de implementación, este diseño ha sufrido pequeños cambios a lo largo de la vida del proyecto, como por ejemplo la incorporación de algunos atributos a ciertas clases, o la incorporación de alguna clase con la que no se había contado anteriormente, pero la esencia y el diseño inicial ha permanecido a lo largo de éste.

En las siguientes imágenes aparece el diseño de clases del sistema de entidades inicial (ver Ilustración 46), y los atributos de la clase padre (ver Ilustración 47).

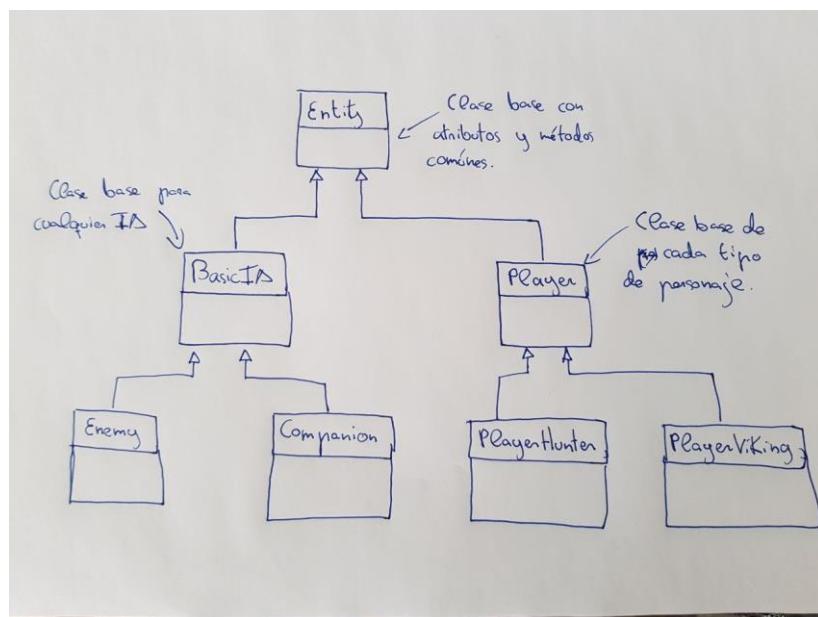
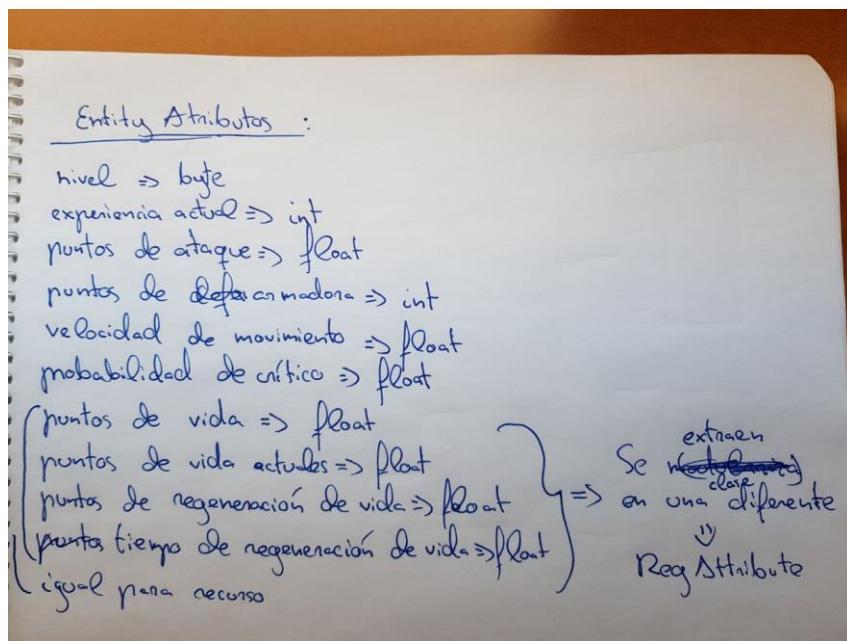


Ilustración 46. Sistema de clases para las entidades en el diseño inicial en papel

**Ilustración 47.** Atributos de la clase entidad en el diseño inicial

El proyecto en Unity, consta de un total de 79 ficheros creados únicamente por mí. Todos y cada uno de los ficheros, así como la definición de las clases, utiliza el siguiente esquema: **ValNombreFichero³⁵** o **ValNombreClase**. De este modo, se evita conflictos de nombres con otros paquetes externos. Estos ficheros tienen extensión cs, ya que como vimos, Unity utiliza el lenguaje C# (C Shard) y Javascript. En mi caso he usado el primer lenguaje.

Por otro lado, el proyecto está compuesto para una serie de espacios de nombres por los siguientes motivos:

- Si hay varias clases con el mismo nombre y pertenecen a un espacio de nombres diferente, no habrá problemas de ambigüedad.
- Si incluimos las clases en un espacio de nombres particionado, el programador podrá elegir qué paquetes de clases quiere cargar y no cargar todas las clases al completo.
- Evita la inclusión de las cabeceras de clases necesarias que sean del mismo espacio de nombres.

³⁵ El inicio Val viene del nombre del videojuego Valhalla

Todos y cada uno de los ficheros parten de un único espacio de nombres llamado **ValhallaGame**, y de éste derivan espacios de nombres que separan los ficheros por diferentes funcionalidades. Los espacios de nombres son los siguientes:

- **ValhallaAbility**: espacio de nombres que engloba todos los ficheros relacionados con las habilidades personajes, enemigos y compañeros.
- **ValhallaMovementController**: espacio de nombres que engloba todos los ficheros relacionados con el control de movimientos del personaje.
- **ValhallaStructures**: espacio de nombres con las estructuras creadas única y exclusivamente para el proyecto.
- **ValhallaEditor**: espacio de nombres para todos los ficheros que modifican de alguna forma la interfaz del motor Unity en modo depurador.
- **ValhallaEntity**: espacio de nombres para los ficheros que forman parte del sistema de entidades.
- **ValhallaGeometry**: espacio de nombres para los ficheros encargados de la geometría tanto para 2D como para 3D en un posible futuro.
- **ValhallaItem**: espacio de nombres que engloba los ficheros relacionados con los objetos en Valhalla.
- **ValhallaJSON**: espacio de nombres con las utilidades del formato JSON.
- **ValhallaNetwork**: espacio de nombres con los ficheros que se encargan de la lógica en el sistema multijugador.

8.1. Sistema de scripting en Unity

Para realizar el diseño de clases, es muy importante conocer el modo de programar en el motor en el que se desarrolle el proyecto, que será definido por las herramientas de las que disponemos. En este caso, en Unity, independientemente del lenguaje (C# o Javascript), utiliza un sistema de scripting que permite la construcción de componentes que se pueden asociar a cualquier GameObject. El conjunto de scripts y otros componentes que vienen ya por defecto, definen lo que representa cada elemento.

Los scripts heredan por defecto de la clase MonoBehaviour, que proporciona una serie de atributos y métodos. Algunos de los más importantes son:

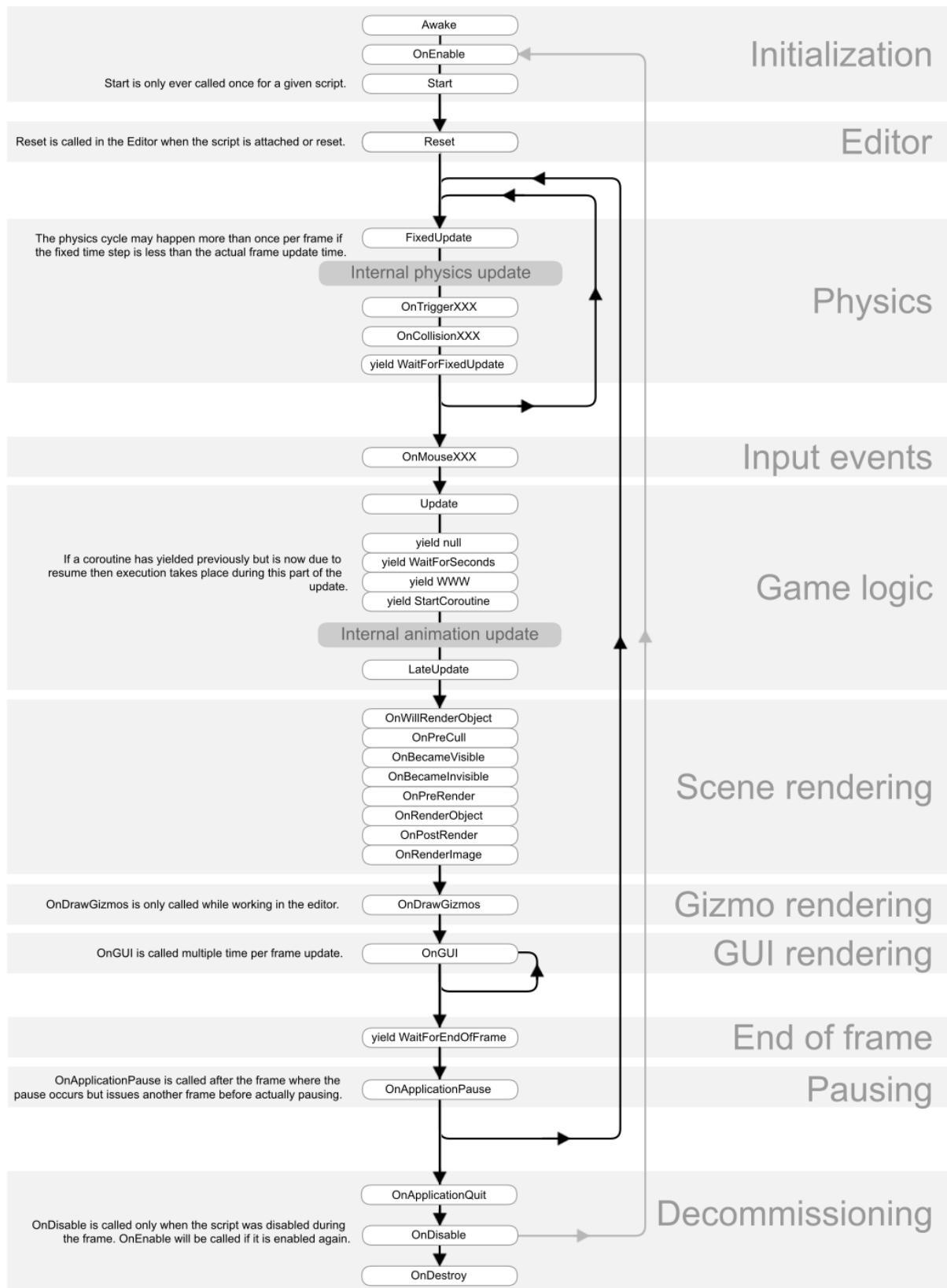
- Acceso al componente Transform, que se encarga de almacenar la posición en el mundo del objeto, la posición local si forma parte de una jerarquía de objetos y los GameObject hijos y padre (si lo tuviera).
- Acceso a su referencia como GameObjet en general.
- Posibilidad de acceder a los diferentes componentes (script) que tiene asociados el elemento, así como añadirle otros o incluso eliminarlos.
- Destruir el GameObject de la escena.

También proporciona una serie de callbacks³⁶, que son llamados cuando se producen unos eventos muy concretos o en ciertas etapas de la vida del GameObject (ver Ilustración 48). Los callbacks más importantes y más usados son:

- **Start**: método callback llamado al comienzo de la vida del GameObject.
- **Update**: método callback llamado en cada frame.

Por otro lado, es importante saber que a priori no se sabe que GameObject ejecutará sus scripts antes que otro, es decir no se puede saber el orden de ejecución de cada uno de los objetos de la escena (en las últimas versiones es posible establecer el orden de scripts). Lo único que sabemos es que hay un orden de ejecución de callbacks en la vida de todo GameObject, y que hasta que todos los objetos de la escena no pasen por dicha etapa no pasarán a la siguiente [38]. Esto es importante de cara al diseño.

³⁶ Métodos que responden a un determinado evento, realizando la acción para la que están programados

**Ilustración 48.** Callbacks y orden de llamadas en la vida de un script en Unity

Otra característica importante de Unity, y que será utilizada en el proyecto, son los Scriptable Object que son simplemente scripts que no necesitan ser asociados a ningún tipo de GameObject (otra de las limitaciones de los scripts) [39].

8.2. Arquitectura cliente-servidor

Antes de comenzar con el diseño, es importante explicar la arquitectura cliente servidor que se ha usado, y que por tanto, tendrá repercusión en el sistema de clases. La arquitectura que toma este tipo de sistemas, suele ser cliente-servidor. En algunos casos, hay servidores que mantienen una única instancia del videojuego y todos los clientes se conectan a dicho servidor (suele haber varios servidores y no un único servidor, dividiendo a los jugadores por regiones). Un ejemplo es World of Warcraft, que proporciona una serie de servidores por zonas.

Otra posibilidad, es que haya un sistema servidor que se encargue de emparejar a todos los jugadores en salas o partidas, y almacene información de éstos, pero realmente quien crea la partida y sincroniza todos los aspectos, es uno de los clientes de la partida. Es decir, hace de servidor en la partida. Para ello, habría que implementar un sistema de migración de partida a otro cliente en caso de que el cliente servidor desconecte, y también un sistema de seguridad para que los datos finales de cada partida no sean devueltos al verdadero servidor modificados fraudulentamente. Un ejemplo de este tipo de sistemas es el videojuego Warframe³⁷.

Para este caso particular, haré uso del segundo tipo de sistema cliente-servidor. El servidor principal será Photon para la parte del sistema multijugador, y PlayFab para la parte de la base de datos remota. El cliente servidor será el primer jugador que cree la sala (ver Ilustración 49), y se encarga de:

- Sincronizar terrenos.
- Crear y mantener las instancias de los enemigos. Los demás clientes sólo tendrán una sincronización en cuanto a posición y animaciones de éstos.

Para simplificar el sistema, el prototipo final no tiene migración de servidor en caso de desconexión, simplemente los clientes terminan la partida y vuelven al menú principal.

³⁷ <https://www.warframe.com/>

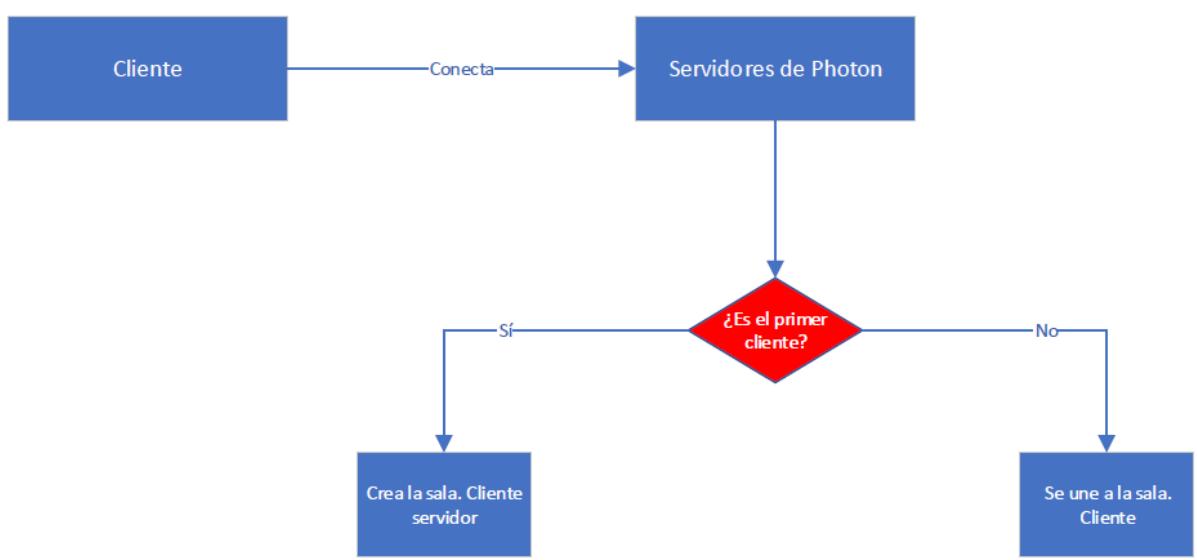


Ilustración 49. Diferencia entre cliente servidor y cliente normal

8.3. Patrones de diseño utilizados

Los patrones de diseño son un elemento esencial en el diseño e implementación software, permitiendo un código de calidad y sencillo. Como definición, se puede decir que son: “*Técnicas ya definidas para resolver problemas comunes en el desarrollo software y otros ámbitos referentes al diseño de interacción o interfaces*” [40] [41].

8.3.1. Patrón de diseño Singleton

Es uno de los patrones de diseño más utilizados y uno de los que se han visto a lo largo del grado. Este patrón permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Consiste principalmente en garantizar que una clase sólo tenga una instancia y proporciona un punto de acceso global a ésta.

Aunque es uno de los más utilizados, no es recomendable su uso en sistemas donde la concurrencia es vital, llegando a desaconsejar o prohibir su uso en videojuegos. El principal problema es que para implementarlo, es necesario guardar la única instancia como atributo de esa clase y del tipo estático. Es decir, un atributo con un acceso global.

Dado este problema, es necesario buscar otra solución o garantizar el patrón Singleton sin el uso de variables globales. Para ello, se ha hecho uso de una técnica para el patrón Singleton en videojuegos [42] (también dispone de una versión online:

<http://gameprogrammingpatterns.com/contents.html>). Esta técnica propone mantener una única clase que implementa verdaderamente el patrón Singleton, y que se encargue de mantener referencias a una única instancia de las clases que por definición lo necesitan, evitando problemas de concurrencia. La clase que efectúa esta técnica es la clase **ValGame**, que será explicada posteriormente. Aunque esta técnica resuelve el problema, para hacer un buen uso de este sistema, las clases que necesitan alguna referencia de un objeto único de alguna clase, deberán pedir la referencia a través de la clase Singleton una única vez al comienzo de su vida, y almacenarlo como atributo. Si no se mantuviera la referencia de la instancia y se pidiera constantemente a la clase Singleton, tendríamos el mismo problema de concurrencia.

La implementación del patrón Singleton en C-Shard, al igual que en otros muchos lenguajes, se realiza definiendo un atributo estático a sí mismo, ocultando el constructor con una visibilidad protegida y dando la oportunidad de poder ser heredado (aunque también sería válido definirlo como privado), y proporcionando un método GetInstance que crea la única instancia si no se ha hecho ya, o devolviéndola si ya está creada.

Las demás clases que tendrán una única instancia sin ser Singleton, deberán registrarse en la clase ValGame al comienzo de su vida, y hacer el proceso inverso en caso de ser destruidas. Todas estas clases son lo que se llama en Unity scripts, y para mantener una única referencia, deberán definir un booleano estático para indicar si ya han sido instanciadas al menos una vez. Proceso:

- Al comienzo de su vida comprueba si ya hay una instancia creada, si no la hay sigue el proceso y modifica el booleano.
- Si ya hay una instancia creada se autodestruye.

Hay alguna excepción, como es el caso de la clase **VallInterfaceMainMenuManager** que sí implementa como tal el patrón, ya que es innecesaria la concurrencia.

8.3.2. Patrón de diseño Observer

El segundo, es el patrón de diseño de software Observer, que define una dependencia del tipo uno a muchos entre objetos, de tal manera que cuando uno de

los objetos cambia su estado, notifica este cambio a todos los dependientes. Se trata de un patrón de comportamiento.

Es utilizado en múltiples clases del videojuego, y se utiliza a través de una clase que lo representa, **ValObserverPattern** (ver Ilustración 50). A través de dicha clase, los objetos dependientes realizan subscripciones para recibir notificaciones de un objeto en cuestión. La clase anterior permite:

- Realizar una subscripción a través de su referencia como GameObject, y suministrando el método que se usará para notificar los cambios. El método pasado por parámetro, debe seguir el modelo de método delegate definido por ValObserverPattern.
- Eliminar la subscripción pasando su referencia de GameObject.
- Notificar individualmente y masivamente el nuevo estado del objeto, haciendo uso del método evento que se suministró.

ValObserverPattern define un modelo de método delegado (evento), al cual se le puede pasar un número indeterminado de parámetros, con lo cual permite a cualquier clase hacer uso de este elemento y adaptar los parámetros pasados según la necesidad.

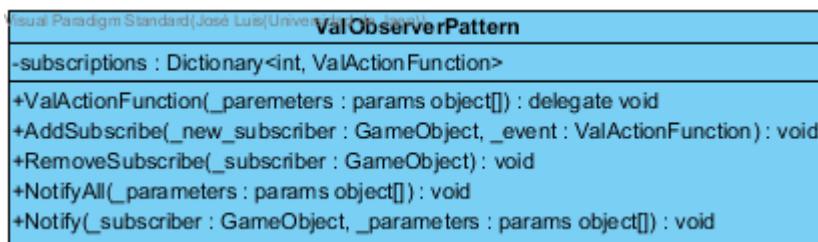


Ilustración 50. UML de la clase que representa el patrón observador

Por último, cabe destacar que el diccionario que almacena todos los métodos evento de cada objeto subscriptor, utiliza como clave el identificador único del GameObject [43].

Este patrón es utilizado para actualizar el nivel de los enemigos cuando el jugador local sube de nivel, y para compartir la referencia del jugador local con los elementos que necesitan su referencia (recordemos que no se sabe el orden a priori de los elementos de una escena).

8.3.3. Patrón de diseño DAO

Por último, está el patrón de diseño DAO, Data Access Object. Es un componente software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos. La interfaz está definida por el siguiente UML (ver Ilustración 51).

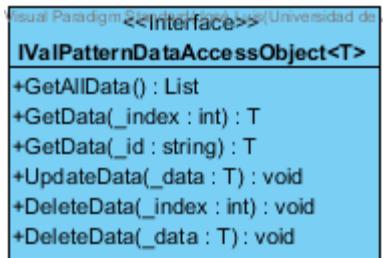


Ilustración 51. Interfaz común para DAO

La interfaz es una clase generalizada (template), y proporciona los siguientes métodos:

- **GetDataAll**: proporciona una lista con todos los datos almacenados.
- **GetData**: proporciona el dato que se identifica con dicho identificador.
- **GetData**: igual que el anterior, pero esta vez el identificador es una cadena de caracteres.
- **UpdateData**: actualiza un dato en concreto.
- **DeleteData**: elimina un dato dado su identificador único.
- **DeleteData**: elimina el dato pasado.

Este patrón es usado por dos clases que manipulan información de una base de datos. En primer lugar, la clase **ValDatabaseManager** encargada de la base de datos remota en PlayFab, y en segundo lugar, la clase **ValDatabaseItemsManager** encargada de la base de datos de objetos local. Ambas serán definidas a continuación.

