



Escuela Técnica Superior de Ingeniería Informática

Ingeniería Informática – Ingeniería del Software

TRABAJO FIN DE GRADO

Planificación y desarrollo de un videojuego RPG en Unity

Autor/es:

María Marín Serrano

Tutor/es:

Juan Manuel Cordero Valle

Segunda convocatoria

Curso 2021/2022

RESUMEN

En este trabajo pongo a prueba mis capacidades para realizar un proyecto con tecnologías y lenguajes nuevos para mí, tras cursar todas las asignaturas de Ingeniería del Software.

Para retarme, he elegido crear un videojuego RPG haciendo uso de Unity. En cuanto a videojuegos, aunque tengo experiencia jugándolos, nunca había intentado desarrollar uno por mí misma. He elegido el género RPG por dos razones: la primera, es mi género favorito; la segunda, creo que al tener tantas mecánicas distintas iba a ser un gran reto en cuanto a programación. Y sí, así ha sido.

En esta memoria recojo toda la planificación realizada para llevar a cabo el proyecto, desde el alcance (presentado en forma de GDD, documento típico en el desarrollo de los videojuegos) hasta la planificación temporal o de costes. Tras la planificación, añado un diario de desarrollo para mostrar como he llevado a cabo cada una de las tareas que he realizado para darle vida a este videojuego.

Como conclusión, ha sido un trabajo exigente que he conseguido terminar con éxito gracias a los conocimientos que he adquirido a lo largo de la carrera. He conseguido crear un videojuego RPG que cumple con todos los requisitos necesarios planteados.

Ha sido una experiencia muy bonita poder crear un videojuego como los que llevo jugando desde pequeña y me ha aportado muchos conocimientos que creo que me serán útiles en un futuro.

ÍNDICE

RESUMEN	2
ÍNDICE DE ILUSTRACIONES	7
OBJETIVOS	13
Objetivo del proyecto	13
Objetivo del producto	13
ALCANCE	14
Título del videojuego	14
Desarrollador	14
Género	15
Plataforma	15
Edad recomendada	15
Tecnologías necesarias para el desarrollo	15
Historia	15
¿Qué propicia la historia?	15
¿Qué le pasa a nuestro protagonista para implicarse?	16
Jugabilidad	16
Controles	16
Mecánicas	16
Flujo y menús	34
Público objetivo	36
ADD	36
Personajes	37
Entorno	37
EDT	40
Diccionario de la EDT	41
Planificación	41
Desarrollo	42
Análisis	43
Entrega y presentación	43
REQUISITOS	45
Requisitos funcionales	45
Protagonista	45
Inventario y objetos	47
Enemigos	49
NPCs	50
Interfaz	51

Flujo.....	53
Otros	54
Requisitos no funcionales.....	54
PLANIFICACIÓN TEMPORAL.....	55
Hitos	55
Lista de tareas	55
Preparar el entorno de trabajo.....	55
Familiarizarse con las tecnologías.....	56
Elementos de arte.....	57
Interfaz de usuario	58
Inputs	65
Protagonista	65
Sistema de vida.....	66
Sistema de maná	68
Sistema de experiencia	68
Sistema de estadísticas	69
Sistema de atributos.....	71
Objetos.....	72
Sistema de inventario	73
NPCs.....	74
Sistema de diálogo.....	75
Sistema de dinero	76
Misiones	77
Tienda	79
Crafeo	80
Enemigos	82
Armas	84
Combate.....	88
Loot.....	89
Mapa	91
Flujo.....	94
Música	95
Cronograma	96
Primer Sprint (1 de Febrero de 2022 a 28 de Marzo de 2022)	97
Segundo Sprint (1 de Abril de 2022 a 27 de Mayo de 2022)	99
Tercer Sprint (1 de Junio de 2022 a 26 de Julio de 2022)	101
PLANIFICACIÓN DE COSTES	103

Costes directos	103
Salarios	103
Costes indirectos.....	103
Amortización de los equipos	104
Software.....	104
Formación.....	105
Local de trabajo	105
Coste total	106
ARQUITECTURA DEL SISTEMA	107
Motor gráfico.....	107
Elección del motor.....	107
Configuración del motor.....	108
IDE	111
Programas de arte	111
Copia de seguridad	112
DESARROLLO.....	113
Sprint 1	113
Protagonista	119
Sistema de vida.....	126
Sistema de maná	130
Sistema de experiencia	133
Sistema de estadísticas y atributos	134
Mapa	142
Sprint 2	150
Objetos.....	151
Sistema de inventario	154
NPCs.....	160
Sistema de diálogos	165
Sistema de dinero	171
Enemigos	172
Armas	181
Sistema de combate	187
Sprint 3	192
Sistema de looteo.....	192
Sistema de misiones	197
Sistema de tienda.....	206
Sistema de crafteo.....	210

Flujo y menús	217
Música	222
TRABAJO A FUTURO	226
Plataformas móviles	226
Controles.....	226
Interfaz.....	226
Cámara.....	226
Balancear dificultad.....	226
Otros	227
CONCLUSIONES.....	228
DICCIONARIO.....	229
REFERENCIAS	231

ÍNDICE DE ILUSTRACIONES

Ilustración 1 Boceto vida enemigos.....	16
Ilustración 2 Boceto vida protagonista	17
Ilustración 3 Boceto maná protagonista	17
Ilustración 4 Boceto experiencia protagonista.....	18
Ilustración 5 Boceto panel de accesos	19
Ilustración 6 Boceto panel de estadísticas y atributos	19
Ilustración 7 Boceto panel de accesos	20
Ilustración 8 Boceto panel de estadísticas y atributos	20
Ilustración 9 Boceto dinero protagonista.....	21
Ilustración 10 Boceto panel de accesos	22
Ilustración 11 Boceto inventario	22
Ilustración 12 Boceto inventario extendido.....	23
Ilustración 13 Sprites protagonista idle	23
Ilustración 14 Sprites protagonista corriendo	23
Ilustración 15 Sprites NPC femenino corriendo.....	24
Ilustración 16 Sprites NPC masculino corriendo	24
Ilustración 17 Sprite NPC con gorro	24
Ilustración 18 Sprite NPC anciano	24
Ilustración 19 Boceto panel de diálogo	25
Ilustración 20 Sprite NPC comerciante	25
Ilustración 21 Boceto panel tienda	26
Ilustración 22 Boceto NPC con misiones	26
Ilustración 23 Boceto panel de misiones	27
Ilustración 24 Boceto panel de accesos	27
Ilustración 25 Boceto panel de misiones aceptadas.....	28
Ilustración 26 Boceto panel de misión completa.....	28
Ilustración 27 Boceto panel de crafteo	29
Ilustración 28 Boceto panel de crafteo sin materiales	29
Ilustración 29 Boceto panel de crafteo con materiales	30
Ilustración 30 Boceto panel de arma sin equipar.....	31
Ilustración 31 Boceto panel de arma equipada.....	31
Ilustración 32 Boceto enemigo seleccionado con arma física	31
Ilustración 33 Boceto enemigo seleccionado con arma mágica	32
Ilustración 34 Boceto enemigo muerto.....	32
Ilustración 35 Sprites enemigo slime azul.....	33
Ilustración 36 Sprites enemigo slime verde.....	33
Ilustración 37 Sprites enemigo slime naranja.....	33
Ilustración 38 Sprites enemigo llama blanca	33
Ilustración 39 Sprites enemigo llama roja	33
Ilustración 40 Diagrama mecánicas.....	33
Ilustración 41 Boceto flujo general	34
Ilustración 42 Boceto menú de inicio	34
Ilustración 43 Boceto menú de pausa	35
Ilustración 44 Boceto menú de configuración	35
Ilustración 45 Boceto menú de fin.....	36
Ilustración 46 Sprites protagonista	37
Ilustración 47 Sprite NPC comerciante	37
Ilustración 48 Spritesheet aldeas	38
Ilustración 49 Sprite árbol celeste	38

Ilustración 50 Sprite árbol rojo	39
Ilustración 51 Sprite puente de madera	39
Ilustración 52 Esquema arquitectura	107
Ilustración 53 Panel de descarga de Unity	108
Ilustración 54 Unity Hub iniciar sesión	109
Ilustración 55 Unity Hub descargar Unity	109
Ilustración 56 Unity Hub crear proyecto	110
Ilustración 57 Unity ventanas	111
Ilustración 58 Build Settings plataforma	113
Ilustración 59 Resolución pantalla	114
Ilustración 60 Project Settings resolución	114
Ilustración 61 Entorno Unity parte superior	114
Ilustración 62 Entorno Unity parte inferior	115
Ilustración 63 Entorno Unity parte derecha	115
Ilustración 64 Página de descarga de sprites	116
Ilustración 65 Sprites protagonista	116
Ilustración 66 Carpetas sprites	117
Ilustración 67 Carpeta sprites entorno	117
Ilustración 68 Carpetas sprites objetos	117
Ilustración 69 Carpeta sprites personajes	117
Ilustración 70 Inspector sprites	118
Ilustración 71 Sprite editor	118
Ilustración 72 Jerarquía canvas	119
Ilustración 73 HUD	120
Ilustración 74 Inputs	120
Ilustración 75 Prefab protagonista	121
Ilustración 76 Jerarquía prefab	121
Ilustración 77 Inspector sprite protagonista	122
Ilustración 78 Controlador	122
Ilustración 79 Girar sprite	123
Ilustración 80 RigidBody protagonista	123
Ilustración 81 Animation protagonista	124
Ilustración 82 Animator protagonista	124
Ilustración 83 Inspector Animator transición andar a idle	125
Ilustración 84 Script condiciones Animator	125
Ilustración 85 Inspector protagonista animación	126
Ilustración 86 Script vida	127
Ilustración 87 Script vida protagonista	127
Ilustración 88 Script revivir protagonista	128
Ilustración 89 Carpeta managers	128
Ilustración 90 Script level manager revivir protagonista	128
Ilustración 91 Inspector level manager	129
Ilustración 92 Manager UI en carpeta	129
Ilustración 93 Jerarquía manager interfaz	129
Ilustración 94 Manager interfaz imagen barra vida	130
Ilustración 95 Manager interfaz calcular relleno barra vida	130
Ilustración 96 Inspector vida protagonista	130
Ilustración 97 Script maná gastar y restaurar	131
Ilustración 98 Script maná recuperar y reiniciar	131
Ilustración 99 Manager interfaz vida y maná	132