8.4. Diseño de clases

En esta sección se mostrará y explicará el diseño de clases finales que he realizado para este proyecto. Estará dividido en diferentes paquetes, tal y como fueron separados en nombres de espacios por funcionalidades parecidas. A su vez, cada paquete estará dividido en múltiples clases, dividiendo el trabajo en tareas sencillas que permiten depurar el código de un modo más eficaz.

Los siguientes UML que se muestran, únicamente contendrán métodos y atributos significativos, y que definen la clase. Los demás estarán disponibles en el código, o bien en la documentación que se adjuntará de éste. Aquellas clases que heredan de **MonoBehaviour** son scripts de Unity.

Cada una de las clases, atributos y variables contienen nombres que auto documentan el código. Por otro lado, también se ha utilizado para cada una de las clases, métodos y atributos una documentación explicativa del tipo Doxygen. Por último, cabe destacar que la nomenclatura usada en el código es la siguiente:

- Nombres de clases, propiedades y métodos hacen uso de CamelCase. Es decir, palabras separadas por mayúsculas. C# usa la misma notación.
- Los atributos de las clases hacen uso de separadores con el carácter “_”, todas las letras son minúsculas y por último, no se hace uso de números.
- Los atributos locales y pasados por parámetros tienen la misma notación que las de los atributos, pero con la diferencia que empiezan siempre con el carácter “_”. Este tipo de nomenclatura es muy personal y lo hago para diferenciar rápidamente de los atributos de clase.

Antes de pasar a cada uno de los paquetes, se mostrarán las clases que no pertenecen a ningún paquete de una componente en particular.

En primer lugar, está la clase **ValCommonFuntions**, que engloba todos los métodos que son usados más comúnmente por la mayoría del resto de clases (ver Ilustración 52). Los métodos son:

- **IsEnemy, IsCompanion, IsPlayer, IsTerrain, IsProjectile:** comprueban si el GameObject es de un tipo concreto (enemigo, compañero, jugador, etc).

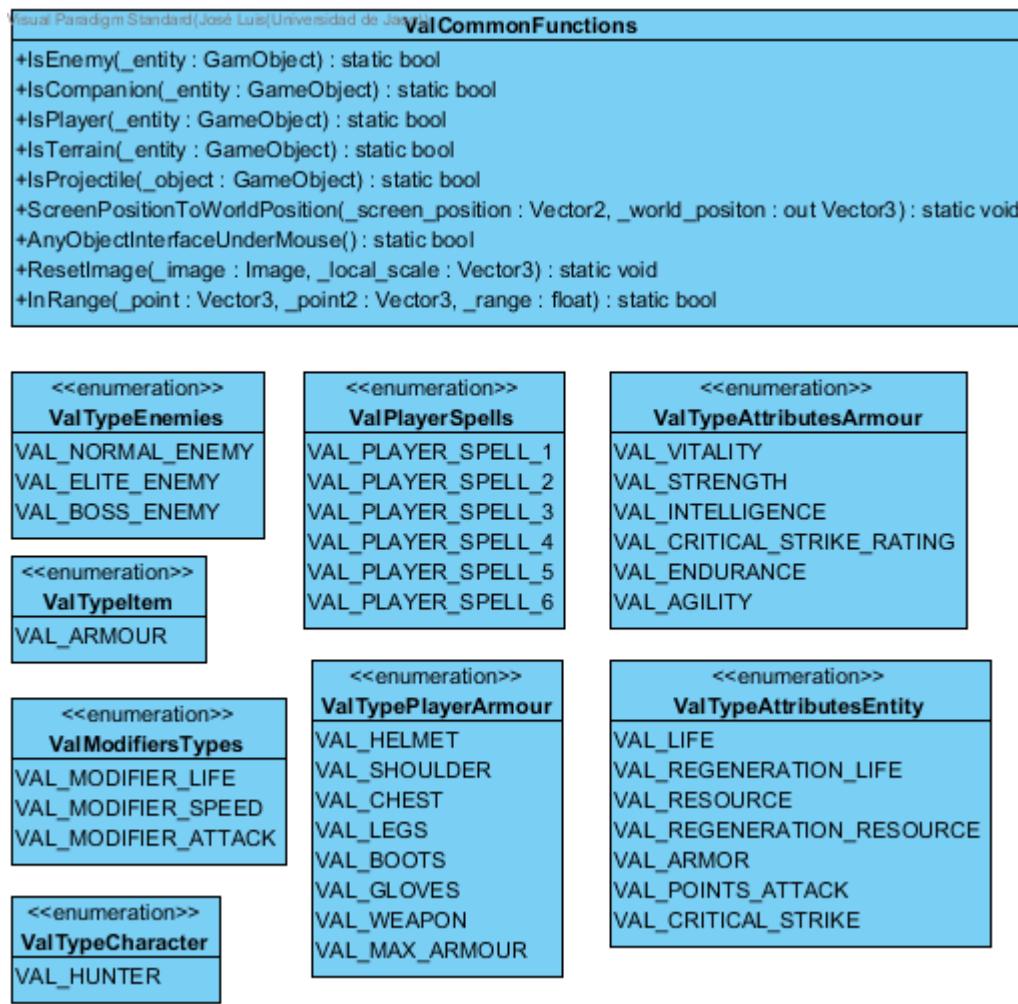


Ilustración 52. UML de las clases del tipo enum y de la clase ValCommonFunctions con métodos estáticos que son usados por un gran número de clases

- **ScreenPositionToWorldPosition:** convierte de coordenadas de pantalla a coordenadas de mundo.
- **AnyObjectInterfaceUnderMouse:** indica si hay algún elemento de la interfaz con el que se puede interaccionar en la posición del ratón.
- **ResetImage:** reinicia el vector de escalado de una imagen en la interfaz.
- **InRange:** comprueba si dos posiciones están en un rango específico.

Por otro lado, hay una serie de clases del tipo enum para enumerar una serie de tipos, los cuales suelen utilizarse a modo de índice en un vector, y otorga una mayor claridad de lo que hay almacenado en dicha posición o lo que se está haciendo en un determinado momento. Las clases son las siguientes:

- **ValTypeEnemies:** enumera los tipos de enemigos.

- **ValTypeItem**: enumera los tipos de objetos. Por ahora sólo hay un tipo, pero en un futuro se pueden añadir más tipos de objetos, como pueden ser las pociones.
- **ValModifiersType**: enumera los tipos de modificadores que existen, y que suelen ser aplicados por hechizos a entidades.
- **ValTypeCharacter**: enumera los tipos de clases que existen.
- **ValPlayerSpells**: enumera los huecos de los hechizos a los que el jugador puede asignar un hechizo o habilidad.
- **ValTypePlayerArmour**: enumera las partes que un personaje puede tener. El último parámetro indica el total de partes que hay, es decir 7.
- **ValTypeAttributesArmour**: enumera los distintos atributos que puede tener un objeto del tipo armadura.
- **ValTypeAttributesEntity**: enumera los distintos atributos que tiene una entidad.

La siguiente clase es **ValConstants**, y únicamente se encarga de almacenar valores cuantitativos y cualitativos que definen algunos aspectos del juego, como puede ser: el nivel máximo del videojuego, el número máximo de huecos que tiene la mochila, etc. De este modo, en versiones futuras sólo será necesario modificar el valor en dicha clase, como por ejemplo si se quiere aumentar el nivel máximo. La clase está definida en el siguiente UML (ver Ilustración 53), y se pueden hacer varias distinciones en las constantes:

- Constantes que definen etiquetas.
- Constantes que definen layers³⁸.
- Constantes que definen aspectos cuantitativos del videojuego.

³⁸ Son valores que tienen los objetos en Unity y son utilizados para discriminar colisiones. Su traducción al español sería capa, pero puede llevar a confusión.

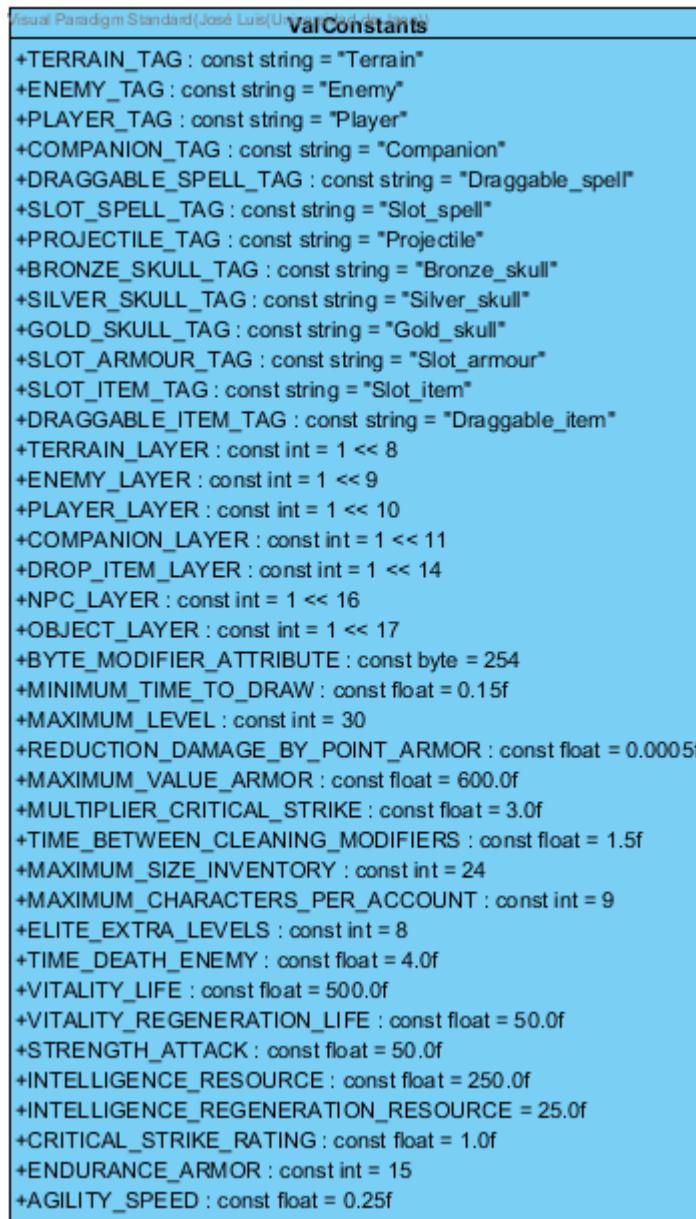


Ilustración 53. UML de la clase ValConstants que contiene las constantes de Valhalla

La tercera clase es **ValGame**, que se encuentra junto con ValConstants y todas las clases del tipo enum en el fichero ValStaticInformation. Implementa el patrón de diseño Singleton, y mantiene referencias a todas clases que únicamente deben tener una única instancia. Viene definido en el siguiente UML (ver Ilustración 54), y estas clases son:

- **ValNetworkManager**: encargada del sistema multijugador.
- **ValAuthenticationManager**: clase encargada de la autentificación de PlayFab.

- **ValDatabaseManager:** encargada de extraer información de la base de datos remota en PlayFab y actualizarla cuando sea necesario.
- **ValDatabaseItemsManager:** extrae información de una base de datos local (tabla de objetos disponibles en el videojuego), y proporciona información de éstos a través de unos métodos. Esta información nunca se actualiza, es estática.
- **ValInfoPlayerManager:** guarda información localmente del personaje elegido, y actualiza su información en la base de datos remota a través de ValDatabaseManager.
- **ValGameManager:** clase maestra de juego, única por nivel.
- **ValPlayer:** jugador local.

Para el caso concreto del jugador local, se mantienen unas suscripciones (patrón observador) de los GameObject que necesitan la información de éste, notificando a todos ellos una nueva referencia cuando la haya, o cuando deje de haberla (recordemos que no sabemos el orden de ejecución de los scripts, y en el caso del jugador es indispensable). Para establecer éstas suscripciones hay dos métodos:

- **SubscribeReferencePlayer:** método a través del cual se realiza una suscripción.
- **RemoveSubscriptionReferencePlayer:** método a través del cual se elimina una suscripción.

Los demás métodos que proporcionan son simples mutadores y observadores para acceder a la única referencia de las clases mencionadas con anterioridad.



Ilustración 54. UML clase ValGame

A continuación, está la clase **VallInfoPlayerManager**, encargada de almacenar información relativa al personaje que ha cargado el jugador en el menú principal. Además, realiza actualizaciones de la información del personaje a la base de datos remota a través de una serie de métodos. A la hora de realizar este tipo de actualizaciones, se debe evitar en la medida de lo posible actualizaciones innecesarias, y que pongan en riesgo la integridad del sistema. Para ello se hace lo siguiente:

- Cuando le llega una petición de actualización, modifica el personaje localmente, y manda una petición de actualización al servidor.
- Si llegan nuevas peticiones de actualización y hay una actualización en curso, se dejan en espera y se vuelve a realizar una actualización cuando el servidor haya modificado la información correctamente.

- Si no hay ninguna actualización pendiente y el servidor no actualiza debidamente la información, se vuelve a mandar una petición de actualización con la información local.

Para poder realizar esto, se añaden dos atributos a la clase del tipo booleano para indicar si hay una actualización pendiente o hay una actualización en curso y además, son estáticos para evitar la concurrencia y tener un acceso global (exclusión mutua).

Esta clase hereda de MonoBehaviour (es un script), y se asocia como componente a un GameObject de la escena, con la particularidad de que nunca es eliminado al cambiar de escena. Cada uno de los métodos permite la actualización de una información concreta del personaje, y están definidos en el siguiente UML (ver Ilustración 55).

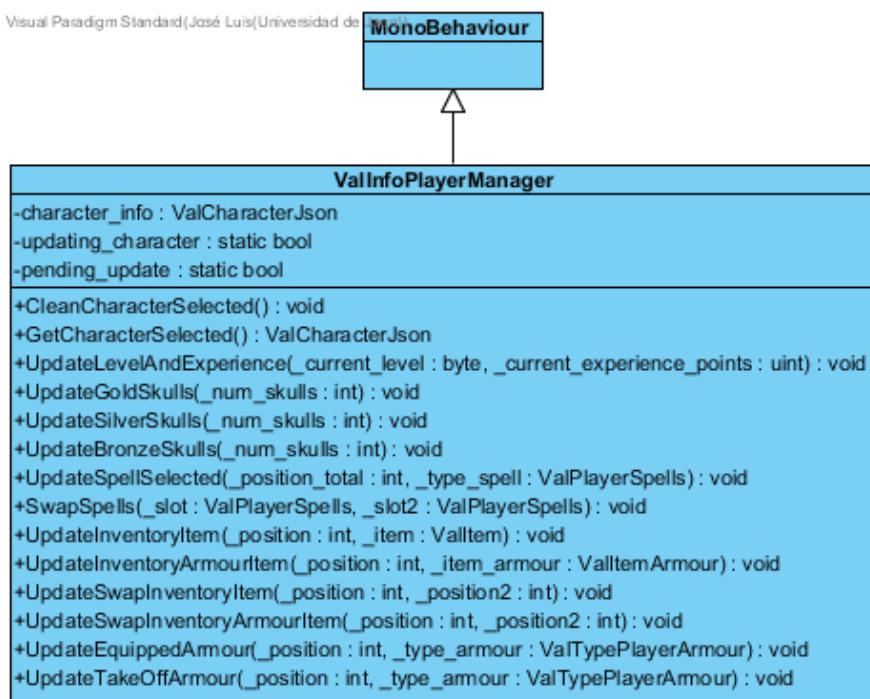


Ilustración 55. UML de la clase ValInfoPlayerManager

En penúltimo lugar, está la clase **ValGameManager**, que se encarga de gestionar cada uno de los niveles. Al igual que la clase anterior, va asociada a un GameObject y tiene una única instancia, con la diferencia de que en cada nivel se crea una instancia nueva y se elimina la anterior.

En cada nivel, el maestro de juego coloca a los jugadores en una posición específica. En los niveles con componentes procedurales se encarga de su ejecución,

y también proporciona experiencia. Todo esto sucede cuando se crea por primera vez y siempre y cuando sea el cliente servidor.

Su UML es el siguiente (ver Ilustración 56).

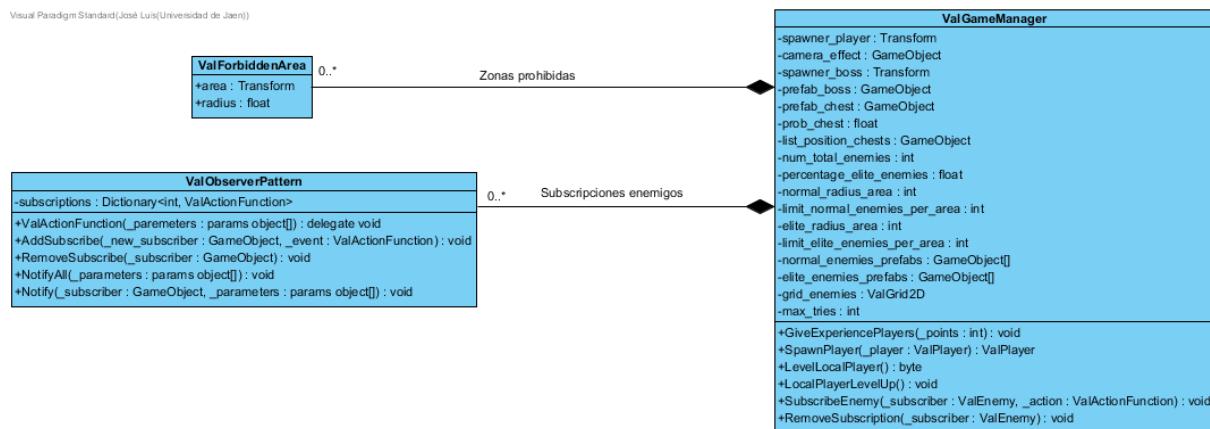


Ilustración 56. UML de la clase ValGameManager

Los atributos son:

- **spawner_player**: zona de comienzo de los jugadores.
- **camera_effect**: efecto de la cámara en el nivel actual.
- Relacionados con el jefe final:
 - **spawner_boss**: zona de colocación del jefe final.
 - **prefab_boss**: modelo del jefe final del nivel.
- Relacionados con los cofres:
 - **prefab_chest**: modelo de los cofres.
 - **prob_chest**: probabilidad aparición de un cofre.
 - **list_position_chests**: lista de las posibles posiciones de los cofres.
- Relacionados con las instanciación de enemigos:
 - **num_total_enemies**: número total de enemigos normales que debe instanciar.
 - **percentage_elite_enemies**: porcentaje de que un enemigo sea élite.
 - Limitaciones para los enemigos normales:

- **normal_radius_area**: radio del área de la zona de influencia.
- **limit_normal_enemies_per_area**: límite de enemigos normales en su zona de influencia.
- Limitaciones para los enemigos élitres:
 - **elite_radius_area**: radio del área de la zona de influencia.
 - **limit_elite_enemies_per_area**: límite de enemigos normales en su zona de influencia.
- **normal_enemies_prefabs**: lista de modelos disponibles de enemigos del tipo normal.
- **elite_enemies_prefabs**: lista de modelos disponibles de enemigos del tipo élite.
- **grid_enemies**: estructura malla regular 2D auxiliar para instanciar a los enemigos.
- **max_tries**: máximo de intentos para poder instanciar a un enemigo.
- Lista de áreas restringidas (zonas donde no puede haber enemigos bajo ningún concepto).
- Subscripciones de los enemigos para avisar de un cambio en el nivel del jugador local.

Algunos de los métodos más importantes que tiene son:

- **GiveExperience**: proporciona puntos de experiencia a todos los jugadores de la partida. Esto sólo puede hacerlo si es el cliente que hace de servidor.
- **SpawnPlayer**: coloca al personaje del jugador en la zona de aparición.
- **LevelLocalPlayer**: proporciona el nivel del jugador local.
- **LocalPlayerLevelUp**: notifica a todos los enemigos subscriptos que el jugador local ha subido un nivel. Esto sólo puede hacerlo si es el cliente que hace de servidor.

- **SubscribeEnemy, RemoveSubscription:** métodos para suscribirse o eliminar una subscripción cuando el enemigo muere.

8.4.1. Paquete de clases relacionadas con entidades

En este paquete se encuentran las clases que se encargan de la lógica de los enemigos, los personajes que controlan los enemigos y los compañeros. Es decir, cualquier entidad que tenga unos atributos y pueda recibir y realizar daño. Las clases vienen definidas en el siguiente UML (ver Ilustración 57).

En primer lugar, está el sistema de clases que definen la lógica de una entidad. Como clase padre **ValEntity** recoge todos los atributos, y funcionalidades comunes a cualquier entidad. Los atributos son:

- **points_attack:** puntos de ataque que tiene la entidad.
- **armor:** puntos de armadura que tiene la entidad.
- **speed:** velocidad de movimiento que tiene la entidad.
- **critical_strike:** probabilidad de golpe crítico que tiene la entidad.
- **damage_text_color:** color del texto flotante cuando recibe daño.

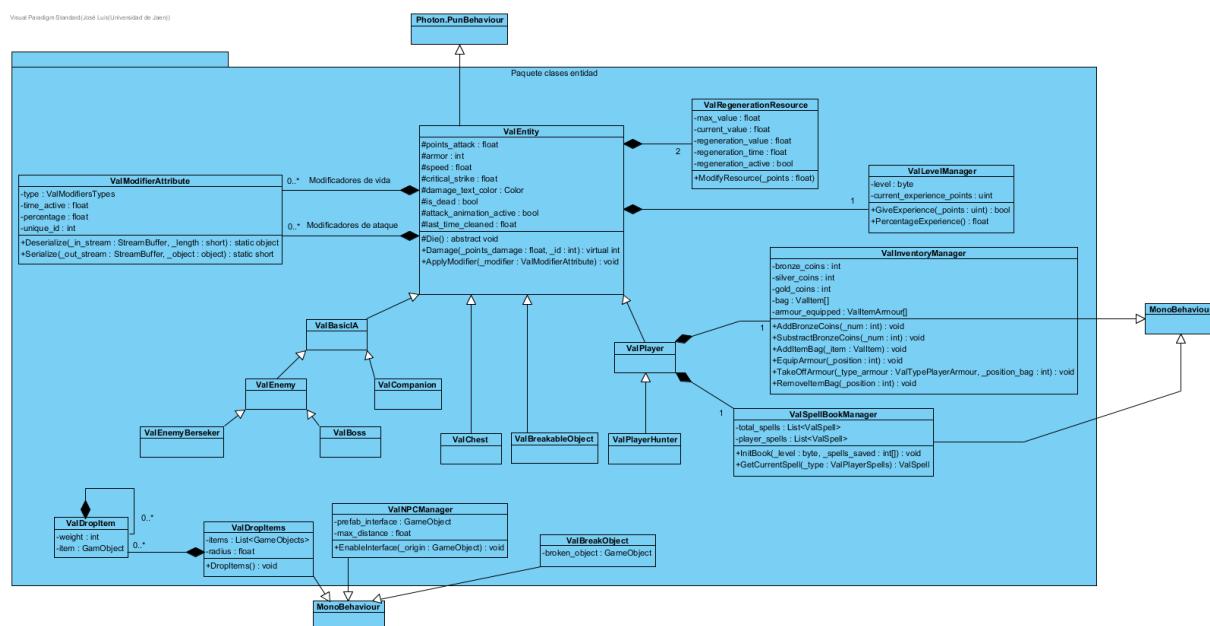


Ilustración 57. UML de las clases relacionadas con entidades

- **is_dead:** indica si la entidad está muerta.