Ilustración 100 Manager interfaz llenar barra maná.....	132
Ilustración 101 Inspector maná protagonista.....	132
Ilustración 102 Script experiencia añadir experiencia.....	133
Ilustración 103 Manager interfaz experiencia	134
Ilustración 104 Inspector script experiencia.....	134
Ilustración 105 Panel de acceso a menús.....	135
Ilustración 106 Menú estadísticas y atributos.....	135
Ilustración 107 Script estadísticas sumar y restar.....	136
Ilustración 108 Script estadísticas restaurar	136
Ilustración 109 Script boton restaurar estadísticas	137
Ilustración 110 Inspector estadísticas	137
Ilustración 111 Script protagonista estadísticas	137
Ilustración 112 Manager interfaz estadísticas	138
Ilustración 113 Script estadísticas añadir atributos	139
Ilustración 114 Script evento botón atributo	139
Ilustración 115 Script suscripción evento botón atributo	140
Ilustración 116 Respuesta evento atributos	140
Ilustración 117 Inspector botones atributos.....	141
Ilustración 118 Manager interfaz estadísticas y atributos	141
Ilustración 119 Script restaurar estadísticas y atributos	142
Ilustración 120 Inspector estadísticas y atributos	142
Ilustración 121 Jerarquía capas del mapa	143
Ilustración 122 Inspector capas	143
Ilustración 123 Inspector grid.....	144
Ilustración 124 Paletas.....	144
Ilustración 125 Ventana Tile Palette	144
Ilustración 126 Mapa primera aldea	145
Ilustración 127 Mapa primer bosque	146
Ilustración 128 Inspector cámara.....	146
Ilustración 129 Script transición de cámaras	147
Ilustración 130 Inspector collider aldea.....	147
Ilustración 131 Aldea límite y cámara.....	148
Ilustración 132 Parámetros inspector Cinemachine	148
Ilustración 133 Escena parámetros Cinemachine	149
Ilustración 134 Script manager de nivel	150
Ilustración 135 Inspector manager de nivel.....	150
Ilustración 136 Escena punto de respawn.....	150
Ilustración 137 Script objetos métodos de uso	151
Ilustración 138 Script crear objeto encriptable objeto.....	151
Ilustración 139 Script objeto poción vida.....	152
Ilustración 140 Script objeto poción maná	152
Ilustración 141 Carpeta para pociones	152
Ilustración 142 Inspector objeto encriptable poción vida.....	153
Ilustración 143 Inspector objeto encriptable poción maná	153
Ilustración 144 Inspector objeto encriptable superpoción vida	154
Ilustración 145 Inspector objeto encriptable ingrediente carne	154
Ilustración 146 Interfaz inventario	155
Ilustración 147 Interfaz inventario abierto	156
Ilustración 148 Script slot actualizar imagen y cantidad	156
Ilustración 149 Script inventario añadir objeto	157