- **attack_animation_active**: indica si hay una animación de ataque en curso.
- **last_time_cleaned**: indica en segundos, cuándo fue la última vez que se limpiaron los modificadores.

Otros atributos surgen de las relaciones con otras clases. Algunos atributos, como en el caso de la vida o el recurso, es preferible encapsularlos en una clase independiente, creada única y exclusivamente para almacenar el conjunto de atributos que lo definen, y métodos para manipularlos. Los atributos que surgen resultante de las relaciones son:

- Una **lista de modificadores del daño recibido**. Hay que recordar que en las mecánicas se definieron ataques que aumentaban el daño recibido durante un tiempo.
- Una **lista de modificadores de ataque**. Igual que el anterior, pero para un aumento del daño.
- Dos atributos regenerativos, la **vida** y el **recurso**.
- Atributo que se encarga de calcular la experiencia que necesita para subir de **nivel**, y almacena la experiencia y el nivel actual. Es decir, de manipular todo lo relacionado con el nivel y la experiencia.

El resto de clases hijas que derivan de la clase padre, son especializaciones de ésta. Las clases que derivan son las siguientes:

- **ValBasicIA**: clase padre de todas las entidades con inteligencia artificial.
- **ValEnemy**: clase que representa a cualquier enemigo con al menos un ataque básico. Aquí se podrá diferenciar a través de un atributo si es un enemigo élite o no, y si lo es tendrá un listado de posibles ataques entre los que se elegirán sólo algunos y 8 niveles más que un enemigo normal.
- **ValEnemyBerseker**: clase especializada de un enemigo genérico que añade un ataque más definido en las mecánicas.
- **ValBoss**: clase especializada de un enemigo genérico que almacena los ataques predefinidos de cada jefe final y su inteligencia artificial correspondiente.

- **ValPlayer:** segunda bifurcación de la clase principal que representa cualquier personaje controlado por el jugador. Permitirá el lanzamiento de hechizos y dos atributos fundamentales que surgen de dos relaciones:
 - Atributo encargado de la lógica del **inventario** y los objetos equipados. Almacena datos y permite su manipulación.
 - Atributo encargado de la lógica del **libro de hechizos**. Permite lo mismo que el anterior.
- **ValPlayerHunter:** especialización del personaje cazador.
- **ValChest:** representa una entidad cofre que al recibir daño se abre y deja caer una serie de objetos.
- **ValBreakableObject:** representa una entidad que al recibir daño y morir se rompe en trozos y salen disparados.

A diferencia del resto de scripts, la clase ValEntity deriva de una clase similar a MonoBehaviour, pero específica de la tecnología Photon. La herencia de esta clase es esencial para los elementos en red.

Las clases que representan un atributo en la clase ValEntity y no heredan de ninguna otra clase, son las siguientes:

- **ValRegenerationResource:** representa cualquier atributo que sea capaz de regenerarse. Contiene a su vez los siguientes atributos:
 - **max_value:** valor máximo.
 - **current_value:** valor actual del recurso.
 - **regeneration_value:** puntos de regeneración del recurso.
 - **regeneration_time:** cada cuanto tiempo regenera (segundos).
 - **regeneration_active:** indica si la regeneración activa. Atributo diseñado de cara al futuro para posibles hechizos o efectos que anulen la regeneración durante un periodo de tiempo.
- **ValLevelManager:** clase encargada de la lógica del sistema de niveles. Tiene los siguientes atributos:

- **level:** nivel actual
- **current_experience_points:** puntos de experiencia obtenidos actualmente.
- **ValModifierAttribute:** representa un modificador en la entidad. Tiene los siguientes atributos:
 - **type:** tipo de modificador definido por el tipo enum ValModifiersTypes.
 - **time_active:** tiempo que permanece activo en segundos.
 - **percentage:** porcentaje de bonificación.
 - **unique_id:** identificador único de la entidad que aplicó dicho modificador. Sólo podrá mantener un modificador activo a la vez cada entidad.

Por último, cabe destacar de la clase padre los métodos más importantes que contiene: el método abstracto **Die**, que deben definir todas las clases hijas para implementar una acción al morir. El método virtual **Damage**, que implementa la lógica cuando se recibe daño, aplicando por ejemplo, la armadura que tiene y reduciendo por tanto, el daño; las clases hijas podrán redefinir el método si necesitan algo más específico. Y el método **ApplyModifier** que aplica un modificador a la entidad.

En la clase **ValPlayer** hay dos atributos en clases independientes que heredan de MonoBehaviour y por tanto, son scripts. Estas clases son:

- **ValInventoryManager:** clase encargada de almacenar los objetos del inventario y los objetos equipados, y por supuesto, operar sobre ellos. Tiene los siguientes atributos:
 - **bronze_coins:** lleva la cuenta de los abalorios de bronce obtenidos.
 - **silver_coins:** lleva la cuenta de los abalorios de plata obtenidos.
 - **gold_coins:** lleva la cuenta de los abalorios de oro obtenidos.
 - **bag:** vector con los objetos del inventario. Cada índice representa la posición en la mochila.

- **armour_equipped:** vector con los objetos equipados. Cada índice representa un hueco de armadura.
- **ValSpellBook:** clase que representa el libro de hechizos donde el usuario puede elegir qué hechizos usar, siempre y cuando tenga nivel suficiente. Tiene como atributos:
 - **total_spells:** todos los hechizos con los que cuenta el personaje.
 - **player_spells:** hechizos que tiene asignados actualmente el jugador para poder usarlos.

El resto de clases del paquete están relacionadas con las clases entidades pero no forman parte de su jerarquía de clases. Estas clases son tres: ValDropItems, ValBreakObject y ValNPCManager, y todas son scripts.

ValDropItems es un script que se asocia al GameObject, que por lo general tendrá también asociado un componente de entidad que deja caer objetos en base a un sistema de tablas anidadas que permite la clase **ValDropItem**. Tiene como atributos:

- **weight:** peso del objeto.
- **item:** objeto modelo que deja caer.
- Con la relación tendría a su vez otro atributo que representaría otra tabla anidada.

ValNPCManager es un script que se asocia a cualquier GameObject que representa un NPC. Tiene como atributos:

- **prefab_interface:** modelo de la interfaz que muestra al jugador cuando interacciona con él.
- **max_distance:** máxima distancia en la que el jugador puede interaccionar con él.

ValBreakObject es otro script que se asocia a un objeto que se desea que se rompa. De tal modo, que cuando el objeto entidad muere, este script se encarga de instanciar el mismo objeto dividido en fragmentos y los esparce.

8.4.2. Paquete de clases controladoras de movimiento

El siguiente paquete contiene clases que de un modo u otro, controlan el movimiento de un objeto en concreto en la escena. El siguiente UML muestra las clases contenidas en dicho paquete (ver Ilustración 58).

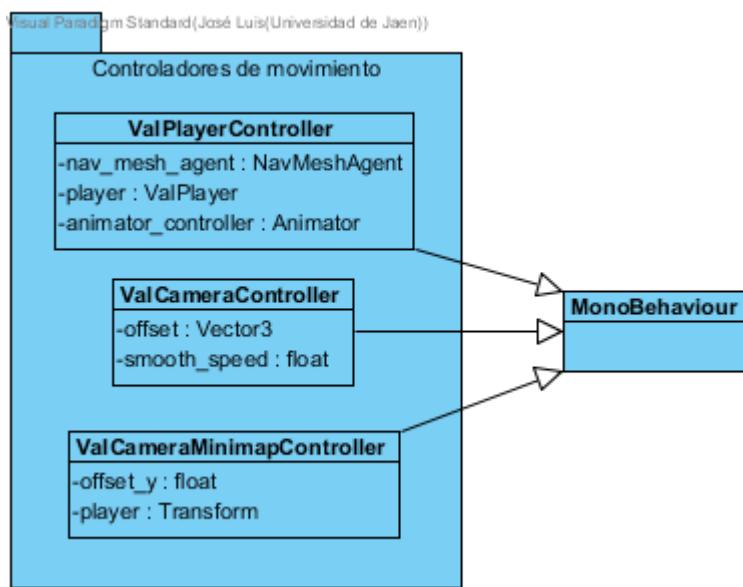


Ilustración 58. UML de las clases controladoras de movimiento

ValPlayerController fue una de las primeras clases diseñadas e implementadas, y se encarga de controlar las acciones para el movimiento de un jugador, liberando a la clase ValPlayer de este cometido. Este script debe ir asociado al mismo GameObject de algunas de las clases hijas de ValPlayer. Tiene los siguientes atributos:

- **nav_mesh_agent**: referencia al componente de malla de navegación para mover el personaje a nuevas posiciones.
- **player**: referencia del jugador que controla.
- **Animator**: referencia al componente animador, a través del cual, cambia entre los estados caminar o permanecer en el sitio para desencadenar una animación en bucle.

ValCameraController fue también una de las primeras clases, y tiene como tarea seguir al personaje manteniendo una distancia, y con un determinado retraso en la posición de la Y para suavizar y evitar movimientos bruscos en un posible relieve irregular del terreno. Tiene como atributos:

- **offset**: vector 3D con las distancia del jugador en las tres coordenadas. Realmente ésta no será la posición de la cámara con respecto al personaje, se colocará siempre en la dirección contraria al forward de éste (sólo en la primera posición, en el resto sólo sigue al personaje).
 - **smooth_speed**: factor real que es utilizado para suavizar el movimiento en la Y a través de interpolaciones.

Por último, **ValCameraMinimapController** se encarga del movimiento de la cámara secundaria utilizada para el mini mapa. Está situada siempre justo encima del personaje con un offset en la Y. Los atributos son:

- **offset_y**: distancia que mantiene con el personaje en el eje Y.
 - **player**: referencia al jugador para saber en todo momento dónde está.

8.4.3. Paquete de clases relacionadas con habilidades o hechizos

Otro de los paquetes es de las clases relacionadas con los hechizos o habilidades de los personajes, enemigos y compañeros. El UML es el siguiente (ver Ilustración 59).

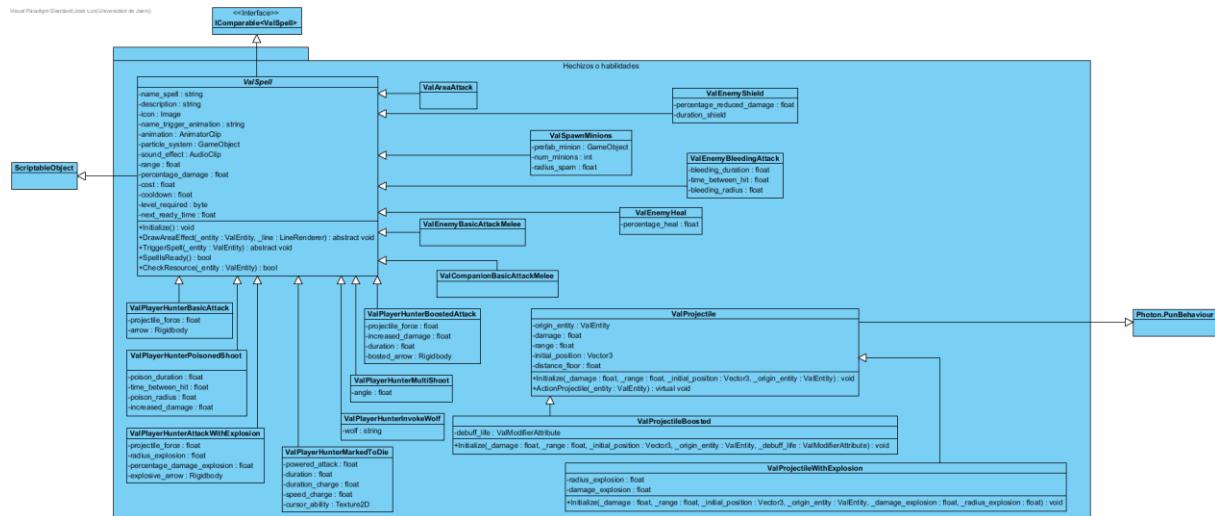


Ilustración 59. UML de las clases relacionadas con los hechizos o habilidades

La clase padre principal es **ValSpell**, que hereda directamente de ScriptableObject y de IComparable. La primera le proporciona las mismas ventajas que un script, pero sin necesidad de estar asociado a un GameObject. Las ventajas que proporciona son:

- Poder gestionar cada habilidad como cualquier otro recurso en el sistema de directorios del proyecto, proporcionándole los valores personalizados a través del editor.
- Se puede instanciar igual que cualquier otro recurso.
- Se pueden crear diferentes recursos del mismo tipo para entidades diferentes. Por ejemplo un hechizo de curación puede servir tanto para un enemigo como para un personaje, sólo serán diferentes recursos dentro del árbol de directorios, y tendrán diferentes valores.
- Se pueden asociar a su vez a otros scripts a través del editor, proporcionando fácilmente a través de una lista los hechizos que tiene disponibles una entidad (ver Ilustración 60).

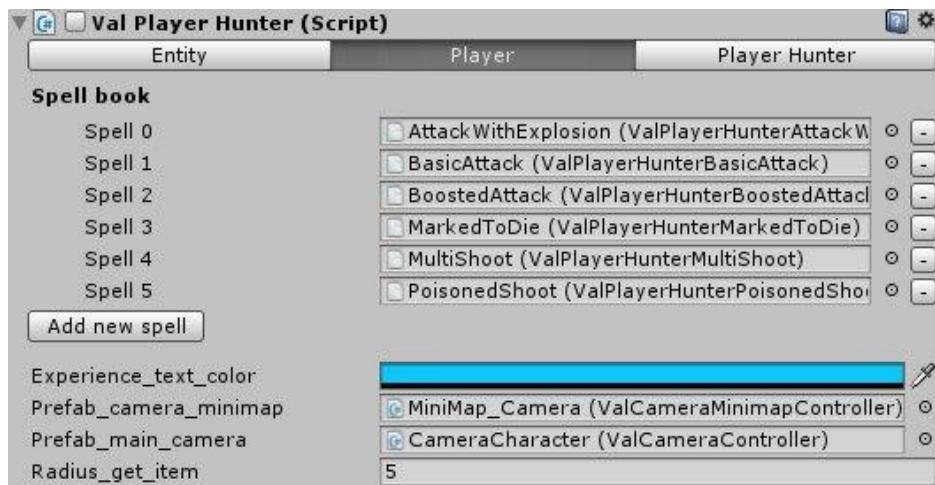


Ilustración 60. Lista de hechizos del componente ValPlayerHunter

La segunda clase sirve para ordenar hechizos dentro de una colección de C-Shard. En este caso, interesa ordenar los hechizos por requisitos de nivel para ir desbloqueándolos en el libro de hechizos conforme el jugador suba de nivel.

Los atributos de la clase:

- **name_spell:** cadena de caracteres con el nombre del hechizo. Será utilizado en un futuro para mostrar información de éste en la interfaz.
- **description:** descripción de la funcionalidad del hechizo. Será utilizado en un futuro para mostrar información de éste en la interfaz.
- **icon:** imagen que representa en la interfaz.

- **name_trigger:** nombre del trigger para activar la animación del hechizo.
- **animation:** animación del hechizo.
- **particle_system:** sistema de partículas para los posibles efectos visuales.
- **sound_effect:** sonido que provoca el hechizo.
- **range:** rango de acción.
- **percentage_damage:** porcentaje de daño.
- **cost:** coste de recurso.
- **cooldown:** tiempo de reutilización (tiempo necesario para volver a lanzarlo).
- **level_required:** nivel requerido para poder usarlo.
- **next_ready_time:** indica cuándo volverá a estar disponible. Depende del tiempo de reutilización.

Los métodos más importantes son:

- **Initialize:** inicializa los recursos.
- **DrawAreaEffect:** método abstracto que deben implementar las clases hijas. Es usado por hechizos de personajes, y se usa para dibujar en el terreno el área de acción o la trayectoria de los hechizos.
- **TriggerSpell:** método abstracto que deben implementar las clases hijas. Define una acción para cada hechizo.
- **SpellIsReady:** indica si el hechizo está disponible para ser lanzado.
- **CheckResource:** comprueba si la entidad tiene recurso suficiente para lanzarlo.

En primer lugar, están las clases hijas que representan hechizos del cazador definidos en las mecánicas. Cada una de ellas tiene atributos especializados y una implementación concreta para los diferentes casos. Las clases son:

- **ValPlayerHunterBasicAttack:** se corresponde con el ataque básico de un cazador. Sus atributos son:

- Projectile_force: fuerza del proyectil.
- Arrow: modelo de la flecha.
- **ValPlayerHunterPoisonedShoot:** se corresponde con el ataque envenenado de un cazador. Sus atributos son:
 - poison_duration: duración del veneno.
 - time_between_hit: tiempo que transcurre cada vez que recibe daño (segundos).
 - poison_radius: radio de acción.
 - increased_damage: daño incrementado que produce en la entidad enemiga.
- **ValPlayerHunterAttackWithExplosion:** se corresponde con el ataque explosivo del cazador. Sus atributos son:
 - projectile_force: fuerza del proyectil.
 - radius_explosion: radio de la explosión.
 - percentage_damage: porcentaje de daño que realiza.
 - explosive_arrow: modelo de la flecha explosiva.
- **ValPlayerHunterMarkedToDie:** se corresponde con el hechizo marcado para morir del cazador. Sus atributos son:
 - powered_attack: porcentaje de potenciamiento de los ataques del lobo.
 - duration: duración del potenciador de ataque (segundos).
 - duration_charge: duración del aumento de velocidad del lobo (segundos).
 - speed_charge: nueva velocidad del lobo.
 - cursor_ability: nuevo cursor que modifica el cursor por defecto del ratón cuando se muestra el radio de acción del hechizo.
- **ValPlayerHunterInvokeWolf:** hechizo encargado de invocar al lobo. Su atributo es:

- wolf: dirección relativa del modelo lobo.
- **ValPlayerHunterMultiShoot:** corresponde con el hechizo multidisparo del personaje cazador. Su atributo es:
 - angle: ángulo de apertura del ataque.
- **ValPlayerHunterBoostedAttack:** corresponde con el ataque potenciado del personaje cazador. Sus atributos son:
 - projectile_force: fuerza del proyectil.
 - increased_damage: porcentaje de incremento del daño sobre el enemigo.
 - duration: duración del incremento (segundos).
 - boosted_arrow: modelo de la flecha potenciada.

En segundo lugar, están los hechizos o habilidades de la inteligencia artificial. Es decir, los compañeros y los enemigos. Los hechizos son los siguientes:

- **ValCompanionBasicAttackMelee:** ataque básico cuerpo a cuerpo de un compañero.
- **ValEnemyBasicAttackMelee:** ataque básico cuerpo a cuerpo de un enemigo.
- **ValAreaAttack:** ataque en área de un enemigo.
- **ValSpawnMinions:** hechizo que invoca otros enemigos. Sus atributos son:
 - prefab_minion: modelo del enemigo que invoca.
 - num_minions: número de enemigos totales que invoca.
 - radius_spam: radio de la posición de invocación de éstos.
- **ValEnemyHeal:** hechizo que cura a un enemigo. Su atributo es:
 - percentage_heal: porcentaje de curación de su vida total.
- **ValEnemyShield:** hechizo que protege durante un tiempo a un enemigo. Sus atributos son:

- percentage_reduced_damage: porcentaje de daño reducido recibido.
- duration_shield: duración del escudo.
- **ValEnemyBleedingAttack:** hechizo que provoca un sangrado a todos los personajes y compañeros de éstos (daño en el tiempo). Sus atributos son:
 - bleeding_duration: duración del sangrado (segundos).
 - time_between_hit: tiempo entre cada sangrado.
 - bleeding_radius: radio de acción.

Por último, están las clases que controlan la acción de cada proyectil. La primera clase es **ValProjectile**, es la clase padre y de la que derivan las demás. Sus atributos son:

- **origin_entity:** referencia a la entidad que lanzó el proyectil. Servirá para eliminar el proyectil si se aleja demasiado de la entidad.
- **damage:** daño que realiza el proyectil al impactar con una entidad.
- **range:** distancia máxima que puede viajar.
- **distance_floor:** distancia mínima que mantiene con el suelo.

Los métodos que tiene son:

- **Initialize:** inicializa los atributos del proyectil.
- **ActionProjectile:** define una acción cuando el proyectil colisiona con una entidad. Al ser virtual, puede ser especializado por una clase hija.

Las clases especialistas del proyectil son: **ValProjectileBoosted**, que representa el proyectil con incremento de daño, y **ValProjectileWithExplosion**, que representa el proyectil con daño explosivo. En el primer caso, especializa la acción añadiendo un modificador de daño aumentado en la entidad, y en el segundo caso, lo especializa añadiendo una explosión.

8.4.4. Paquete de clases relacionadas con los objetos del juego

El siguiente paquete es el de las clases relacionadas con los objetos que hay dentro del juego. El UML de estas clases es el siguiente (ver Ilustración 61).

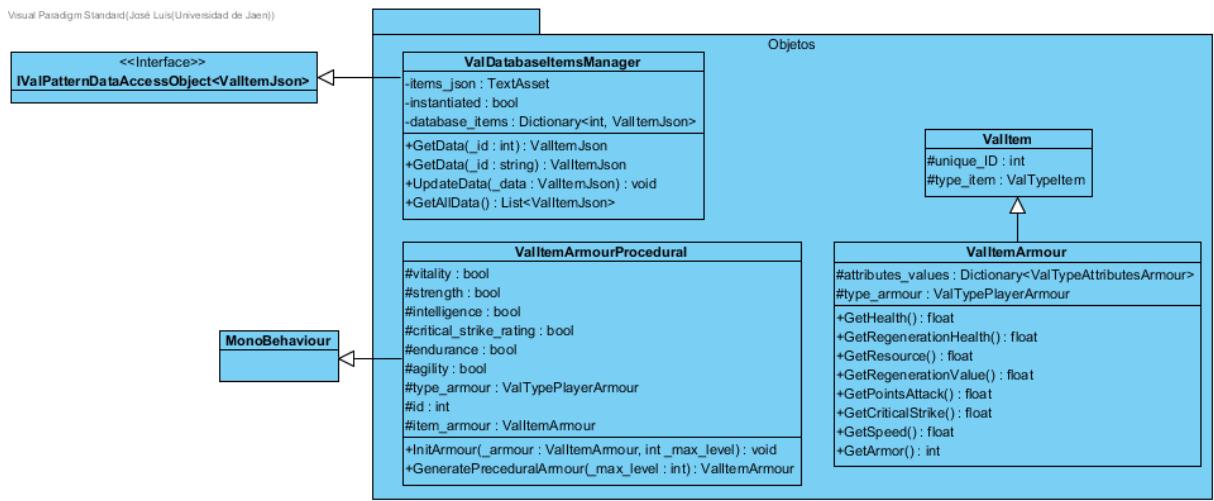


Ilustración 61. UML de las clases relacionadas con los objetos

En primer lugar, está la clase **VallItem**, de la cual heredan todos los tipos de objetos. Esta clase se ha pensado para añadir futuros objetos, y sus atributos son: **unique_ID**, como identificador único del objeto, y el **tipo de objeto** del que se trata. La única clase hija que se encuentra en este momento es **VallItemArmour**, que hereda de la anterior y mantiene los valores generados proceduralmente de dicho objeto.