Ilustración 150 Script slot lanzar evento clicar	157
Ilustración 151 Script slot lanzar evento usar, equipar y quitar	158
Ilustración 152 Script inventario escuchar evento interactuar.....	158
Ilustración 153 Script inventario respuesta evento interactuar.....	159
Ilustración 154 Script interfaz activar slot.....	159
Ilustración 155 Script interfaz descripción objetos	159
Ilustración 156 Spritesheet NPC	160
Ilustración 157 Animation NPC	160
Ilustración 158 Inspector Animator NPC	161
Ilustración 159 Script movimiento gizmos.....	161
Ilustración 160 Script movimiento velocidad.....	162
Ilustración 161 Script movimiento actualizar punto objetivo	162
Ilustración 162 Script movimiento girar personaje	163
Ilustración 163 Inspector movimiento NPC	163
Ilustración 164 Escena movimiento NPC con gizmos	164
Ilustración 165 Escena creación de NPCs	164
Ilustración 166 Jerarquía NPCs	165
Ilustración 167 Inspector NPCs collider.....	165
Ilustración 168 Interfaz diálogo.....	165
Ilustración 169 Interfaz botón diálogo	166
Ilustración 170 Inspector NPC collider trigger	166
Ilustración 171 Script detectar protagonista diálogos	167
Ilustración 172 Script diálogo	167
Ilustración 173 Inspector objeto encriptable diálogo	168
Ilustración 174 Script manager diálogo botón siguiente	168
Ilustración 175 Script manager diálogo continuar diálogo.....	169
Ilustración 176 Script manager diálogo actualizar datos panel	169
Ilustración 177 Script manager diálogo animación texto	170
Ilustración 178 Inspector objeto encriptable diálogo misiones	170
Ilustración 179 Interfaz diálogo actualizada.....	171
Ilustración 180 Script manager dinero	171
Ilustración 181 Inspector manager dinero	172
Ilustración 182 Script manager interfaz actualizar dinero.....	172
Ilustración 183 Spritesheet enemigo	172
Ilustración 184 Animation enemigo	172
Ilustración 185 Inspector animator enemigo	173
Ilustración 186 Inspector movimiento enemigo	173
Ilustración 187 Escena movimiento enemigo.....	174
Ilustración 188 Scripts IA enemigos	174
Ilustración 189 Script IA ejecutar acciones	175
Ilustración 190 Inspector objeto encriptable estado enemigo	175
Ilustración 191 Inspector controlador IA enemigo.....	176
Ilustración 192 Script detectar protagonista en area de ataque	176
Ilustración 193 Script rangos gizmos	177
Ilustración 194 Escena rangos gizmos	177
Ilustración 195 Script movimiento para ataque cargado	178
Ilustración 196 Inspector controlador ataques.....	178
Ilustración 197 Jerarquía enemigos	179
Ilustración 198 Escena enemigos	179
Ilustración 199 Inspector enemigos vida	180

Ilustración 200 Inspector enemigos rigidbody y collider	181
Ilustración 201 Interfaz arma equipada	181
Ilustración 202 Inspector objeto encriptable arma	182
Ilustración 203 Carpeta objetos encriptables armas	182
Ilustración 204 Inspector arma física 1	183
Ilustración 205 Inspector arma física 2	183
Ilustración 206 Inspector arma física 3	184
Ilustración 207 Inspector arma física 4	184
Ilustración 208 Inspector arma mágica 1	185
Ilustración 209 Inspector arma mágica 2	185
Ilustración 210 Inspector arma mágica 3	186
Ilustración 211 Inspector arma mágica 4	186
Ilustración 212 Script interfaz arma equipada	187
Ilustración 213 Script manager seleccionar enemigo arma mágica	187
Ilustración 214 Script manager selección enemigo arma física	188
Ilustración 215 Script selección mágica o física	188
Ilustración 216 Escena enemigo seleccionado con arma física	189
Ilustración 217 Script ataque protagonista usar arma	189
Ilustración 218 Script ataque protagonista calcular daño	189
Ilustración 219 Inspector ataque protagonista	190
Ilustración 220 Inspector pooler daño protagonista	190
Ilustración 221 Escena coordenadas pooler daño protagonista	190
Ilustración 222 Inspector pooler daño enemigos	191
Ilustración 223 Escena pooler daño enemigos	191
Ilustración 224 Interfaz looteo	192
Ilustración 225 Script desactivar enemigo	193
Ilustración 226 Enemigo muerto	193
Ilustración 227 Script objeto looteable	193
Ilustración 228 Script loot enemigo	194
Ilustración 229 Inspector loot enemigo	194
Ilustración 230 Manager looteo cargar en el panel	195
Ilustración 231 Manager looteo cerrar panel	195
Ilustración 232 Interfaz looteo actualizada	195
Ilustración 233 Script recoger objeto loot	196
Ilustración 234 Script añadir experiencia loot	196
Ilustración 235 Script añadir experiencia	197
Ilustración 236 Interfaz misiones disponibles	198
Ilustración 237 Interfaz misiones activas	198
Ilustración 238 Interfaz misión completada	199
Ilustración 239 Script misiones	199
Ilustración 240 Inspector misión	200
Ilustración 241 Script manager cargar misiones	200
Ilustración 242 Script interfaz misiones	200
Ilustración 243 Script descripción misiones	201
Ilustración 244 Inspector manager misiones	201
Ilustración 245 Interfaz misiones actualizada	202
Ilustración 246 Script aceptar misiones	202
Ilustración 247 Script interfaz misiones activas	203
Ilustración 248 Interfaz misiones activas actualizada	203
Ilustración 249 Script manager evento respuesta misión completada	204

Ilustración 250 Script respuesta misión completa	204
Ilustración 251 Interfaz misión completada actualizada	204
Ilustración 252 Script manager reclamar recompensa.....	205
Ilustración 253 Assets misiones	205
Ilustración 254 Inspector misión	205
Ilustración 255 Diálogo misiones	206
Ilustración 256 Jerarquía interfaz tienda	206
Ilustración 257 Interfaz tienda.....	207
Ilustración 258 Inspector manager tienda.....	207
Ilustración 259 Script añadir objetos a la tienda.....	208
Ilustración 260 Script tienda comprar objeto	208
Ilustración 261 Script tienda sumar y restar cantidad.....	209
Ilustración 262 Script interfaz tienda.....	209
Ilustración 263 Script interfaz abrir y cerrar tienda.....	209
Ilustración 264 Interfaz tienda actualizada.....	210
Ilustración 265 Inspector diálogo tienda	210
Ilustración 266 Jerarquía interfaz crafting	211
Ilustración 267 Interfaz crafting	211
Ilustración 268 Inspector lista de recetas.....	212
Ilustración 269 Script manager crafteo cargar recetas.....	212
Ilustración 270 Inspector manager crafteo	213
Ilustración 271 Script manager crafting comprobar materiales.....	214
Ilustración 272 Script manager crafting crear	214
Ilustración 273 Script interfaz crafting	215
Ilustración 274 Interfaz crafting actualizada	215
Ilustración 275 Inspector lista de recetas.....	216
Ilustración 276 Inspector diálogo crafting	216
Ilustración 277 Edificio crafting.....	217
Ilustración 278 Interfaz menú inicio.....	217
Ilustración 279 Interfaz menú pausa.....	218
Ilustración 280 Botón pausa	218
Ilustración 281 Interfaz configuración	218
Ilustración 282 Pantalla de fin	219
Ilustración 283 Inspector menú inicio botón start	220
Ilustración 284 Inspector menú inicio botón settings.....	220
Ilustración 285 Inspector menú pausa botón back to game.....	221
Ilustración 286 Inspector menú pausa botón settings.....	221
Ilustración 287 Inspector menú pausa botón main menu	221
Ilustración 288 Inspector botón pausa	222
Ilustración 289 Inspector menú de configuración botón back	222
Ilustración 290 AudioMixer y carpeta de audio	222
Ilustración 291 Jerarquía manager audio.....	223
Ilustración 292 Inspector manager audio.....	223
Ilustración 293 AudioMixer	224
Ilustración 294 AudioMixer parámetros.....	224
Ilustración 295 Inspector Audio Source	224
Ilustración 296 Script preferencias	225
Ilustración 297 Inspector menu configuración barra volumen.....	225

OBJETIVOS

Podemos diferenciar el objetivo del proyecto y del producto.

Objetivo del proyecto

Mi propósito para este proyecto es desarrollar un videojuego con un motor gratuito, aplicando para ello los conocimientos adquiridos a lo largo de la carrera. En concreto, quiero desarrollar un videojuego RPG.

Además, tengo como objetivo desarrollar este videojuego en un tiempo aproximado de 300 horas de trabajo y terminarlo antes de Septiembre de 2022.

Objetivo del producto

Pretendo desarrollar este videojuego en Unity y que se pueda jugar en ordenadores Windows (al menos) usando teclado y ratón; y que cuente con una estética estilo píxel y colores brillantes. Además, como el videojuego desarrollado será un RPG, quiero que cumpla con algunas características que normalmente están presentes en este tipo de videojuegos: combates, conseguir, comprar y crear objetos, aceptar y completar misiones e interactuar con otros personajes.

ALCANCE

El primer paso para crear un videojuego es que el Game Designer cree un GDDⁱ, que será la base para todo el desarrollo de este.

Un GDD o Game Design Document (en español Documento de Diseño del Juego) es un documento vivo, es decir, que se irá modificando y al que se irá añadiendo información durante el periodo de desarrollo del videojuego. Este documento es una síntesis de todo lo que va a ser el videojuego: la historia, los personajes, el género, el concepto... Es importante que abarque una visión amplia y esté constantemente actualizado para marcar el rumbo del proceso de creación lo más claramente posible. Además, es importante que al leer un GDD quede claro por qué los usuarios deberían jugar al juego.

Este documento puede ser desarrollado en cualquier plataforma, desde en una pizarra en la pared del equipo de desarrollo, hasta en una libreta. Sin embargo, ya que es importante que todos puedan consultarla o editarla, es común (y una buena práctica) que sea un documento digital alojado en un sitio al que todo el equipo de desarrollo tenga acceso.