En segundo lugar, está la clase **VallItemArmourProcedural**, que se asocia a los objetos del tipo armadura, la cual genera los valores de los atributos de ésta en el momento en que un enemigo la deja caer. Sus atributos son booleanos indicando qué atributos tiene y además, el tipo de armadura del que se trata.

Por último, está la clase **ValDatabaseItemsManager** cuya función es cargar la información en formato JSON de los objetos de los que dispone el juego en un fichero local. De este modo, mantiene una base de datos local que puede ser consultada por las demás clases. Implementa el patrón DAO.

8.4.5. Paquete de clases controladoras de la interfaz

El paquete de clases que controlan la interfaz se dividen en:

- Clases que manipulan la interfaz del menú principal.
- Clases que manipulan la interfaz dentro del juego.

En primer lugar, están las clases que se encargan de manipular el menú principal, y su UML es el siguiente (ver Ilustración 62). Todas y cada una de ellas son scripts.

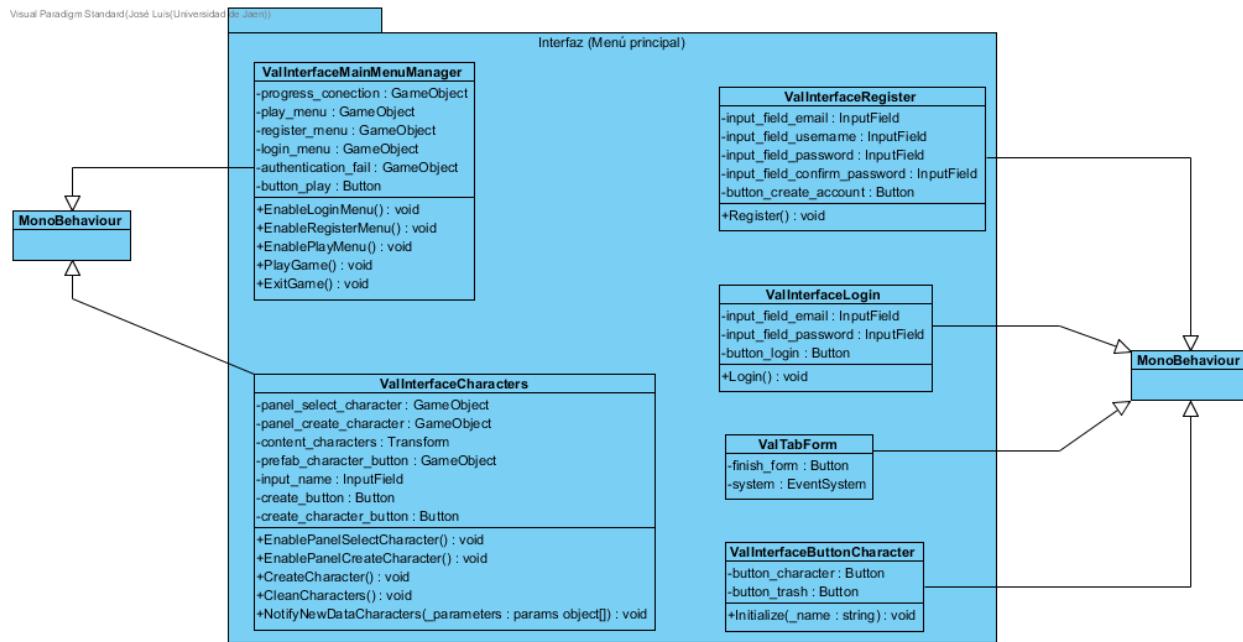


Ilustración 62. UML de clases relacionadas con la interfaz del menú principal

Las clases son las siguientes:

- **ValInterfaceMainMenuManager**: controla la interfaz del menú principal e implementa el patrón Singleton. A diferencia de otras clases, no está dentro de la clase ValGame, ya que no necesita tanta concurrencia y sólo estará en el menú principal. Sus atributos son:
 - progress_connection: referencia al panel de progreso de conexión con la sala del servidor Photon.
 - play_menu: referencia al botón que activa el panel de jugar partida.
 - register_menu: referencia al botón que activa el panel de registro de una nueva cuenta.
 - login_menu: referencia al botón que activa el panel de inicio de sesión.
 - authentication_fail: panel que se usa cuando hay un fallo en la autenticación.
 - button_play: referencia al botón que comienza una partida.

- **VallInterfaceCharacters:** controla el panel que se encarga de la creación de personajes y de la selección de éstos para poder jugar. Sus atributos son:
 - panel_select_character: referencia al panel para seleccionar los personajes disponibles.
 - panel_create_character: referencia al panel para crear personajes.
 - content_characters: referencia al panel donde están listados los botones que representan cada uno de los personajes disponibles en una cuenta.
 - input_name: referencia elemento encargado de recoger el nombre del personaje que se va a crear.
 - create_button: referencia al botón que activa el panel de creación de personajes.
 - create_character_button: referencia al botón que crea un personaje.
- **VallInterfaceRegister:** se encarga de la lógica en el proceso de registro del jugador. Sus atributos son:
 - input_field_email: referencia al elemento correo dentro del formulario.
 - input_field_username: referencia al elemento nombre de usuario dentro del formulario.
 - input_field_password: referencia al elemento contraseña dentro del formulario.
 - input_field_confirm_password: referencia al elemento confirmar contraseña dentro del formulario.
 - button_create_account: referencia al botón que crea la cuenta si está todo el formulario completo y sin ningún tipo de error.
- **VallInterfaceLogin:** se encarga de la lógica en el proceso de inicio de sesión del jugador. Sus atributos son:

- input_field_email: referencia al elemento correo dentro del formulario.
 - input_field_password: referencia al elemento contraseña dentro del formulario.
 - button_login: referencia al botón para iniciar sesión.
- **ValTabForm:** permite cambiar entre los campos del formulario a través de la tecla tabuladora del teclado, y activar el botón que envía el formulario pulsando el botón intro del teclado.
 - **ValInterfaceButtonCharacter:** representa el botón de un personaje. Se divide a su vez en dos botones:
 - button_character: referencia al botón con el nombre del jugador y que se usa para seleccionar el personaje con el que se desea jugar.
 - button_trash: botón que elimina el personaje.

Por otro lado, están las clases que se encargan de la lógica de la interfaz dentro del videojuego, y están en el siguiente UML (ver Ilustración 63). Para simplificar la explicación no se va a comentar cada uno de los atributos, ya que sólo son meras referencias a todos los elementos de la interfaz y con el propio nombre se puede saber qué almacenan dichos elementos.

La clase principal es **ValInterfacePlayerManager**, y es la encargada de la interfaz del usuario. Ésta tiene una gran cantidad de atributos que mantienen referencias a los elementos de la clase, como puede ser la vida, el recurso, los puntos de experiencia, etc. Tiene como función actualizar todos los componentes debidamente en el momento que se produce un cambio, y dar acceso al libro de hechizos, menú e inventario.

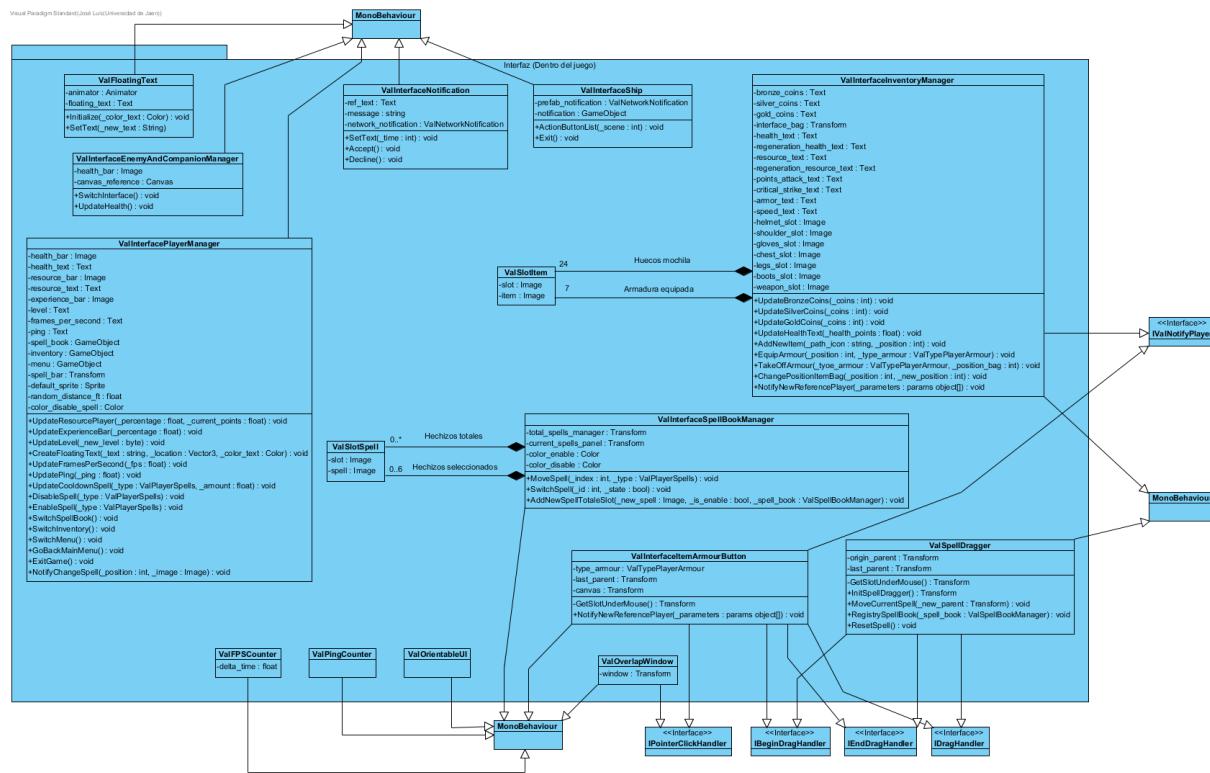


Ilustración 63. UML de clases relacionadas con la interfaz dentro del juego

Otra de las clases fundamentales de este paquete son **ValInterfaceSpellBookManager** y **ValInterfaceInventoryManager**. La primera de ellas se encarga la interfaz del libro de hechizos, realizando actualizaciones visuales y comunicándose con la lógica del personaje para notificar dichos cambios. La segunda se encarga de la interfaz del inventario, realizando actualizaciones en la mochila, los objetos equipados y actualizando los atributos del jugador en el panel izquierdo. Además, al igual que el libro de hechizos, se comunica con el personaje para notificar cambios.

El resto de clases son utilizadas para componentes muy concretas, y son las siguientes:

- **ValFloatingText**: se encarga de los textos flotantes para el daño recibido, daño causado, puntos de experiencia obtenidos, y la subida de un nivel.
- **ValInterfaceEnemyAndCompanionManager**: se encarga de mostrar la vida de los compañeros y enemigos y actualizarla.
- **ValInterfaceNotification**: se encarga de las notificaciones para cambiar de nivel.

- **ValInterfaceShip**: se encarga ofrecer al usuario la lista de niveles disponibles y seleccionar uno de ellos, generando una notificación a todos los jugadores.
- **ValFPSCounter**: controla los frames por segundo.
- **ValPingCounter**: controla el retraso sufrido en la conexión con el servidor Photon.
- **ValOrientableUI**: orienta la vida de enemigos y compañeros a la cámara del jugador.
- **ValInterfaceItemArmourButton**: lógica del botón que representa un objeto armadura. Hereda de interfaces para la notificación de eventos con el ratón.
- **ValSpellDragger**: lógica del botón que representa un hechizo. Hereda de interfaces para la notificación de eventos con el ratón.

8.4.6. Paquete de clases de estructuras personalizadas

Aquí se encuentran las clases que representan estructuras construidas por mí y que he necesitado en algún momento en el proyecto. En este caso, sólo está la malla regular en 2D usada para la instanciación de enemigos en los sistemas de niveles. En el siguiente UML se puede ver la totalidad de las clases del paquete (ver Ilustración 64).

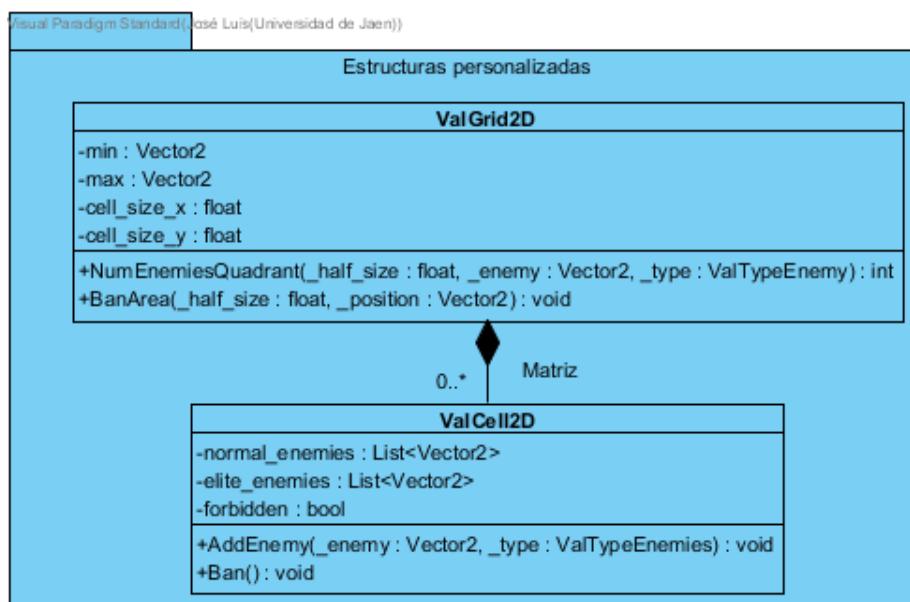


Ilustración 64. UML de las clases de estructuras personalizadas

En primer lugar, está la clase auxiliar ValCell2D que representa una casilla en una malla regular 2D, una porción cuadrada de área. En cada casilla se almacena: una lista de enemigos normales con sus posiciones en dicha área, otra lista para los enemigos élites, y para acabar, un booleano que indica si la zona es prohibida o no. También dispone de un método para añadir enemigos al área (indicando el tipo), y otro para prohibir esa área, entre otros métodos.

Por último, está la clase ValGrid2D que representa una malla regular en 2D que es construida a partir de un punto 2D con los valores mínimos del área total, y otro punto 2D con los valores máximos y además, el número de divisiones a lo ancho y a lo alto. Los atributos son los siguientes:

- **min**: punto 2D con los valores mínimos del área.
- **max**: punto 2D con los valores máximos del área.
- **cell_size_x**: tamaño de cada casilla en su coordenada X.
- **cell_size_y**: tamaño de cada casilla en su coordenada Y.
- **Matriz de casillas** resultante de la relación, a través de la cual se tiene acceso a todas las porciones de área.

Por otro lado, los métodos más importantes son:

- **NumEnemiesQuadrant**: obtiene el número de enemigos de un tipo en un área cuadrada.
- **BanArea**: prohíbe las porciones de área alrededor de un área cuadrada en una posición dada.

8.4.7. Paquete de clases para el editor de Unity

Este paquete incluye clases con las que he redefinido la interfaz del editor del motor gráfico Unity de las clases del sistema de entidades. Dada la gran cantidad de atributos que tienen, y la gran confusión que suponía la mezcla de todos los atributos, sin ningún tipo de distinción, decidí cambiar y personalizar la visualización y modificación de los atributos a través del editor de Unity. Estas clases aunque no tienen un papel dentro del videojuego, son importantes, ya que ayudan y facilitan en gran medida la depuración del proyecto. A continuación, se muestra en el UML las

clases que forman el paquete, y sus herencias (ver Ilustración 65). Todas y cada una de ellas tienen el mismo nombre de la clase que modifican.

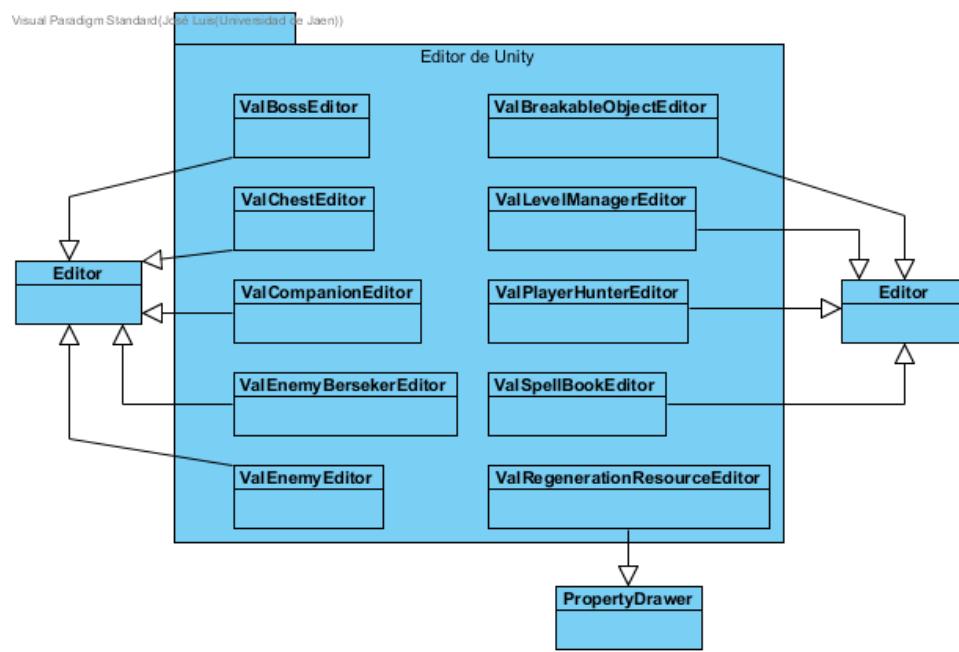


Ilustración 65. UML con las clases que personalizan el editor de Unity

Este es un ejemplo del cambio del editor en Unity para la clase ValPlayerHunter. Los atributos están divididos en tres pestañas, una para los atributos de cada clase (ver Ilustración 66).

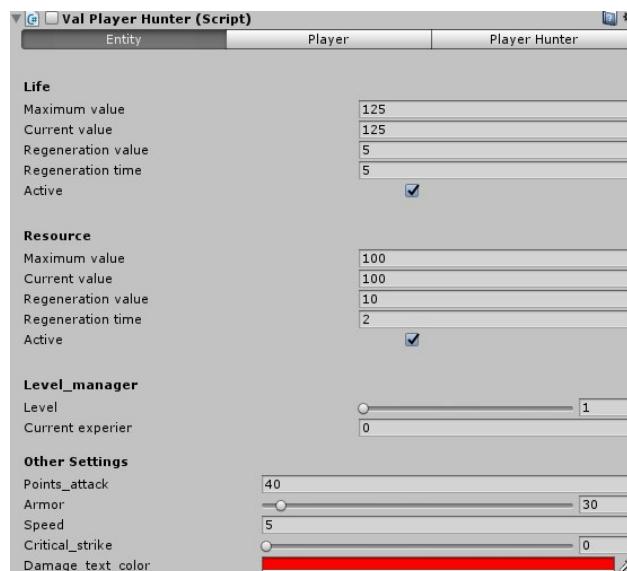


Ilustración 66. Editor personalizado para la clase ValPlayerHunter

8.4.8. Paquete de clases de geometría

Este paquete de clases está pensado para clases puramente geométricas. En este caso, es un paquete enfocado para posibles clases futuras, sólo hay una única clase con métodos estáticos para obtener geometría de segmentos y círculos, es decir, geometría en 2D. Esta clase está pensada concretamente para visualizar el área de acción o trayectoria de las habilidades del jugador en el suelo. El UML de este paquete es el siguiente (ver Ilustración 67).

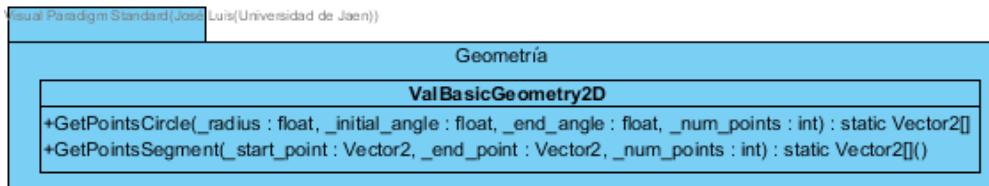


Ilustración 67. UML con la clase de geometría 2D

La clase consta de dos métodos:

- **GetPointsCircle**: devuelve un vector con la geometría 2D de una circunferencia (o simplemente un arco), la cantidad de geometría deberemos pasarla como parámetro, así como el ángulo inicial y final, dando la posibilidad de generar una circunferencia completa (0° , 360°) o una porción de éste.
- **GetPointsSegment**: al igual que la anterior, devuelve un vector con la geometría del segmento, y como parámetros necesita el punto inicial del segmento, el punto final y la cantidad de puntos.

8.4.9. Paquete de clases de utilidades JSON

Paquete compuesto por clases con utilidades para tratar información con formato JSON. Todas estas clases son serializables, lo cual permite convertir la instancia de la una clase en una cadena de caracteres con formato JSON para ser almacenada posteriormente en una base de datos o un simple fichero, y el proceso inverso, a través de la clase de utilidades JSON [44]. Las clases vienen definidas por el UML que viene a continuación (ver Ilustración 68).

Las clases se pueden subdividir en tres partes: clases JSON de propósito general, clases JSON para la tabla local de los objetos disponibles en el juego, y las

clases utilizadas para la base de datos remota que guarda información relativa a los personajes de un jugador.

Como clases de propósito general, están **ValJsonArray** y **ValWrapper**, utilizadas para pasar a formato JSON cualquier vector (templates), y viceversa.