En cuanto a la estructura de un GDD, no está predefinida, lo importante es que sea clara y el equipo la entienda para poder añadir información o usarla de guía fácilmente. Un ejemplo típico de estructura puede ser el siguiente:

- Título del videojuego
- Nombre del estudio o desarrolladores
- Género
- Edad permitida para jugarlo
- Plataformas en las que estará disponible (puede incluir orden si no estará disponible para todas a la vez)
- Sinopsis o historia completa
- Controles
- Mecánicas
- Tecnologías necesarias para el desarrollo
- Público objetivo

A continuación, desarrollo el GDD de mi proyecto. A demás de servirme de base y guía, he usado para conocer el alcance.

Título del videojuego

Jueguito Monstruito.

Desarrollador

María Marín Serrano.

Un equipo de desarrollo está normalmente formado por distintos roles, de los cuales se encargan distintas personas o equipos. Sin embargo, en este proyecto, seré yo misma quien haga las tareas de todos los distintos roles.

Género

RPGⁱⁱ (Role Playing Game, o en español Juego de Rol). También puede clasificarse como CRPG (Computer Role Playing Game, o en español Juego de Rol para Ordenador), pero este término no suele usarse tanto.

En este género el jugador controla a un personaje en un mundo recreado donde transcurre una historia y debe cumplir misiones para avanzar hasta alcanzar un objetivo.

Plataforma

Ordenador, haciendo uso de teclado y ratón.

Edad recomendada

3 años según PEGIⁱⁱⁱ, 7 años por la posible complejidad de los controles y mecánicas.

PEGI (Pan European Game Information) es el sistema de clasificación por edades oficial de los videojuegos en Europa. Podemos ver esta clasificación en la esquina inferior izquierda de las portadas de los videojuegos. Aunque es una clasificación oficial que esta entidad (no los desarrolladores del videojuego) debe hacer, este juego cumple con los requisitos para obtener el sello de PEGI 3: es apto para todos los públicos, la única “violencia” que contiene son batallas contra monstruos ficticios animados y en las que no hay sangre ni se ven heridas. Por otro lado, este videojuego no contiene ningún otro elemento inapropiado como podría ser la violencia, el lenguaje soez, el uso de drogas o sustos.

Sin embargo, PEGI solo tienen en cuenta estos elementos inapropiados para establecer la edad para jugar a un videojuego, pero no la dificultad de los niños de ciertas edades para controlar el videojuego o seguir la trama; es por eso que, además de la edad PEGI, he recomendado una edad distinta por la posible complejidad.

Tecnologías necesarias para el desarrollo

Usaré el motor de videojuegos Unity para el desarrollo, en concreto la versión Unity 2021.1.22f1, haciendo uso del lenguaje de programación C#.

Historia

¿Qué propicia la historia?

La historia transcurre en un mundo repleto de monstruos. Los humanos viven en pequeñas aldeas donde se sienten seguros, pero no pueden salir de ellas si no quieren ser atacados por estos monstruos. Es por esto que el mundo se siente solitario y los aldeanos han perdido la conexión con los habitantes de otras aldeas. Todo el mundo tiene miedo, sin embargo, nadie parece intentar plantar cara a estos monstruos para conseguir la libertad que tanto ansían y devolver la conexión a las aldeas.

¿Qué le pasa a nuestro protagonista para implicarse?

Nuestra protagonista tiene un sueño en el que aparece un hilo rojo que lo lleva hasta una silueta humana que la llama desde lo que parece otra aldea. Al levantarse, piensa que el sueño ha sido demasiado real y que puede que realmente alguien esté intentando comunicarse con ella. Nuestra protagonista decide averiguar qué significa este misterioso sueño y se arma de valor, intenta perder su miedo a los monstruos y aprende a luchar contra ellos, haciéndose cada vez más fuerte para poder viajar a otras aldeas sin problemas y entender el significado de su sueño.

Jugabilidad

Controles

- W: moverse hacia arriba
- A: moverse hacia la izquierda
- S: moverse hacia abajo
- D: moverse hacia la derecha
- Clic izquierdo: seleccionar un enemigo, interactuar con NPCs cuando tienen un símbolo de “!” o de “?” y con casillas del inventario. También se usa para interactuar con los botones de los menús.
- Espacio: atacar a un enemigo seleccionado

Mecánicas

Sistema de vida

Tanto nuestro personaje principal como los enemigos tienen asignado un atributo de vida, variable y que controla si el personaje sigue vivo o está muerto (se considera que un personaje está muerto si su vida está por debajo de un punto, ya que siempre son números enteros).

En el caso de los enemigos, esta vida se les asigna sin poder aumentar, sólo puede disminuir cuando es atacado por nuestro personaje. Cuando un enemigo muere, su animación y acción se desactiva y su imagen cambia por una sombra. Esta sombra puede “lootearse”, como se explica más adelante. Los enemigos muestran su vida con una barra roja encima de ellos.