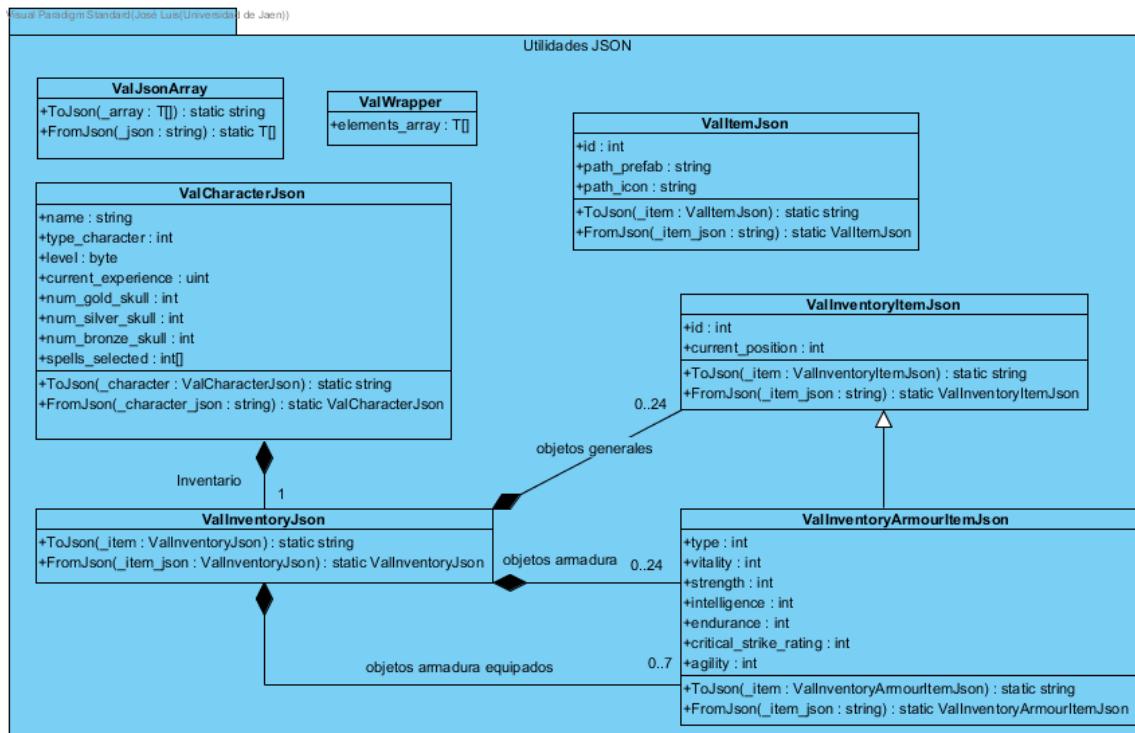


Ilustración 68. UML de clases con utilidades JSON

Por otro lado, para el caso de la tabla local de objetos disponibles en el juego, se hace uso de la clase **VallItemJson**, que almacena:

- **id**: identificador único del objeto.
- **path_prefab**: dirección relativa a la carpeta Resources del proyecto, donde se encuentra el modelo del objeto con sus componentes asociadas.
- **path_icon**: dirección relativa a la carpeta Resources del proyecto, para el icono del objeto en la interfaz.

Por último, están las clases encargadas de la información en formato JSON de la base de datos remota en PlayFab. Las clases son:

- **ValCharacterJson**: información necesaria para recuperar un personaje de la base de datos.

- **VallInventoryJson:** información necesaria para recuperar el inventario de un personaje de la base de datos. Es un atributo del personaje.
- **VallInventoryItemJson:** información necesaria para recuperar un objeto genérico del inventario de un personaje. Clase creada para objetos futuros, actualmente sólo hay objetos armadura.
- **VallInventoryArmourJson:** información necesaria para recuperar un objeto del tipo armadura del inventario de un personaje.

Todas y cada una de las clases anteriores tienen dos métodos estáticos: **ToJson** para pasar a formato JSON la información de una instancia, o bien **FromJson** para pasar de una cadena de caracteres a una instancia de clase.

8.4.10. Paquete de clases relacionadas con el sistema en red

Por último, está el paquete que engloba a todas las componentes de red, ya sea del sistema multijugador, autenticación o incluso base datos. Cada uno de los componentes en red está dividido en clases. De este modo, se separa su funcionalidad en tareas más específicas y sencillas. También evita el acoplamiento en la medida de lo posible, con las tecnologías Photon y PlayFab, de tal forma que si en algún momento se decide cambiar por ejemplo, el sistema multijugador, sólo afectará a estas clases y no al resto. El UML es el que viene a continuación (ver Ilustración 69).

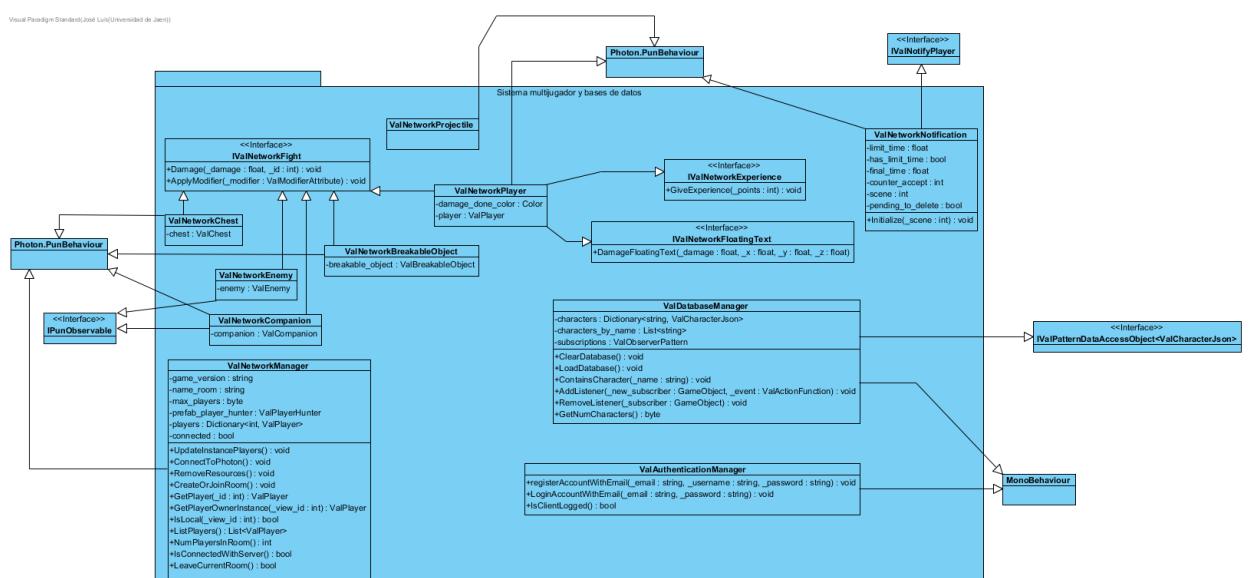


Ilustración 69. UML con las clases del sistema multijugador y la base de datos

Para empezar, están las clases que permiten el sistema multijugador. Como clase principal, está **ValNetworkManager**, que hereda de PunBehaviour [45], que es una clase similar a MonoBehaviour, pero con la diferencia que proporciona callbacks para el sistema online, tanto callbacks de conexión con Photon, como errores que se puedan producir.

Este script debe ser asociado en un GameObject de la primera escena, y no será eliminado al hacer un cambio de ésta. Tendrá una serie de tareas básicas, a modo resumido son:

- Conecta con Photon.
- Crea una sala si eres el primer jugador, o se une a la única sala disponible que hay (límite 4 jugadores).
- Recoge cualquier clase de error en la conexión (se han implementado el mayor número de casuísticas posibles):
 - Si se produce un error en la conexión, lleva al jugador al menú principal y le avisa de que no hay conexión a Internet. Si ya estaba en el menú, sólo le muestra el mensaje.
 - Si el jugador servidor se desconecta, los demás clientes salen de la partida y son dirigidos al menú principal (no hay migración del cliente).
- Proporciona una serie de métodos necesarios para el resto de clases.

El resto de scripts del sistema, son clases auxiliares que habilitan algunas componentes de los elementos que son parte del sistema multijugador, ya que algunos scripts sólo deben ejecutarse en el lado del cliente servidor. Por ejemplo, para el caso de las entidades el script especializado de la clase **ValEntity** sólo debe estar activo en el cliente servidor, o en el caso de los proyectiles, su script con la lógica de éste y el componente que le permite detectar colisiones. Además de esto, también algunos de ellos heredan de interfaces personalizadas que definen una serie de comportamientos a través de RPC (explicado en la sección 6). Estas interfaces son:

- **IValNetworkFight**: contiene los métodos modelo RPC para el sistema de combate, y es usado por entidades. Por ejemplo, en el caso de los enemigos, sólo tienen el script ValEnemy disponible en la parte

servidora, pero en los demás clientes es necesario comunicar el daño realizado, lo cual se realiza a través de estos métodos. Los métodos disponibles son los siguientes:

- Damage: método a través del cual se comunica daño realizado a dicha entidad.
- ApplyModifier: método con el que se comunica la aplicación de modificadores.
- **IValNetworkExperience**: encapsula los métodos modelo RPC para el sistema de experiencia, y es usado por entidades que representan jugadores. El método es:
 - GiveExperience: comunica nuevos puntos de experiencia.
- **IValNetworkFloatingText**: contiene los métodos modelo para los textos flotantes. El método es:
 - DamageFloatingText: comunica el texto que debe mostrar y su posición. Aunque el daño puede ser calculado en el lado del cliente, ya que el jugador es el que realiza el daño y el que lo recibe, el daño de entrada que recibe el enemigo será modificado por los modificadores y la armadura de la que disponga.

Las clases auxiliares de las que se ha hablado antes son las siguientes:

- **ValNetworkChest**: componente multijugador para el caso de los cofres.
- **ValNetworkEnemy**: componente multijugador para el caso de los enemigos. Además de activar componentes al inicio de su vida, se encarga de sincronizar la barra de vida a los enemigos duplicados en los clientes que no son servidores.
- **ValNetworkCompanion**: componente multijugador para el caso de los compañeros. Realiza la misma sincronización de vida que la clase anterior.
- **ValNetworkBreakableObject**: componente multijugador para el caso de los objetos rompibles.

- **ValNetworkProjectile:** componente multijugador para los proyectiles. En este elemento es importante no activar el componente que permite colisionar al proyectil, ya que si lo hace llamará indirectamente al callback de colisión en el script ValProjectile (que no estará activo, sólo en el servidor), y dará lugar a un fallo en el resto de clientes.
- **ValNetworkPlayer:** componente multijugador para cualquier clase de personaje que representa un jugador.
- **ValNetworkNotification:** componente multijugador para el caso de las notificaciones masivas.

En último lugar, están las clases necesarias para el sistema de autentificación y base de datos remota. Aunque ambas usen la misma tecnología PlayFab, es preferible diferenciar entre el sistema de autentificación y base de datos.

La primera de ellas es la clase **ValDatabaseManager**, encargada de extraer información de la base de datos y actualizarla debidamente a través de unos métodos que proporciona al resto de clases. Hereda de la interfaz para implementar el patrón Data Access Object, y de la clase MonoBehaviour. Sus atributos son:

- **characters:** diccionario de personajes del jugador que usa como clave el nombre del personaje y almacena toda su información en formato JSON.
- **characters_by_name:** lista de los nombres de todos los personajes.
- **subscriptions:** suscripciones para recibir actualizaciones de nuevos personajes.

La segunda clase es **ValAuthenticationManager**, encargada del proceso de autentificación en PlayFab a través de unas credenciales básicas, correo y contraseña. La clase permite:

- Iniciar sesión con unos credenciales (no permite iniciar sesión con servicios ajenos como Google o Facebook).
- Registrarse con una cuenta.
- Comprobar si el cliente ha iniciado sesión.

Como sistema autentificador tiene una carencia muy grave, y es que no tiene la oportunidad de poder cerrar sesión con una cuenta, debido a que la API de PlayFab no nos lo permite. Este problema será analizado con más profundidad en las conclusiones finales.

8.5. Comunicación entre clases

Para finalizar, se explican brevemente las comunicaciones más relevantes entre dichas clases. Las comunicaciones más importantes, y quizás más complejas, son entre la lógica que subyace al personaje de un jugador, y la interfaz de éste (ver Ilustración 70).



Ilustración 70. Comunicación entre las componentes lógicas y de la interfaz de un personaje

Tanto la lógica de un personaje, en cuanto a su inventario y libro de hechizos se refiere, como la interfaz de éste, se comunican de un modo bidireccional. La parte lógica notificará cambios en su estado, y la parte de la interfaz hará lo mismo. A modo de ejemplo, el libro de hechizos desbloquea hechizos conforme el jugador sube de nivel, y por tanto, debe comunicar a la interfaz estos cambios. También ocurre al contrario, cuando el jugador cambia de hechizos o modifica su posición en el libro, debe ser notificado al libro de hechizos lógico. De igual forma pasa con el inventario.

Este tipo de comunicaciones se podrían haber realizado haciendo uso del patrón observador, pero es innecesario, ya que sólo se comunican elementos 1 a 1.

Otra comunicación es entre el personaje y su interfaz. A diferencia de las anteriores, es unidireccional, solamente notifica la lógica del personaje su nuevo estado a modo informativo en la interfaz, como puede ser la vida o el recurso, pero el jugador no tiene manera de interactuar con la interfaz.

Por último, como se indicó en la parte de los patrones de diseño, existe una comunicación unidireccional entre la clase ValGameManager y los enemigos (sólo en

la parte del servidor, ya que ahí están instanciados los enemigos), haciendo uso del patrón Observador. Todos los enemigos se suscriben al comienzo de su vida a dicha clase para recibir cambios sobre el nivel del jugador local, adaptando su nivel al de éste. También por otro lado, se produce una comunicación entre la clase ValGame, y los objetos dependientes de la referencia del jugador local, y de igual modo que la comunicación anterior se produce en un único sentido, y se hace uso del patrón Observador.

9. Prototipo final Valhalla

Una vez se han realizado las investigaciones oportunas y se han hecho los diseños con los aspectos importantes de una aplicación, se realiza la implementación de ésta. El proyecto progresará hasta llegar la versión definitiva, el prototipo final.

A continuación se va a tratar cada uno de los componentes de los que está compuesta la versión final.

9.1. Modos de juego

En la versión final sólo se ha podido implementar el modo historia, dejando el modo carrera y core para posibles versiones futuras.

9.2. Escenas o niveles disponibles

El prototipo contiene un total de tres escenas:

- La escena con el menú principal.
- La escena con la zona neutral, también llamada Yggdrasil.
- La escena con el primer nivel, también llamado Bilskirnir.

En el primer nivel, se coloca una cámara principal en el terreno duplicado de la zona neutral, y se superpone la interfaz que representa el menú principal. La siguiente imagen es del escenario visto desde el editor (ver Ilustración 71).



Ilustración 71. Primera escena: Menú principal

También, se instancian por primera vez objetos de las clases:

- ValNetworkManager.
- ValAuthenticationManager.
- ValDatabaseManager.
- ValDatabaseItemsManager.
- VallInfoPlayerManager.

En el segundo escenario, está definida la zona neutral (ver Ilustración 72) que se especificó en las mecánicas, y donde se encuentran los futuros vendedores, que por ahora sólo hablan contigo, y el barquero que permite a los jugadores cambiar a otros niveles. En este nivel, la clase maestra de juego únicamente coloca a los jugadores en el punto de instantiación.



Ilustración 72. Escena de la zona neutral vista desde el modo editor. Los círculos verdes son NPCs con los cuales se puede hablar, y el círculo rojo es donde son colocados los jugadores inicialmente

En el tercer escenario, se utiliza el sistema de niveles procedurales sobre un terreno estático. En dicho nivel se encuentra el jefe final Thor. Los valores de la clase ValGameManager para este nivel vienen definidos por la siguiente imagen (ver Ilustración 73), y el nivel uno visto desde el editor (ver Ilustración 74).

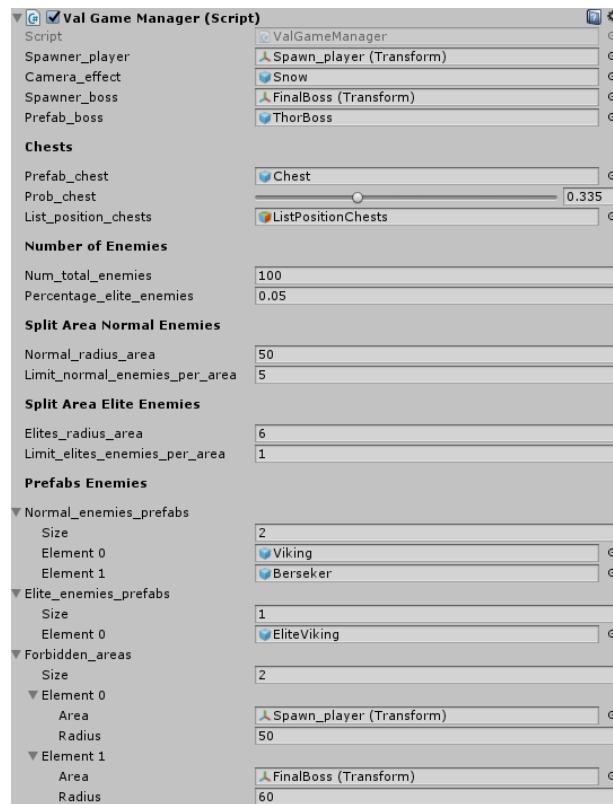


Ilustración 73. Valores de los atributos de la clase maestra de juego en el nivel uno



Ilustración 74. Nivel uno visto desde el editor

9.3. Personajes

Dios Valiel y futuro vendedor de objetos para los cazadores (ver Ilustración 75).



Ilustración 75. Modelo 3D del Dios Valiel

Dispone de una serie de animaciones básicas idle de Mixamo, que se repiten en bucle (ver Ilustración 76).

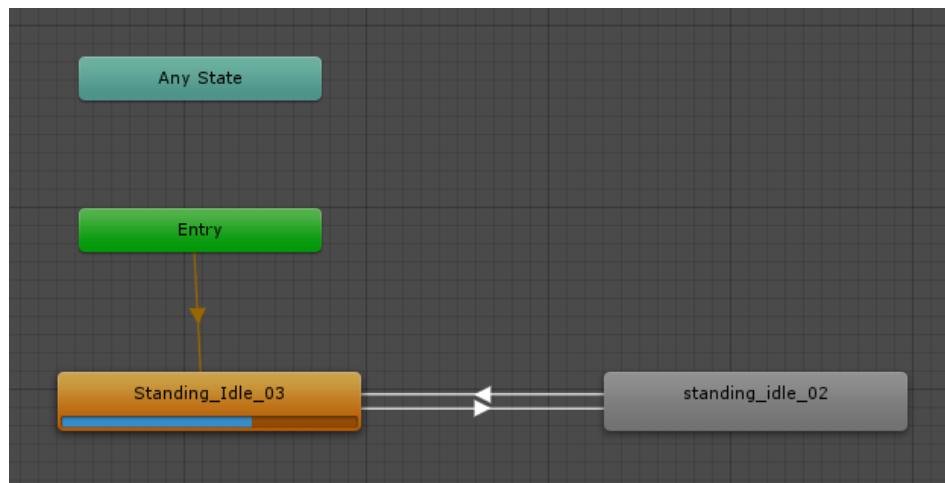


Ilustración 76. Sistema de animaciones para el Dios Valiel

El jugador puede hablar con dicho NPC haciendo click izquierdo en su modelo. En el momento en el que se interactúe con él, abrirá una interfaz como la de la siguiente ilustración (ver Ilustración 77).



Ilustración 77. Interfaz del Dios Valiel

El texto de la interfaz es el siguiente: “*Soy Váliel, el Dios de los arqueros e hijo de Odín y la giganta Rind. Tengo maestría y una puntería insuperable con el arco, y proporciono armaduras y armas, hechas por mí mismo, a los cazadores más fuertes*”.

Dios Ull y futuro vendedor de objetos para los guerreros vikingos (ver Ilustración 78).



Ilustración 78. Modelo 3D del Dios Ull

Dispone de las mismas animaciones que el Dios Valiel, y por tanto, hace uso del mismo animador. El jugador puede hablar con dicho NPC, haciendo click izquierdo en su modelo. En el momento en el que se interactúe con él, abrirá una interfaz como la de la siguiente ilustración (ver Ilustración 79).



Ilustración 79. Interfaz del Dios Ull

El texto de la interfaz es el siguiente: “*Soy Ull, el Dios de los guerreros. Soy el más diestro con la espada, el hacha y el escudo, y nadie puede superarme en duelo. Proporciono armaduras y armas, hechas por mí mismo, a los guerreros vikingos más fuertes*”.

Posadero y futuro comprador de objetos (ver Ilustración 80).



Ilustración 80. Modelo 3D del posadero

Dispone de una serie de animaciones idle obtenidas de la plataforma Mixamo, y están definidas en la siguiente imagen (ver Ilustración 81).

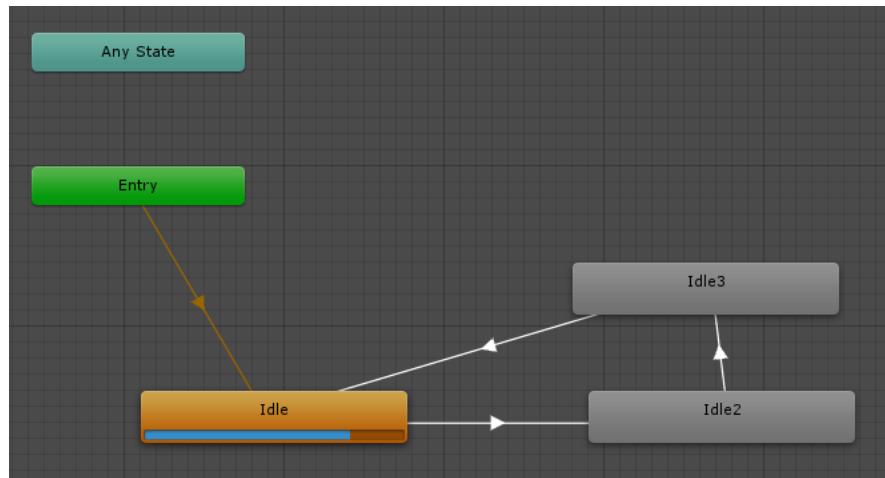


Ilustración 81. Sistema de animaciones para el posadero

El jugador puede hablar con dicho NPC haciendo click izquierdo en su modelo. En el momento en el que se interactúe con él, abrirá una interfaz como la de la siguiente ilustración (ver Ilustración 82).