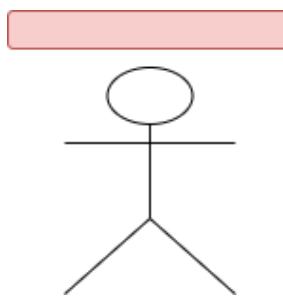


Ilustración 1 Boceto vida enemigos

En cuanto al personaje principal, tiene un atributo de salud máxima cuyo valor predeterminado podemos fijar, y que puede aumentar al subir de nivel. Su atributo de salud actual empieza con el mismo valor que la salud máxima, aunque puede bajar al ser

atacado por un enemigo o aumentar al tomar pociones de vida, sin sobrepasar en ningún caso al valor de salud máxima. Cuando nuestro personaje muere, vuelve al punto de inicio de la partida y su salud actual se iguala a su salud máxima.

Podemos ver el porcentaje de vida de nuestro personaje en el panel de información del personaje que se muestra en la esquina superior izquierda de la pantalla. La barra de vida aparece de color verde.

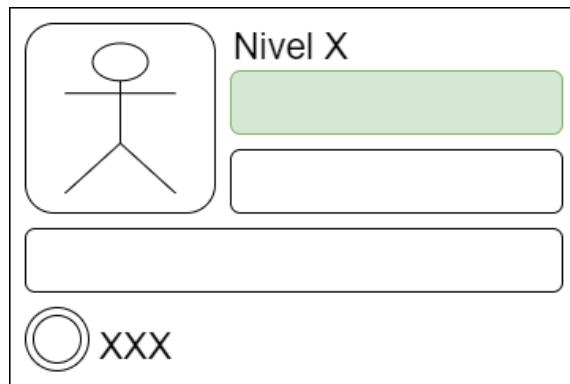


Ilustración 2 Boceto vida protagonista

Sistema de maná

El personaje principal tiene asociado un atributo de maná, al que se le asigna un número entero al crearlo e indica el valor máximo de maná que puede tener nuestro personaje. Este valor máximo no aumenta, aunque el personaje suba de nivel. El valor del maná actual empieza siendo igual al maná máximo y se reduce en X cuando el personaje usa un arma mágica, la cual lleva asignado el número de maná que gasta por cada uso. El personaje puede quedarse sin maná, pero esto no tiene ninguna otra implicación aparte de no poder usar armas mágicas. El valor del maná actual aumenta un punto por segundo hasta igualar al nivel máximo y también puede aumentar usando una poción de maná, sin superar en ningún caso al valor de maná máximo.

Podemos ver el porcentaje de maná de nuestro personaje en el panel de información del personaje que se muestra en la esquina superior izquierda de la pantalla. La barra de maná aparece de color celeste.

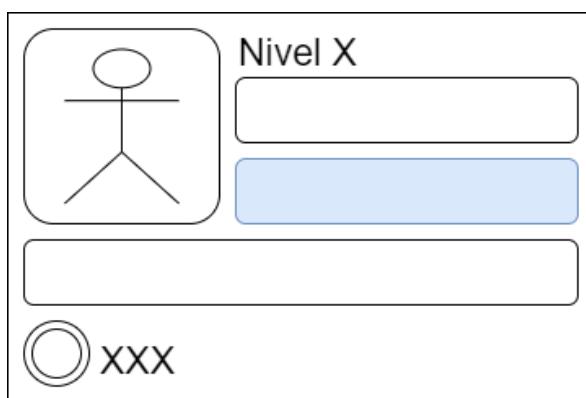


Ilustración 3 Boceto maná protagonista

Sistema de experiencia

Nuestro personaje principal también tiene asociado un atributo de experiencia. Al comenzar el juego, el nivel de nuestro personaje 1 y tiene 0 puntos de experiencia. Podemos definir un nivel máximo que no podrá sobrepasar nuestro personaje y la experiencia que va a necesitar nuestro personaje para alcanzar el siguiente nivel. En este caso, para alcanzar el segundo nivel necesitaremos dos puntos de experiencia y cada vez que subamos un nivel necesitaremos el doble de experiencia que para el nivel anterior. Para conseguir experiencia, podremos completar misiones y matar enemigos.

Podemos ver el nivel de nuestro personaje y la experiencia necesaria para alcanzar el siguiente nivel en el panel de información del personaje que se muestra en la esquina superior izquierda de la pantalla. Esta experiencia aparece en una barra rosa como la mostrada a continuación, donde X representa la experiencia acumulada para el siguiente nivel e Y representa el total de experiencia necesaria para alcanzar el siguiente nivel.