Ilustración 82. Interfaz del posadero

El texto de la interfaz es el siguiente:

“Bienvenido a Yggdrasil mortal, desde aquí podrás acceder a los nueve mundos gracias a nuestro navegante que se encuentra en la playa, habla con él si deseas viajar.

En los últimos meses los dioses se comportan de un modo extraño, han sembrado de pánico y terror los nueve mundos, y no podemos dejar que destruyan toda la vida que habita en ellos.

Visita a Thor, está en Bilskirnir pero mucho cuidado, tiene un gran sequito de vikingos de Valhalla y un sin fin de criaturas mucho más temibles, si hace falta mátalo con tus propias manos”.

El **gigante Ysvar** y futuro comprador de objetos (ver Ilustración 83). Dispone de las mismas animaciones que el posadero, y por tanto, el mismo animador. El jugador puede hablar con dicho NPC, haciendo click izquierdo en su modelo. En el momento en el que se interactúe con él, abrirá una interfaz como la de la siguiente ilustración (ver Ilustración 84).

El texto de la interfaz es el siguiente: “*Me llamo Ysvar y soy un gigante de Jotunheim, huí en el ataque de Odín a mi tierra, y me refugié aquí para ayudar en todo lo posible y derrotar a los Dioses*”.



Ilustración 83. Modelo 3D del gigante Ysvar



Ilustración 84. Interfaz del gigante Ysvar

Barquero y encargado de proporcionar al usuario un cambio de nivel (ver Ilustración 85). Dispone de las mismas animaciones básicas que el posadero y el gigante, y por tanto, tienen el mismo animador. El jugador puede hablar con dicho NPC, haciendo click izquierdo en su modelo. En el momento en el que se interactúe con él, abrirá una interfaz como la de la siguiente ilustración (ver Ilustración 86).

Por último, están los espectros del Valhalla, capaces invocar héroes en la zona neutral (ver Ilustración 87). Tienen una única animación de invocación en bucle.



Ilustración 85. Modelo 3D del barquero



Ilustración 86. Interfaz del barquero



Ilustración 87. Modelo 3D de los espectros

9.4. Enemigos

Se han implementado todos los enemigos a excepción de los Dioses Freya, Loki y Odín. Todos y cada uno de ellos dispone de un modelo 3D con sus respectivas animaciones, y mantiene los atributos tal y como fueron definidos en las mecánicas.

La inteligencia artificial de éstos es muy básica:

- Comprueban cada cierto periodo de tiempo si hay jugadores o compañeros asociados a jugadores, en un determinado área alrededor suyo.
- Si encuentran un objetivo lo persiguen, y realizan el ataque más poderoso, siempre y cuando esté disponible y esté a rango.
- Si el objetivo se aleja demasiado, o simplemente muere, deja de perseguir al objetivo y vuelve a su estado inicial de búsqueda.

9.4.1. Vikingo

Su modelo 3D es el que aparece en la siguiente imagen (ver Ilustración 88).



Ilustración 88. Modelo 3D del enemigo vikingo

El componente animador define el siguiente grafo (ver Ilustración 89). Dispone de cuatro estados:

- **Estado Idle:** a su vez está subdividido en otro grafo con 3 estados idle. Este tipo de animaciones se ejecutan cuando el booleano standing es verdadero, o lo que es lo mismo, cuando el enemigo no se mueve.

- **Estado Run:** similar al anterior, con la diferencia de que es una sola animación. Se ejecuta si el booleano walking es verdadero, o lo que es lo mismo, cuando el enemigo se está moviendo.
- **Estado Attack:** se ejecuta cuando se activa el trigger correspondiente al ataque, volviendo a su estado anterior en función de booleano que esté activo.
- **Estado Death:** igual que el estado anterior pero para el caso de la muerte, y con la diferencia de que no volverá a ningún otro estado.

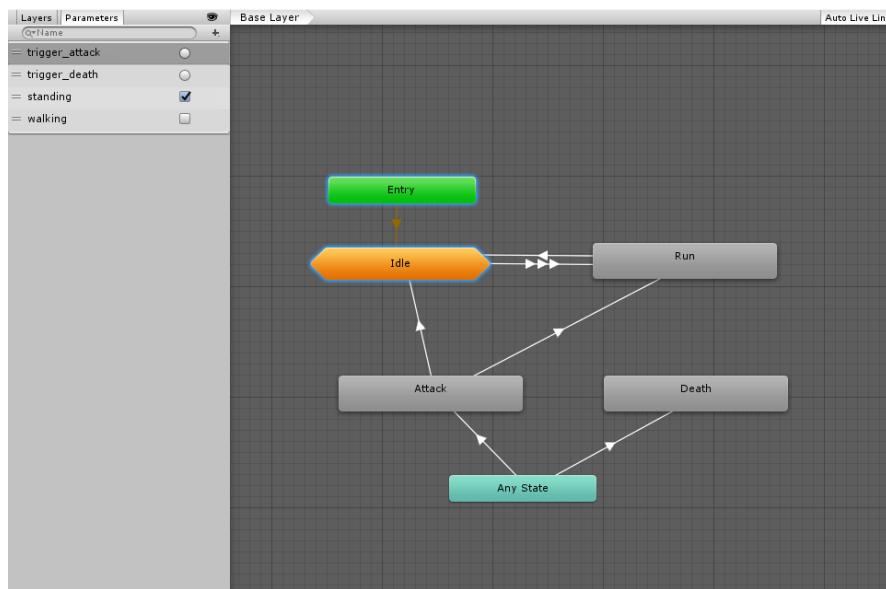


Ilustración 89. Sistema de animaciones para el enemigo Vikingo

Tiene implementado un único ataque, el ataque básico cuerpo a cuerpo.

9.4.2. Vikingo élite

Es exactamente igual que el enemigo anterior, con la diferencia de que es un poco más grande, y tiene un sistema de partículas de fuego (ver Ilustración 90).



Ilustración 90. Modelo 3D del vikingo élite

El sistema de animaciones es exactamente el mismo, únicamente difiere en los hechizos que posee. Como se definió en las mecánicas, todo enemigo élite tendría una serie de hechizos extras elegidos de un modo aleatorio. Pero en este prototipo, solo se han implementado dos hechizos, y al ser únicamente dos se elige haciendo una simple tirada aleatoria. Los posibles hechizos que puede tener el enemigo élite son los siguientes:

- Curación de un porcentaje de la vida.
- Reducción el daño recibido durante un periodo de tiempo.

Para un prototipo futuro, queda implementar los cuatro hechizos restantes y el sistema de elección a través de pesos.

9.4.3. Berserker

Es el penúltimo enemigo desarrollado y terminado. Su modelo es el siguiente (ver Ilustración 91).



Ilustración 91. Modelo 3D del enemigo berserker

El sistema de animaciones es el mismo que para los dos enemigos anteriores, teniendo en cuenta que hace uso de diferentes animaciones.

Se han implementado los dos ataques de los que disponía este enemigo:

- Realiza ataques básicos.
- Realiza un ataque en área que provoca un sangrado (daño que se repite en el tiempo)

9.4.4. Jefe final Thor

Por último, está el jefe final del primer nivel, el Dios Thor. Su modelo es el siguiente (ver Ilustración 92).

A diferencia de los demás modelos, éste no ha sido obtenido a través de la plataforma Mixamo. En un principio no disponía de ningún sistema de huesos para poder realizar animaciones, por lo que se subió a la plataforma de Mixamo y se le dotó de dicho sistema (ver Ilustración 93). Antes de realizar este proceso se le quitó las partes que eran innecesarias, es decir, toda su armadura y capa con la herramienta Blender.



Ilustración 92. Modelo 3D del Dios Thor

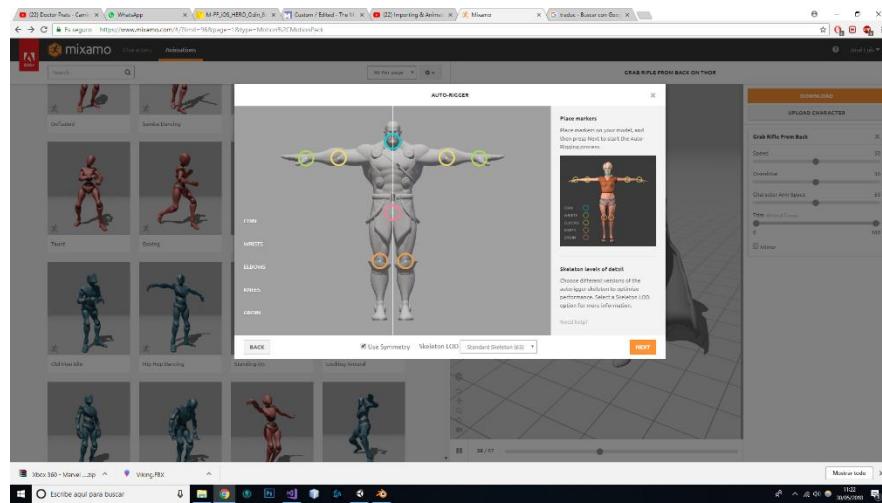


Ilustración 93. Subida de Thor a la plataforma Mixamo para la obtención de un sistema de huesos

Una vez hecho ésto ya era capaz de ejecutar cualquier animación de la plataforma Mixamo. El sistema de animaciones del jefe final es similar a los enemigos anteriores, añadiendo un nuevo tipo de animación, y por tanto, un nuevo trigger, pero el comportamiento es exactamente el mismo (ver Ilustración 94).

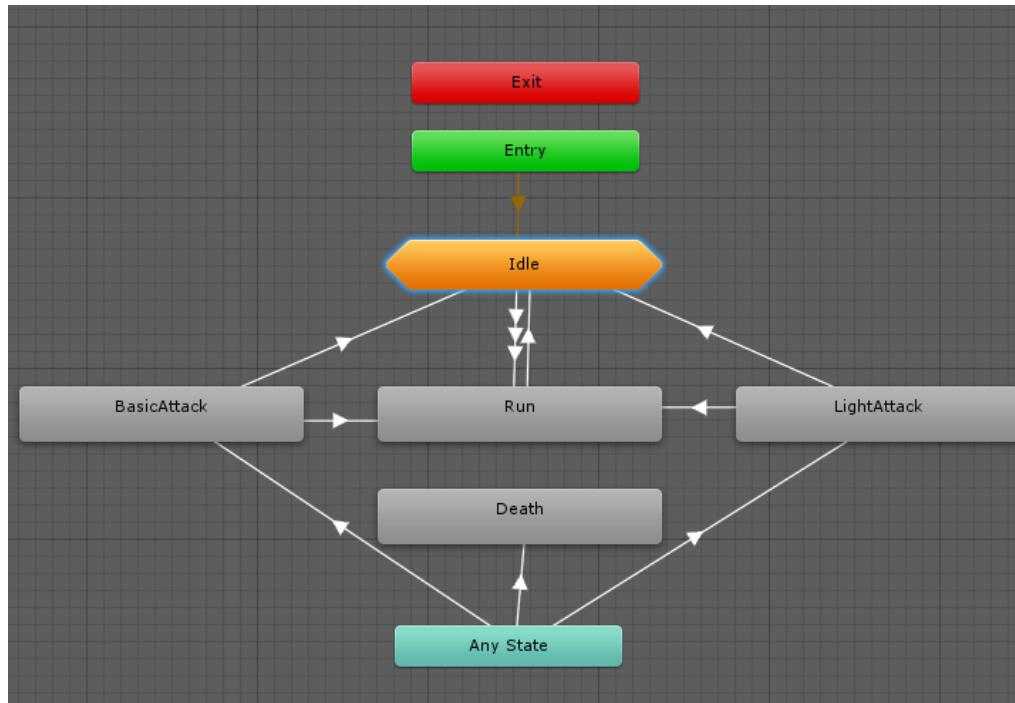


Ilustración 94. Sistema de animaciones del jefe final Thor

Dispone de todas las habilidades definidas en las mecánicas:

- Ataque básico.
- Ataque en área de rayos.
- Invocación de esbirros.
- Hechizo que protege a Thor durante un tiempo.

9.5. Clases o roles

En las mecánicas de clases se definieron dos tipos:

- Clase cazador.
- Clase guerrero vikingo.

Sólo se ha podido implementar la clase cazador, y únicamente se han terminado seis hechizos. Se ha modificado el nivel requerido de éstos para facilitar el testeo de la clase:

- Ataque básico. Nivel requerido: 1.
- Ataque explosivo. Nivel requerido: 3.
- Lanzamiento envenenado. Nivel requerido: 2.

- Marcado para morir. Nivel requerido: 3.
- Ataque potenciado. Nivel requerido: 2.
- Multidisparo. Nivel requerido: 2.

El modelo final del cazador es el siguiente (ver Ilustración 95).



Ilustración 95. Modelo 3D del cazador

Su sistema de animaciones similar al de los enemigos, y es el siguiente (ver Ilustración 96).

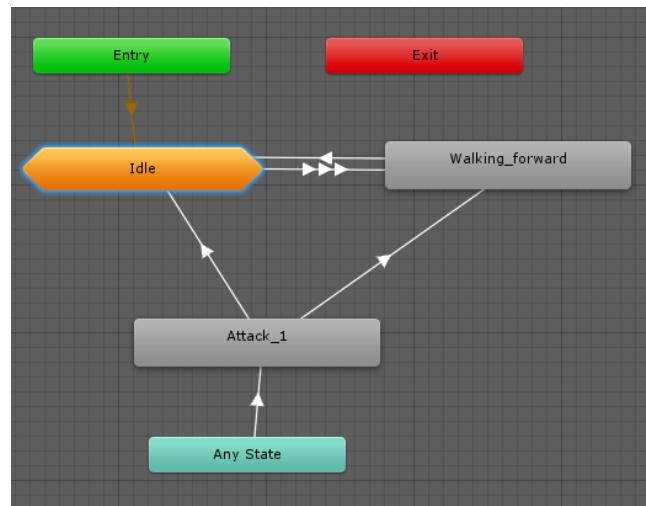


Ilustración 96. Sistema de animaciones para el cazador

Otro de los componentes de la clase cazador es que tenía un compañero permanente llamado Sköll, un perro lobo. El modelo del compañero lobo es el siguiente (ver Ilustración 97).



Ilustración 97. Modelo 3D del lobo Sköll

El sistema de animaciones del compañero es el siguiente (ver Ilustración 98).

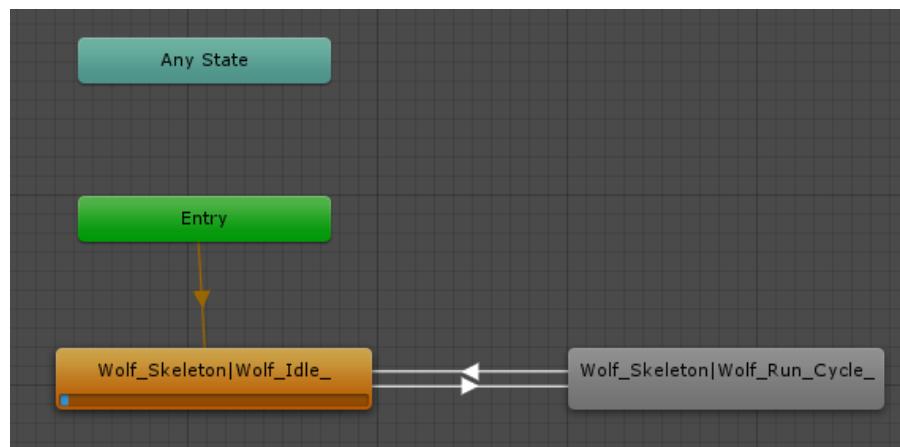


Ilustración 98. Sistema de animaciones del lobo Sköll

Tiene implementado el único ataque que fue definido, el ataque básico cuerpo a cuerpo.

9.6. Objetos que se pueden romper

En este prototipo sólo hay un objeto que se puede romper, y es la columna de piedra, definida por el siguiente modelo (ver Ilustración 99).

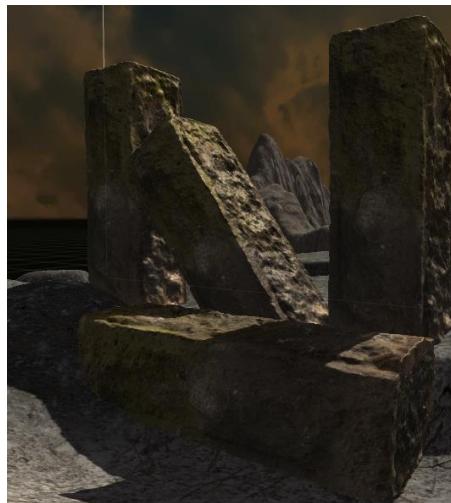


Ilustración 99. Modelo 3D de la columna que puede romperse

Para romper la columna, se divide en pequeños trozos a través de un plugin en Blender llamado Cell Fracture [46], y se sustituye por la columna original para conseguir el efecto de una fractura por un golpe.

Este objeto está disponible en el nivel uno.

9.7. Interfaz definida

La interfaz puede diferenciarse en dos bloques independientes:

- Interfaz del menú principal.
- Interfaz del resto del videojuego.

A continuación, se mostrará como ha quedado cada una de las secciones de la interfaz. Todas y cada una de las partes, se ha diseñado para que se adapte a cualquier tipo de resolución (dentro de unos valores normales).

9.7.1. Interfaz del menú principal

Cada uno de los principales paneles del menú principal: registro, inicio de sesión, y creación de personajes y comienzo de partida, tiene una serie de animaciones:

- **Left_in:** el panel entra por la parte izquierda.
- **Left_out:** el panel sale por la parte izquierda.
- **Right_in:** el panel entra por la parte derecha.
- **Right_out:** el panel sale por la parte derecha.



Ilustración 100. Panel con el formulario para iniciar sesión



Ilustración 101. Mensaje de error al equivocarse en las credenciales

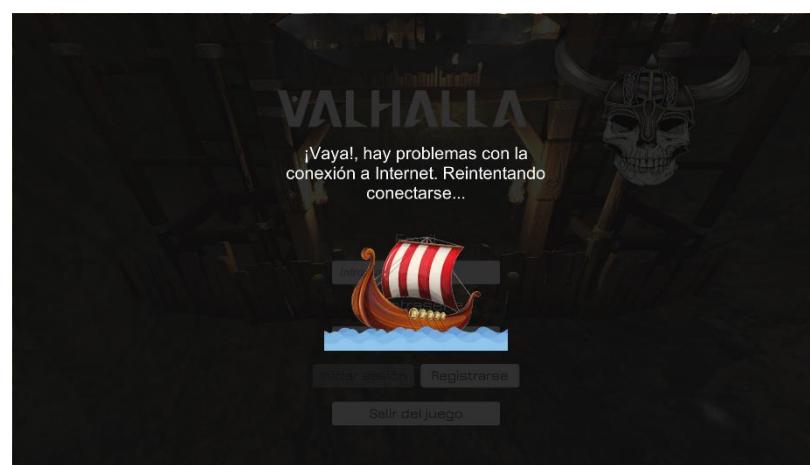


Ilustración 102. Mensaje de error cuando no hay conexión a Internet



Ilustración 103. Panel con el formulario de registro



Ilustración 104. Panel para seleccionar un personaje de la lista y comenzar una partida



Ilustración 105. Panel para la creación de un personaje

9.7.2. Interfaz del resto del juego

La interfaz principal dentro del juego, contiene los siguientes elementos (ver Ilustración 106):

- Dos círculos, uno verde y otro amarillo, que representan la vida y la energía del cazador respectivamente.
- Una barra de experiencia de color azul.
- Una barra de hechizos.
- Un panel con tres elementos:
 - La rueda, que abre el menú del juego (ver Ilustración 109).
 - La bolsa, que abre el inventario (ver Ilustración 108).
 - El libro, que abre el libro de hechizos (ver Ilustración 107).
- Mini mapa en la esquina superior derecha.
- Contador de frames por segundo y retraso en la conexión.



Ilustración 106. Interfaz principal dentro del juego



Ilustración 107. Interfaz del libro de hechizos



Ilustración 108. Interfaz del inventario



Ilustración 109. Interfaz del menú dentro de una partida

9.8. Música y sonidos

Los sonidos y música disponibles en el videojuego son:

- Música del menú principal: Northern Mist – Chulainn.
- Sonido de fondo de una tormenta en el menú principal.
- Música del nivel neutral: música de taberna de World of Warcraft.
- Música del primer nivel: Late Game – Butchers Bridge.

9.9. Controles del prototipo

Los controles disponibles son los siguientes:

- Desplazamiento del jugador haciendo click izquierdo del ratón en el terreno.

- Para poder lanzar habilidades se deben desplazar los hechizos a los huecos correspondientes en el libro de hechizos. Las diferentes teclas para accionar hechizos son:
 - Ratón izquierdo + barra espaciadora.
 - Ratón derecho.
 - Tecla Q.
 - Tecla W.
 - Tecla S.
 - Tecla R.
- Hay dos tipos de teclas:
 - **Teclas rápidas:** si dejas pulsada la combinación lanzará el hechizo todo el rato. Estas teclas son las del ratón.
 - **Teclas lentas:** si dejas pulsada la tecla no lanzará el hechizo hasta que la sueltes (resto de teclas). Si mantienes pulsada la tecla y mueves el ratón dibujará en el suelo la zona de acción del hechizo. El hechizo se lanzará una vez sueltes la tecla.
- Pulsar V para mostrar/ocultar las barras de vidas de enemigos y compañeros.
- Para abrir el menú pulsa escape.
- Para abrir el libro de hechizos pulsa H.
- Para abrir el inventario pulsa I.
- Otro modo de abrir las 3 opciones anteriores es a través de unos botones en el panel inferior derecho.
- Para interaccionar con los objetos:
 - Hacer click izquierdo en los objetos azules que representan los objetos armadura.
 - Pasar por encima de las calaveras para recogerlas.

- Para interaccionar con los NPCs hay que hacer click izquierdo en dichos personajes. Es necesario estar a menos de la distancia máxima.

10. Mejora en el sistema de colisiones de Unity

Como una de las secciones finales, aquí se tratan los mejores métodos para mejorar el uso de colisiones y la optimización de la matriz de colisiones de Unity. Para garantizar un producto de calidad y lo más eficiente posible, es necesario hacer uso de las mejores técnicas [47]. En este caso, la mayoría de los componentes del videojuego, se cimientan en las colisiones. Por lo tanto, es necesario optimizar en la medida de lo posible el uso de éstas.

Las colisiones son usadas desde una colisión implícita, por ejemplo, en un terreno con cada uno de los objetos posicionados en él, hasta la colisión explícita para obtener todos los enemigos o jugadores que hay a un área (usado mucho para la ejecución de ciertos hechizos).

Para realizar este tipo de mejoras es necesario hacer uso de dos tipos de componentes:

- **Layer o máscara:** componente visto con anterioridad en otras secciones. Es similar a las etiquetas de un GameObject, pero en este caso sirven para discriminar colisiones entre objetos, o discriminar colisiones a la hora de hacer ray casting o colisión en esfera³⁹
- **Matriz de colisiones:** es una matriz que registra como colisionan entre cada una de las máscaras especificadas (ver Ilustración 110).

Lo principal es definir una máscara para cada tipo de objeto que se prevé que hará uso de las colisiones. Como por ejemplo, en el caso de todas las entidades. De este modo, cuando se realice una colisión en esfera se puede especificar el tipo de máscara del objeto deseado, evitando que compruebe colisiones innecesarias, y garantizando que todos los objetos con los que ha hecho colisión son de ese tipo.

Por otro lado, es necesario optimizar la matriz, desmarcando parejas de máscaras pertenecientes a un tipo de objetos que sabes a ciencia cierta que no es

³⁹ Área con forma de esfera que se utiliza para encontrar que objetos colisionan con ella

necesario que colisionen, y de este modo evitar cálculos innecesarios por parte del motor gráfico. Por defecto todas las casillas están marcadas en la matriz.

Ilustración 110. Matriz de colisiones final del proyecto

Si no se hace uso de estas técnicas en un proyecto de tales dimensiones, y en una plataforma de pc, es posible que no se aprecie mucho el cambio. En cambio, en otras plataformas como pueden ser las consolas o móviles, es indispensable el uso de estas técnicas para garantizar la calidad y la correcta ejecución del software diseñado e implementado.

11. Control de versiones de prototipos

A lo largo de la vida del proyecto se han obtenido diversos prototipos. En total han sido 5, y son los siguientes:

- Prototipo versión 1.0.
 - Prototipo versión 1.1.
 - Prototipo versión 1.2.
 - Prototipo versión 1.3.
 - Prototipo versión 2.0.

A continuación, se muestran las características más importantes de las que disponían cada uno de los prototipos. También se adjuntará un video donde se puede apreciar el cambio de una versión a otra.

11.1. Prototipo v1.0

Primer prototipo del proyecto. Añade las siguientes características:

- Interfaz muy básica con información del jugador:
 - Vida.
 - Recurso.
 - Barra de experiencia.
 - Etc
- Interfaz para los enemigos y compañeros, y también está ya disponible el texto flotante con animaciones.
- Implementadas todas las mecánicas a excepción del inventario y los objetos procedurales.
- Implementado el sistema de combate.
- Libro de hechizos e interfaz de éste implementados. Por ahora no se incorporan los hechizos automáticamente a la barra de hechizos conforme se desbloquean.
- De momento, no es posible guardar el progreso.
- Escena básica para probar las mecánicas, donde se instancian de manera aleatoria alrededor de un cilindro los enemigos.
- Provisionalmente, tampoco hay sistema multijugador.
- Se le proporciona unos modelos básicos a todas las entidades:
 - **Personajes:** cápsulas verdes.
 - **Enemigos:** cilindros rojos.
 - **Compañeros:** cubos azules.

A continuación, se muestra una imagen de esta versión (ver Ilustración 111).



Ilustración 111. Imagen del prototipo v1.0 en el escenario para testear

11.2. Prototipo v1.1

Segundo prototipo del proyecto. Añade las siguientes características:

- Implementación de un sistema multijugador básico.
- Cambios en la interfaz del jugador:
 - Nueva distribución de algunos elementos.
 - Cambios en algunos colores.
- Inclusión de una escena con un menú principal básico.

En este prototipo se realizó una de las evaluaciones del proyecto. A continuación, se muestran unas imágenes del prototipo (ver Ilustración 112, Ilustración 113).



Ilustración 112. Nuevo menú principal



Ilustración 113. Jugador en la escena de prueba con la nueva interfaz y con otro jugador al lado

11.3. Prototipo v1.2

Tercer prototipo del proyecto. Añade las siguientes características:

- Implementación de la zona neutral. Las cápsulas blancas son los futuros NPCs.
- Cambio de la interfaz y muy parecido al diseño final.
- Inclusión de la mecánica del inventario.

- Ahora los enemigos dejan caer calaveras y objetos.
- Los objetos y las calaveras se pueden recoger y se incorporan al inventario.
- Se añade en la interfaz un inventario a modo visual.
- Se implementa la instanciación de enemigos mediante la malla regular.
- Se realiza una pequeña aplicación para probar la instanciación de enemigos en un terreno.
- Se realizan algunas mejoras en el sistema multijugador, añadiendo las notificaciones para poder cambiar de nivel. Por ahora, sólo se puede acceder al nivel de testeo.

A continuación, se muestran unas imágenes del prototipo (ver Ilustración 114, Ilustración 115, Ilustración 116, Ilustración 117, Ilustración 118).



Ilustración 114. Jugador en la zona neutral

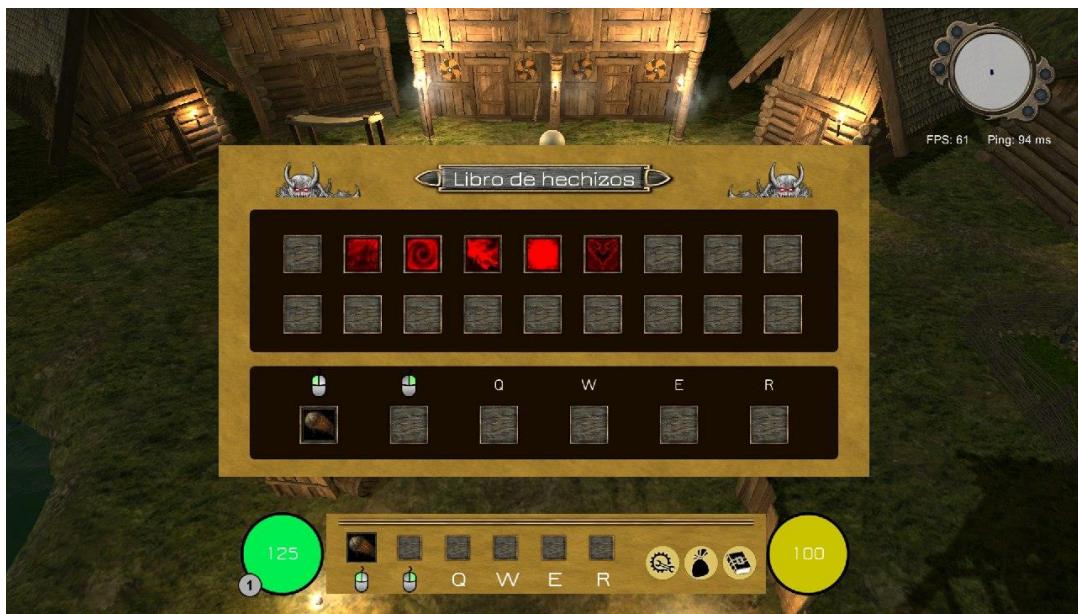


Ilustración 115. Nueva interfaz para libro de hechizos



Ilustración 116. Interfaz del barquero para cambiar de nivel, y la notificación que envía a todos los jugadores para poder cambiar



Ilustración 117. Nueva interfaz para el inventario. Zona superior izquierda atributos actuales del jugador, zona superior derecha objetos armadura equipados y la zona inferior para la mochila y las calaveras conseguidas



Ilustración 118. Nivel de testeo con objetos que han dejado caer los enemigos

11.4. Prototipo v1.3

Cuarto prototipo del proyecto. Añade las siguientes características:

- Conexión con la base de datos de PlayFab.
- Implementación del sistema de autentificación que utiliza tecnología PlayFab.

- Ahora se guarda el progreso de los personajes.
- Nuevo menú del juego adaptado al sistema de autentificación y a la información de personajes guardados.
- Algunas mejoras simples en la interfaz del juego. Aquí se define la interfaz definitiva de Valhalla.
- Algunas mejoras en el terreno del nivel neutral.

A continuación, se muestran unas imágenes del prototipo (ver Ilustración 119, Ilustración 120).

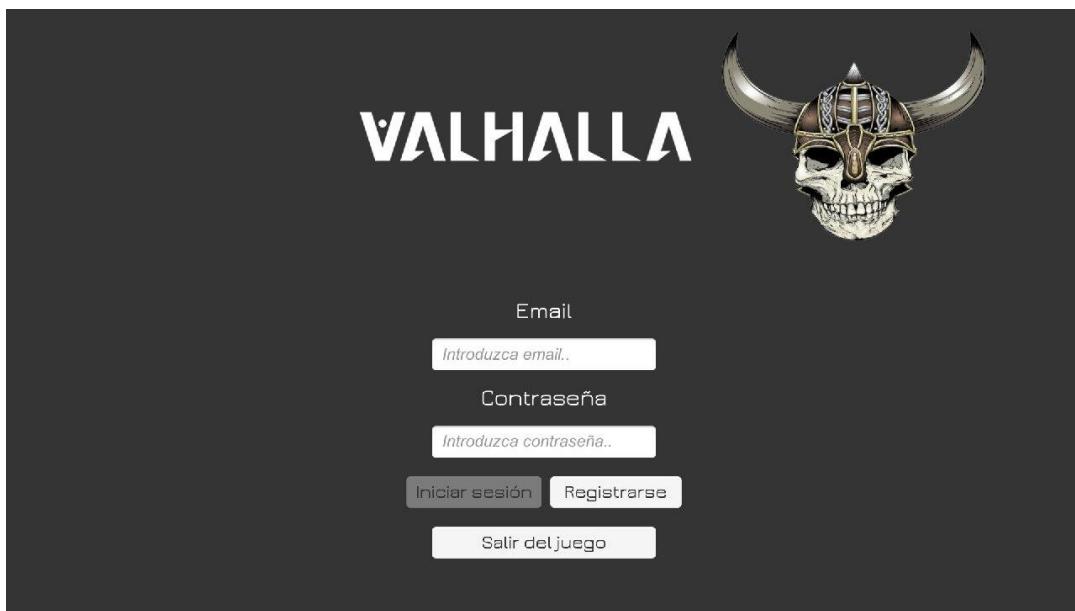


Ilustración 119. Nuevo menú principal para poder autentificar a un jugador con una cuenta



Ilustración 120. Jugador en la zona neutral modificada y con la interfaz definitiva

11.5. Prototipo v2.0

Quinto prototipo y último del proyecto. Añade las siguientes características:

- Cambios en el menú principal: animaciones de los paneles, mensajes de error, terreno neutral de fondo, sistema de partículas para la lluvia, y se añade sonido de una tormenta de fondo.
- Inclusión de modelos 3D con sus respectivas animaciones (ya no hay modelos básicos).
- Efectos de las habilidades del cazador realizados con sistema de partículas.
- Sonidos para algunas habilidades.
- Música de fondo en la escena del terreno neutral y en el primer nivel.
- Realización del primer nivel con jefe final Thor.

Las imágenes de esta versión ya se han mostrado en la sección del prototipo final.

12. Evaluaciones de los prototipos

Una fase fundamental final en la fase cíclica del desarrollo, es el testeo de la aplicación por parte del cliente, proporcionando un feedback a los desarrolladores para hacer posibles mejoras, y adaptarlo lo mejor posible a lo que el usuario desea.

Dada las limitaciones, he realizado únicamente un testeo con grupo relativamente pequeño de usuarios, que han ejecutado la aplicación y han respondido a un formulario para detectar posibles fallos y recoger nuevas ideas o elementos que pueden faltar al videojuego o que resultan confusos.

Las evaluaciones se han realizado sobre los prototipos v1.1 y v2.0. Para ello, se proporcionó a los usuarios un formulario en la plataforma de Google, en el cual, antes de realizar las preguntas, se proporcionaba un ejecutable del prototipo y un documento con los controles y acciones que podía llevar a cabo.

12.1. Primera evaluación

La primera evaluación se realizó sobre el **prototipo v1.1**, en pleno ecuador de la implementación del videojuego, y con la mayoría de elementos fundamentales terminados.

Las preguntas del cuestionario eran las siguientes:

- ¿Con qué frecuencia juega a videojuegos? Escala lineal del 1 al 5, donde 1 es nada y 5 es mucho (ver Tabla 29).
- ¿Es jugador habitual de videojuegos RPG del estilo Diablo 3 o Path of Exile? Selección múltiple con respuestas Sí y No (ver Tabla 30).
- ¿La cámara del jugador está debidamente posicionada? Selección múltiple con las siguientes respuestas (ver Tabla 31):
 - La cámara está demasiado cerca, no me deja ver bien lo que me rodea.
 - La cámara está demasiado lejos.
 - Está perfecta.
- ¿El control del jugador es fácil e intuitivo? Si la respuesta es "No" indique a continuación el porqué. Respuesta párrafo (ver Tabla 32).

- ¿Entiende la utilidad de todos los elementos de la interfaz? Si la respuesta es “No” indique a continuación cuáles son y el porqué. Respuesta párrafo (ver Tabla 33).
- ¿Le ha resultado fácil incorporar hechizos, cambiarlos de posición o quitarlos? Si la respuesta es “No” indique a continuación el porqué. Respuesta párrafo (ver Tabla 34).
- ¿Qué color tiene el texto flotante que indica el daño recibido? Selección múltiple con respuestas Rojo y Amarillo (ver Tabla 35).
- ¿Le ha resultado difícil infiligrir daño a los enemigos? Si la respuesta es “Sí” indique a continuación el porqué. Respuesta párrafo (ver Tabla 36).
- Indique qué mejoras haría en la interfaz ya existente. Respuesta párrafo (ver Tabla 37).
- Indique qué elementos incorporaría a la interfaz que no se encuentran actualmente y serían indispensables. Respuesta párrafo (ver Tabla 38).
- Si quiere comentar alguna mejora o algún problema que no haya podido comentar en las cuestiones anteriores hágalo a continuación. Respuesta párrafo (ver Tabla 39).

Para este cuestionario hubo un total de 6 encuestados que probaron el ejecutable y posteriormente, completaron la encuesta de manera anónima. Las respuestas están divididas por tablas, y cada una de ellas representa una pregunta en concreto.

¿Con qué frecuencia juega a videojuegos?	
Encuestado	Respuesta
Encuestado 1	4
Encuestado 2	4
Encuestado 3	4
Encuestado 4	4
Encuestado 5	5
Encuestado 6	1

Tabla 29. Respuestas de la primera pregunta del formulario v1.1

Analizando esta pregunta se puede ver que la mayoría de encuestados, a excepción del último, suele jugar con una frecuencia bastante alta. Es importante saberlo para comprobar con qué factores del videojuego tienen problemas los usuarios más expertos y los menos expertos.

¿Es jugador habitual de videojuegos RPG del estilo Diablo 3 o Path of Exile?	
Encuestado	Respuesta
Encuestado 1	No
Encuestado 2	Sí
Encuestado 3	No
Encuestado 4	No
Encuestado 5	Sí
Encuestado 6	No

Tabla 30. Respuestas de la segunda pregunta del formulario del prototipo v1.1

Un 66% de los encuestados no suele ser consumidor habitual de videojuegos del estilo de Valhalla. Por lo tanto, es posible que no estén muy familiarizados con sus mecánicas ni con el modo de interactuar con la interfaz.

¿La cámara del jugador está debidamente posicionada?	
Encuestado	Respuesta
Encuestado 1	Está perfecta
Encuestado 2	Está perfecta
Encuestado 3	Está perfecta
Encuestado 4	Está perfecta
Encuestado 5	Está perfecta
Encuestado 6	Está perfecta

Tabla 31. Respuestas de la tercera pregunta del formulario del prototipo v1.1

El 100% de los jugadores ha coincidido en que la cámara está perfecta. Por lo que no será necesario tocar este componente.

¿El control del jugador es fácil e intuitivo? Si la respuesta es "No" indique a continuación el porqué	
Encuestado	Respuesta
Encuestado 1	Sí
Encuestado 2	Es intuitivo y fácil
Encuestado 3	Sí
Encuestado 4	Sí
Encuestado 5	Sí
Encuestado 6	Me ha costado moverme con fluidez al principio

Tabla 32. Respuestas de la cuarta pregunta del formulario del prototipo v1.1

El 83% de los encuestados coincide en que es fácil e intuitivo, a diferencia de la última persona que recordemos que no juega habitualmente a juegos. Como conclusión, se puede sacar que sería recomendable un minitutorial al comenzar a jugar por primera vez.

¿Entiende la utilidad de todos los elementos de la interfaz? Si la respuesta es "No" indique a continuación cuáles son y el porqué	
Encuestado	Respuesta
Encuestado 1	Sí
Encuestado 2	Sí, aunque puede llegar a algo de confusión que la barra de magia sea amarilla en vez de azul como suele ser
Encuestado 3	Sí, pero estaría mejor si se añadiera una breve descripción de la utilidad de los hechizos
Encuestado 4	Sí
Encuestado 5	Sí, aunque hay dos botones en el juego que no funcionan, imagino que aún no tienen funcionalidad.
Encuestado 6	Creo que sí, pero hay dos botones que no hacen nada

Tabla 33. Respuestas de la quinta pregunta del formulario del prototipo v1.1

La mayoría no tiene problemas graves en la identificación de elementos de la interfaz. La respuesta del segundo puede justificarse ante la desinformación de la clase que está jugando actualmente, por norma general el recurso del tipo energía suele mostrarse con un color amarillo.

También es importante no añadir botones que no hacen por ahora nada para la siguiente versión, y añadir información sobre los hechizos, un objetivo que ya lo tuve en cuenta pero que por ahora no he podido implementar.

¿Le ha resultado fácil incorporar hechizos, cambiarlos de posición o quitarlos? Si la respuesta es "No" indique a continuación el porqué	
Encuestado	Respuesta
Encuestado 1	Sí
Encuestado 2	Es fácil pero al principio he intentado anclarlos directamente en vez de pasar por la barra intermedia, la cual quizás sea redundante
Encuestado 3	Sí, pero debería de avisar que ciertos hechizos se tienen que desbloquear al alcanzar un cierto nivel
Encuestado 4	Sí, pero debería avisar cuando se pueden desbloquear ciertos hechizos.
Encuestado 5	Sí, pero yo añadiría que los hechizos que se pueden usar se incorporen directamente a los hechizos que utilizas.
Encuestado 6	Sí

Tabla 34. Respuestas de la sexta pregunta del formulario del prototipo v1.1

Algunas de las respuestas son similares a la anterior pregunta. Se llega a la misma conclusión que antes, falta de información sobre los hechizos: qué hacen, su nombre y cómo se desbloquean.

¿Qué color tiene el texto flotante que indica el daño recibido?	
Encuestado	Respuesta
Encuestado 1	Rojo

Encuestado 2	Rojo
Encuestado 3	Rojo
Encuestado 4	Amarillo
Encuestado 5	Rojo
Encuestado 6	Amarillo

Tabla 35. Respuestas de la séptima pregunta del formulario del prototipo v1.1

Un 66% de los encuestados ha acertado la pregunta. Las dos personas que han fallado nunca han jugado a videojuegos de este estilo, y una de ellas no juega habitualmente a nada.

¿Le ha resultado difícil infiligr daño a los enemigos? Si la respuesta es “Sí” indique a continuación el porqué	
Encuestado	Respuesta
Encuestado 1	No
Encuestado 2	No
Encuestado 3	No
Encuestado 4	Sí, porque me ha costado entenderlo
Encuestado 5	No
Encuestado 6	No, es muy fácil

Tabla 36. Respuestas de la octava pregunta del formulario del prototipo v1.1

Todos, a excepción de una persona, no les ha costado infiligr daño a los enemigos.

Indique qué mejoras haría en la interfaz ya existente	
Encuestado	Respuesta
Encuestado 1	Posicionaría el mini mapa en la parte inferior de la pantalla, de tal forma que de un vistazo se puedan ver los enfriamientos de las habilidades, la posición de cada jugador en el mapa...

Encuestado 2	La indicación de los fps la cambiaría de lugar a la esquina izquierda superior
Encuestado 3	Añadir la cantidad de energía porcentualmente y no sólo visualmente
Encuestado 4	Está bien así
Encuestado 5	Le añadiría en la barra de vida y de energía la cantidad que se tiene actualmente en texto
Encuestado 6	Ninguna

Tabla 37. Respuestas de la novena pregunta del formulario v1.1

Indique qué elementos incorporaría a la interfaz que no se encuentran actualmente y serían indispensables	
Encuestado	Respuesta
Encuestado 1	Nada
Encuestado 2	No se me ocurre ninguno, los básicos están
Encuestado 3	Nada
Encuestado 4	Pócimas para revivir
Encuestado 5	La latencia, y un menú de salida
Encuestado 6	Nada

Tabla 38. Respuestas de la décima pregunta del formulario v1.1

Si quiere comentar alguna mejora o algún problema que no haya podido comentar en las cuestiones anteriores hágalo a continuación	
Encuestado	Respuesta
Encuestado 1	Utilizar colores más intuitivos (y más diferenciados) para mostrar la cantidad de maná y de vida, por ejemplo, verde para la vida y azul para el maná. También sería interesante que el mini mapa tuviera mayor rango de visión
Encuestado 2	No funciona el botón de configuración ni el de inventario (la bolsa y la rueda de engranajes) pero no sé si es porque es un

	prototipo. Además, hay un pequeño bug que cuando intentas añadir un hechizo a una tecla ya asignada el hechizo que estaba asignado con anterioridad se queda debajo y ya no lo puedes asignar hasta que quitas el hechizo con el que lo has sustituido (en la misma tecla me refiero), solucionarlo debería ser relativamente fácil, más que nada a la hora de asignar un hechizo a la tecla habría que comprobar si hay alguno y si lo hay, cambiarlo a la barra de arriba con todos los demás hechizos o nunca quitar los hechizos de la barra de arriba aunque los asigne
Encuestado 3	Nada
Encuestado 4	Me ha costado distinguir los colores asignadas a las puntuaciones.
Encuestado 5	Nada
Encuestado 6	Nada

Tabla 39. Respuestas de la décima primera pregunta del formulario del prototipo v1.1

Tras analizar todas y cada una de las respuestas, llegué a las siguientes conclusiones:

- Para los siguientes prototipos no dejaré botones o recursos de la interfaz que no hagan nada y confundan a los usuarios.
- Sería necesario incluir un mini tutorial para aquellos que no estén familiarizados con este tipo de juegos (o ninguno en general), explicando las mecánicas básicas.
- Es necesario mostrar más información relativa a la vida y al recurso, proporcionando los valores que tienen en ese momento.
- En una versión futura añadir una pequeña ventana que sea mostrada al posicionar el ratón sobre la habilidad, y proporcione:
 - Nombre de la habilidad.
 - Qué hace.
 - Cuánto cuesta lanzarla.

- Nivel requerido.
- El mini mapa debe tener un mayor rango de visión.
- Los usuarios echan en falta algunos elementos informativos, como el contador de latencia.
- Al parecer hay un error en el libro de hechizos. Es necesario encontrar la casuística que provoca el error.

Una vez analizado se incorporó aquellas propuestas que fueron factibles en la siguiente versión v1.2:

- Se aumentó el rango de visión del mini mapa.
- Se hizo un cambio en las barras de vida y recurso para comprobar en una versión futura si mejora el entendimiento.
- Se añadieron los valores de vida y recursos también con números.
- Se incluyó el contador de latencia.
- Por último, se arregló un error en la interfaz del libro de hechizos.

12.2. Segunda evaluación

La segunda evaluación se realizó sobre el **prototipo v2.0**, es decir, para la versión final que se presenta en este trabajo fin de grado.

Las preguntas del cuestionario son las mismas que las anteriores pero con algún añadido:

- ¿Con qué frecuencia juega a videojuegos? Escala lineal del 1 al 5, donde 1 es nada y 5 es mucho (ver Tabla 40).
- ¿Es jugador habitual de videojuegos RPG del estilo Diablo 3 o Path of Exile? Selección múltiple con respuestas Sí y No (ver Tabla 41).
- ¿La cámara del jugador está debidamente posicionada? Selección múltiple con las siguientes respuestas (ver Tabla 42):
 - La cámara está demasiado cerca, no me deja ver bien lo que me rodea.
 - La cámara está demasiado lejos.
 - Está perfecta.

- ¿El control del jugador es fácil e intuitivo? Si la respuesta es "No" indique a continuación el porqué. Respuesta párrafo (ver Tabla 43).
- ¿Entiende la utilidad de todos los elementos de la interfaz? Si la respuesta es "No" indique a continuación cuáles son y el porqué. Respuesta párrafo (ver Tabla 44).
- ¿Le ha resultado fácil incorporar hechizos, cambiarlos de posición o quitarlos? Si la respuesta es "No" indique a continuación el porqué. Respuesta párrafo (ver Tabla 45).
- ¿Qué color tiene el texto flotante que indica el daño recibido? Selección múltiple con respuestas Rojo y Amarillo (ver Tabla 46).
- ¿Le ha resultado difícil infligir daño a los enemigos? Si la respuesta es "Sí" indique a continuación el porqué. Respuesta párrafo (ver Tabla 47).
- Indique qué mejoras haría en la interfaz ya existente. Respuesta párrafo (ver Tabla 48).
- Indique qué elementos incorporaría a la interfaz que no se encuentran actualmente y serían indispensables. Respuesta párrafo (ver Tabla 49).
- ¿Cree que la música está demasiado alta? Respuesta múltiple con las respuestas Sí o No (ver Tabla 50).
- Si quiere comentar alguna mejora o algún problema que no haya podido comentar en las cuestiones anteriores hágalo a continuación. Respuesta párrafo (ver Tabla 51).

Para este cuestionario hubo un total de 7 encuestados que probaron el ejecutable, y posteriormente completaron la encuesta de manera anónima. Las respuestas están divididas por tablas, y cada una de ellas representa una pregunta en concreto.

¿Con qué frecuencia juega a videojuegos?	
Encuestado	Respuesta
Encuestado 1	5
Encuestado 2	4

Encuestado 3	4
Encuestado 4	4
Encuestado 5	2
Encuestado 6	4
Encuestado 7	4

Tabla 40. Respuestas de la primera pregunta del formulario del prototipo v2.0

Al igual que la encuesta anterior, todos los encuestados salvo una persona, juegan bastante a juegos en general.

¿Es jugador habitual de videojuegos RPG del estilo Diablo 3 o Path of Exile?	
Encuestado	Respuesta
Encuestado 1	No
Encuestado 2	Sí
Encuestado 3	No
Encuestado 4	Sí
Encuestado 5	No
Encuestado 6	Sí
Encuestado 7	No

Tabla 41. Respuestas de la segunda pregunta del formulario del prototipo v2.0

De igual modo que en la anterior encuesta, el número de personas que no han jugado a videojuegos de este tipo es mayor.

¿La cámara del jugador está debidamente posicionada?	
Encuestado	Respuesta
Encuestado 1	Está perfecta
Encuestado 2	Está perfecta
Encuestado 3	La cámara está demasiado cerca, no me deja ver bien lo que me rodea

Encuestado 4	Está perfecta
Encuestado 5	Está perfecta
Encuestado 6	Está perfecta
Encuestado 7	La cámara está demasiado cerca, no me deja ver bien lo que me rodea

Tabla 42. Respuestas de la tercera pregunta del formulario del prototipo v2.0

La cámara fue modificada con respecto a la versión 1.1, y fue alejada un poco más, aunque los usuarios decían que estaba bien situada. Aun así, hay gente que quiere ver más el entorno que los rodea.

¿El control del jugador es fácil e intuitivo? Si la respuesta es "No" indique a continuación el porqué	
Encuestado	Respuesta
Encuestado 1	Sí
Encuestado 2	Sí, con cierto problema referente a la selección para la habilidad R por ejemplo, no me selecciona bien los enemigos
Encuestado 3	Sí, es fácil e intuitivo
Encuestado 4	Es fácil e intuitivo
Encuestado 5	Sí
Encuestado 6	Sí
Encuestado 7	Sí

Tabla 43. Respuestas de la cuarta pregunta del formulario del prototipo v2.0

La gran mayoría no han tenido ningún problema. Quizás tuvo problema con la habilidad de marcar objetivo porque no dejó pulsado la tecla para comprobar cuál era su rango máximo.

¿Entiende la utilidad de todos los elementos de la interfaz? Si la respuesta es "No" indique a continuación cuáles son y el porqué	
Encuestado	Respuesta

Encuestado 1	Sí
Encuestado 2	Sí, se entiende perfectamente
Encuestado 3	Sí
Encuestado 4	Sí
Encuestado 5	Sí
Encuestado 6	Sí
Encuestado 7	Sí

Tabla 44. Respuestas de la quinta pregunta del formulario del prototipo v2.0

Los cambios en la interfaz han resultado satisfactorios y no lleva confusión en ninguno de los usuarios.

¿Le ha resultado fácil incorporar hechizos, cambiarlos de posición o quitarlos? Si la respuesta es “No” indique a continuación el porqué	
Encuestado	Respuesta
Encuestado 1	Ha sido fácil
Encuestado 2	Al principio me ha costado entender que desde el comienzo del juego no puedes incorporarlos todos, sino que se va desbloqueando
Encuestado 3	Sí
Encuestado 4	Sí
Encuestado 5	Sí
Encuestado 6	Sí
Encuestado 7	Sí

Tabla 45. Respuestas de la sexta pregunta del formulario del prototipo v2.0

Al igual que en la encuesta anterior, sigue estando el mismo problema, falta de información en los hechizos.

¿Qué color tiene el texto flotante que indica el daño recibido?	
Encuestado	Respuesta
Encuestado 1	Rojo
Encuestado 2	Rojo
Encuestado 3	Rojo
Encuestado 4	Rojo
Encuestado 5	Rojo
Encuestado 6	Rojo
Encuestado 7	Rojo

Tabla 46. Respuestas de la séptima pregunta del formulario del prototipo v2.0

Esta vez, todos han acertado y no ha habido ningún problema.

¿Le ha resultado difícil infiligrar daño a los enemigos? Si la respuesta es “Sí” indique a continuación el porqué	
Encuestado	Respuesta
Encuestado 1	No
Encuestado 2	No, pero algunas veces no podía seleccionarlos (habilidad R)
Encuestado 3	Sí, es demasiado difícil conseguir matar al enemigo
Encuestado 4	No
Encuestado 5	No
Encuestado 6	No
Encuestado 7	Cuando me veían y se dirigían a mí me devolvían a la playa. No sé si es que no te pueden ver o que me acababan matando muy rápido. Soy muy malo, lo reconozco. Pero, se veía bien la distancia a la que llegaba la flecha y sabía a partir de cual podía infringir daño

Tabla 47. Respuestas de la octava pregunta del formulario del prototipo v2.0

Uno de los principales problemas de la versión actual, es el desequilibrio en enemigos y armaduras. Al comienzo es bastante difícil sobrevivir, ya que el enemigo berserker mata prácticamente de un golpe, pero cuando el jugador empieza a obtener armadura, es bastante difícil que lo maten.

Indique qué mejoras haría en la interfaz ya existente	
Encuestado	Respuesta
Encuestado 1	Nada
Encuestado 2	Poner las estadísticas de los objetos y armas encontrados
Encuestado 3	Está bien así
Encuestado 4	Ninguna. Me parece que está completa y es fácil de entender
Encuestado 5	Ninguna
Encuestado 6	Ninguna
Encuestado 7	No sé qué mejoraría. La veo bien

Tabla 48. Respuestas de la novena pregunta del formulario del prototipo v2.0

Al igual que con los hechizos sería necesario mostrar la información de un objeto recogido.

Indique qué elementos incorporaría a la interfaz que no se encuentran actualmente y serían indispensables	
Encuestado	Respuesta
Encuestado 1	Ninguno
Encuestado 2	Está todo lo esencial
Encuestado 3	No añadiría ninguno más
Encuestado 4	Ninguna. Está bien así
Encuestado 5	Ninguno
Encuestado 6	Ninguno

Encuestado 7	Ninguno
--------------	---------

Tabla 49. Respuestas de la décima pregunta del formulario del prototipo v2.0

¿Cree que la música está demasiado alta?	
Encuestado	Respuesta
Encuestado 1	No
Encuestado 2	No
Encuestado 3	No
Encuestado 4	No
Encuestado 5	No
Encuestado 6	No
Encuestado 7	No

Tabla 50. Respuestas de la décima primera pregunta del formulario del prototipo v2.0

No hay ningún problema con la música añadida en la reciente versión.

Si quiere comentar alguna mejora o algún problema que no haya podido comentar en las cuestiones anteriores hágalo a continuación	
Encuestado	Respuesta
Encuestado 1	Nada
Encuestado 2	La selección de enemigos como he comentado antes y mostrar también que es posible utilizar la primera habilidad con el espacio y no sólo con el ratón
Encuestado 3	Estaría mejor si el zoom se pudiera alejar, y también que se pudiera elegir el nivel de dificultad para jugar
Encuestado 4	Que haya posibilidad de elegir el nivel de dificultad y añadiría un mensaje de felicitación al finalizar el juego
Encuestado 5	Que no te maten con que te respiren en la cara

Encuestado 6	Las habilidades Q y W pondría que al situar el cursor sobre ellas en la barra de abajo, me mostrase el cuadro de texto indicando la función de cada una
Encuestado 7	A nivel personal, poder mover la cámara (estilo GTA o Call of duty) con el ratón (por ver mejor alrededor, aunque para no poder tocar la cámara y desde esa perspectiva está bastante bien posicionada) y mover el personaje con las flechas

Tabla 51. Respuestas de la décima segunda pregunta del formulario v2.0

Tras analizar todas y cada una de las respuestas, llegué a las siguientes conclusiones:

- El principal problema es la ausencia de información en hechizos y objetos, dejando esta mejora para versiones futuras.
- Hay un desequilibrio, la dificultad debe aumentar progresivamente, pero no estancarse, no debe ser ni demasiado difícil ni demasiado fácil. Es necesario ajustar los valores de los enemigos y los valores que proporcionan los atributos de las armaduras.
- Es necesario pulir algunos elementos, como la posición de la cámara, añadir la posibilidad de hacer zoom, y algunos elementos del menú principal.

13. Conclusiones finales

Como colofón final, en esta sección se pretende hacer un análisis global del proyecto, analizando el alcance de éste, las herramientas y tecnologías usadas, y algunas mejoras que se pueden llevar a cabo en el proyecto.

En primer lugar, el motor de videojuegos Unity me ha facilitado alcanzar y terminar un gran porcentaje del diseño inicial. Me ha ayudado a profundizar y a adquirir una mayor experiencia en este mundo, y me ha sido muy fácil la consulta de pequeñas dudas, búsqueda de documentación y el acceso a una gran cantidad de tutoriales. En cambio, Unity tiene algunos componentes que dejan mucho que desear, como es el

caso de la malla de navegación que se ha hecho uso para mover a personajes, enemigos y compañeros. Para mejorar ese componente, se podría implementar el algoritmo búsqueda en A estrella. Por otro lado, su sistema de iluminación tampoco es tan bueno si se compara con motores como Unreal Engine, dejando mucho que desear en las calidad de las soft shadows (no hay apenas diferencias con las sombras duras), y con el componente material que no permite efectos tan elaborados como en Unreal.

En segundo lugar, cabe destacar el resultado obtenido con las dos tecnologías que han permitido el sistema online. Por un lado, está la tecnología Photon, usada para implementar el sistema multijugador.

En general, la documentación es bastante completa, y hay tutoriales suficientes para comenzar a utilizarlo. Aunque permite la sincronización de la posición y animaciones, no ofrece resultados demasiado fluidos. Por lo general, la latencia suele oscilar entre los 70 y los 90 milisegundos (un retraso aceptable), pero aun así, no sincroniza adecuadamente las animaciones en la zona del primer nivel, quizás provocado por el gran número de enemigos que tiene que actualizar de manera simultánea (más de 100) y no está diseñado para tales fines, o bien la versión gratuita limita en gran medida las peticiones de sincronización. También cabe destacar que el número de usuarios simultáneos está muy limitado, teniendo que pagar una gran cantidad de licencias para poder tener un número ilimitado. En conclusión, aunque Photon te solventa en gran medida muchos problemas, es más recomendable la implementación por ti mismo a nivel de bytes de la sincronización de cualquier información que haga falta, dando lugar por ejemplo, a unas animaciones más fluidas y un número ilimitado de usuarios conectados, aunque quizás sea más que suficiente para videojuegos que no necesitan una sincronización masiva de información.

La segunda tecnología utilizada, ha sido PlayFab, que en general ha funcionado bastante bien, pero presenta un problema en la autenticación e implica un problema general. Por desgracia, es imposible cerrar sesión y comprobar si una cuenta está en uso, y a día de hoy no han dado ninguna solución. Este inconveniente da lugar a muchos problemas, como por ejemplo, permitir que dos jugadores estén jugando con la misma cuenta y el mismo personaje simultáneamente, originando problemas en la actualización de la información del personaje. Este es un problema que no he podido solventar y que sin lugar a dudas, me llevaría a implementar mi propio sistema de

autentificación en un futuro en un servidor remoto con base de datos incluida. Otra posible solución sería esperar a que el servicio de Firebase de Google esté disponible para más plataformas, ya que es un sistema robusto y sin ningún tipo de problema.

Por último, me gustaría añadir que aunque estas tecnologías tienen sus inconvenientes, y quizás no sean la mejor opción de cara al futuro para las siguientes versiones, me han ayudado en la medida de los posible para alcanzar todos y cada uno de los objetivos, proporcionando un prototipo medianamente robusto (salvo por algunas casuísticas que por cuestiones de tiempo no ha sido posible resolverlas) y con unos resultados aceptables.

14. Bibliografía

- [1] D. Takahashi, «Éxito del videojuego Clash Royale,» 15 Febrero 2017. [En línea]. Available: <https://venturebeat.com/2017/02/15/clash-royale-clash-of-clans-push-supercell-to-2-3-billion-in-2016-revenue/>. [Último acceso: 2018].
- [2] Asociación Española de Videojuegos, «El código PEGI,» [En línea]. Available: <http://www.aevi.org.es/documentacion/el-codigo-pegi/>. [Último acceso: 2018].
- [3] E. Avedon y B. Sutton-Smith, *The Study of Games*, J. Wiley, 1971.
- [4] C. Crawford, *The Art of Computer Game Design*, McGraw-Hill/Osborne, 1984.
- [5] «Géneros de videojuegos - Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/List_of_video_game_genres. [Último acceso: 2018].
- [6] Y. Turnes, «Diccionario de géneros,» [En línea]. Available: <http://www.gamerdic.es/tema/generos>. [Último acceso: 2018].
- [7] N. Varonas, «Historia de los motores de videojuegos,» [En línea]. Available: <https://www.neoteo.com/los-motores-graficos-mas-importantes-de-la-historia/>. [Último acceso: 2018].
- [8] Epic Games, «Información sobre Unreal Engine 4,» [En línea]. Available: <https://www.unrealengine.com/en-US/what-is-unreal-engine-4>. [Último acceso: 2018].
- [9] Unity technologies, «Plataformas disponibles para Unity,» [En línea]. Available: <https://unity3d.com/es/public-relations>. [Último acceso: 2018].
- [10] Unity Technologies, «Listado de productos de Unity,» [En línea]. Available: <https://unity3d.com/es/unity>. [Último acceso: 2018].
- [11] Crytek, «Plataformas CryEngine,» [En línea]. Available: <https://www.cryengine.com/features/platforms>. [Último acceso: 2018].
- [12] CryTek, «Productos CryEngine,» [En línea]. Available: <https://www.cryengine.com/get-cryengine/memberships>. [Último acceso: 2018].
- [13] «Metodología SCRUM,» [En línea]. Available: <http://scrummethodology.com/>. [Último acceso: 2018].
- [14] «Bolsa de trabajo y ofertas InfoJobs,» [En línea]. Available: <https://www.infojobs.net/>. [Último acceso: 2018].
- [15] «Bolsa de trabajo y ofertas Domestika,» [En línea]. Available: <https://www.domestika.org>. [Último acceso: 2018].
- [16] R. S. Pressman y B. Maxim, *Software Engineering: A Practitioner's Approach*, McGraw-Hill.
- [17] «Atributos en World of Warcraft,» [En línea]. Available: <http://es.worldofwarcraft.wikia.com/wiki/Atributos>. [Último acceso: 2018].
- [18] Hansen, *Diseño y administración de bases de datos*, Prentice Hall, 2000.
- [19] E. Adams, *Fundamentals of game design*, Pearson New Riders, 2013.
- [20] Unity Technologies, «API multijugador básica de Unity,» [En línea]. Available: <https://docs.unity3d.com/ScriptReference/Networking.NetworkManager.html>. [Último acceso: 2018].
- [21] Unity Technologies, «API multijugador de alto nivel Unity,» [En línea]. Available: <https://docs.unity3d.com/Manual/UNetUsingHLAPI.html>. [Último acceso: 2018].
- [22] Exit Games, «Photon Network Engine,» [En línea]. Available: <https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro>. [Último acceso: 2018].
- [23] Exit Games, «Planes y licencias de Photon,» [En línea]. Available: <https://www.photonengine.com/en-US/PUN/pricing#plan-20>. [Último acceso: 2018].
- [24] Exit Games, «Listado de juegos que usan la tecnología Photon,» [En línea]. Available: <https://www.photonengine.com/en-US/PUN/showcase>. [Último acceso: 2018].
- [25] Exit Games, «Sistema de regiones de la tecnología Photon,» [En línea]. Available: <https://doc.photonengine.com/en-us/pun/current/connection-and-authentication/regions>. [Último acceso: 2018].
- [26] Exit Games, «Sistema de emparejamiento de Photon,» [En línea]. Available: <https://doc.photonengine.com/en-us/pun/current/lobby-and-matchmaking/matchmaking-and-lobby>. [Último acceso: 2018].

- [27] Exit Games, «Instanción en un sistema multijugador con Photon,» [En línea]. Available: <https://doc.photonengine.com/en-us/pun/current/gameplay/instantiation>. [Último acceso: 2018].
- [28] Exit Games, «Sincronización en Photon,» [En línea]. Available: <https://doc.photonengine.com/en-us/pun/current/gameplay/synchronization-and-state>. [Último acceso: 2018].
- [29] Exit Games, «Remote Procedural Call Photon,» [En línea]. Available: <https://doc.photonengine.com/en-us/pun/current/gameplay/rpcsandraiseevent>. [Último acceso: 2018].
- [30] OVH, «Página principal del proveedor de servidores dedicados OVH,» [En línea]. Available: <https://www.ovh.es/>. [Último acceso: 2018].
- [31] OpenBSD, «Información oficial de la herramienta OpenSSH,» [En línea]. Available: <https://www.openssh.com/>. [Último acceso: 2018].
- [32] OVH, «Planes de servidores OVH,» [En línea]. Available: https://www.ovh.es/servidores_dedicados/all_servers.xml. [Último acceso: 2018].
- [33] Google, «Servicio Firebase,» [En línea]. Available: <https://firebase.google.com/products/?hl=es-419>.
- [34] PlayFab, «Servicio PlayFab,» [En línea]. Available: <https://playfab.com/>. [Último acceso: 2018].
- [35] PlayFab, «Características principales del servicio PlayFab,» [En línea]. Available: <https://playfab.com/features/>. [Último acceso: 2018].
- [36] «Información sobre el formato JSON,» [En línea]. Available: <https://www.json.org/json-es.html>. [Último acceso: 2018].
- [37] Unity Technologies, «Definición de GameObject Unity,» [En línea]. Available: <https://docs.unity3d.com/ScriptReference/GameObject.html>. [Último acceso: 2018].
- [38] Unity Technologies, «Orden de eventos en un script de Unity,» [En línea]. Available: <https://docs.unity3d.com/Manual/ExecutionOrder.html>. [Último acceso: 2018].
- [39] Unity Technologies, «ScriptableObject Unity,» [En línea]. Available: <https://docs.unity3d.com/ScriptReference/ScriptableObject.html>. [Último acceso: 2018].
- [40] E. Gamma, R. Helm, J. Vlissides y R. Johnson, *Design Patterns*, Addison-Wesley, 1994.
- [41] K. Sierra y E. Freeman, *Head First Design Patterns*, O'Reilly Media, 2004.
- [42] R. Nystrom, *Game Programming Patterns*, 2014.
- [43] Unity Technologies, «Identificador único de un GameObject,» [En línea]. Available: <https://docs.unity3d.com/ScriptReference/Object.GetInstanceID.html>. [Último acceso: 2018].
- [44] Unity Technologies, «Utilidad JSON proporcionada por Unity,» [En línea]. Available: <https://docs.unity3d.com/Manual/JSONSerialization.html>. [Último acceso: 2018].
- [45] Exit Games, «Documentación de la clase PunBehaviour en la API de Photon,» [En línea]. Available: https://doc-api.photonengine.com/en/pun/current/class_photon_1_1_pun_behaviour.html. [Último acceso: 2018].
- [46] «Plugin Cell Fracture para Blender,» [En línea]. Available: <https://www.blendernation.com/2017/01/26/cell-fracture-blender/>. [Último acceso: 2018].
- [47] Unity Technologies, «Mejores prácticas en las físicas de Unity,» [En línea]. Available: <https://unity3d.com/es/learn/tutorials/topics/physics/physics-best-practices>. [Último acceso: 2018].
- [48] Unity Technologies, «Serialización Unity,» [En línea]. Available: <https://docs.unity3d.com/ScriptReference/Serializable.html>. [Último acceso: 2018].