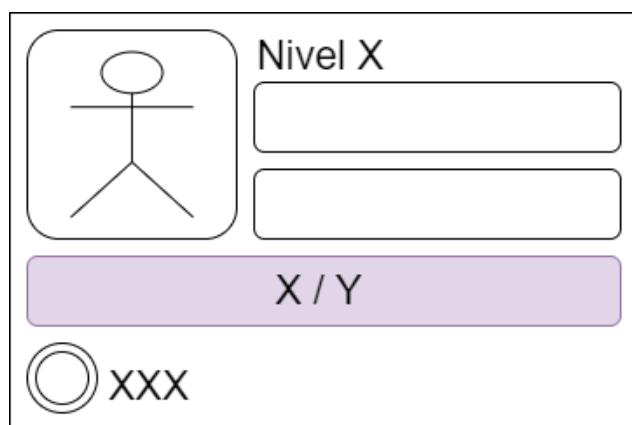


Ilustración 4 Boceto experiencia protagonista

Sistema de estadísticas

Al crear a nuestro personaje principal se le asignan ciertas estadísticas con valores predefinidos que podrán aumentar a medida que nuestro personaje suba de nivel y emplea sus puntos de atributos o si se equipa armas con bonus de estadísticas, en cuyo caso se suman los valores de las armas con los de nuestro personaje. Las estadísticas vuelven a su valor inicial si nuestro personaje muere.

Estas estadísticas, además de incluir el nivel y la experiencia mencionadas anteriormente, incluyen otros datos importantes para nuestro personaje:

- Daño: puntos de vida que quita al enemigo.
- Defensa: los puntos de defensa se restan a los puntos de daño del enemigo antes de restar salud al personaje.
- Velocidad: cuanto más alto el número, menos tiempo tiene que esperar nuestro personaje entre dos ataques seguidos.
- Porcentaje de crítico: probabilidad de que un ataque haga más daño del normal.
- Porcentaje de bloqueo: probabilidad de esquivar el ataque de un enemigo sin que te haga daño.

Podemos consultar las estadísticas de nuestro personaje accediendo al menú de estadísticas y atributos. Este menú se encuentra en el panel situado en la parte inferior izquierda de la pantalla, representado con un brazo musculoso, y se abre (y cierra) haciendo clic en el botón del brazo.



Ilustración 5 Boceto panel de accesos

Una vez abierto el menú podemos consultar las estadísticas en la parte superior.

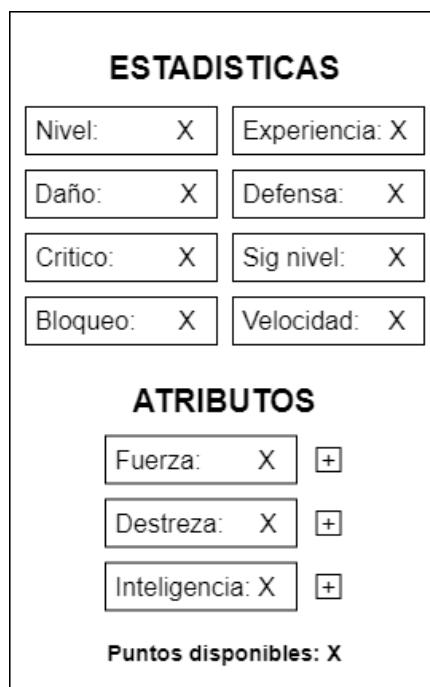


Ilustración 6 Boceto panel de estadísticas y atributos

Sistema de atributos

Nuestro personaje principal tiene asociado un sistema de atributos que le permite incrementar sus estadísticas. Al subir de nivel, nuestro personaje adquiere puntos que puede usar para incrementar sus atributos. El número de puntos conseguidos por nivel es una variable, pero en este caso vamos a establecerlo en 3 unidades por nivel. Al inicio del juego, el valor de estos puntos será de 0 y al morir se pierden.

Los puntos se pueden sumar a cada uno de los tres atributos disponibles, y al usarlos se restan de los puntos disponibles. Al aumentar el nivel de un atributo, sube las estadísticas asociadas a ese atributo. En concreto, cada atributo sube las siguientes estadísticas:

- Fuerza:
 - o Daño +2
 - o Defensa +1
 - o Porcentaje de bloqueo +0.3

- Destreza:
 - o Porcentaje de bloqueo +0.2
 - o Defensa +2
 - o Velocidad +1
- Inteligencia:
 - o Daño +3
 - o Porcentaje de bloqueo +0.1
 - o Porcentaje de crítico +0.3

Podemos consultar los atributos de nuestro personaje accediendo al menú de estadísticas y atributos. Este menú se encuentra en el panel situado en la parte inferior izquierda de la pantalla, representado con un brazo musculoso, y se abre (y cierra) haciendo clic en el botón del brazo.



Ilustración 7 Boceto panel de accesos

Una vez abierto el menú podemos consultar e incrementar los atributos en la parte inferior.

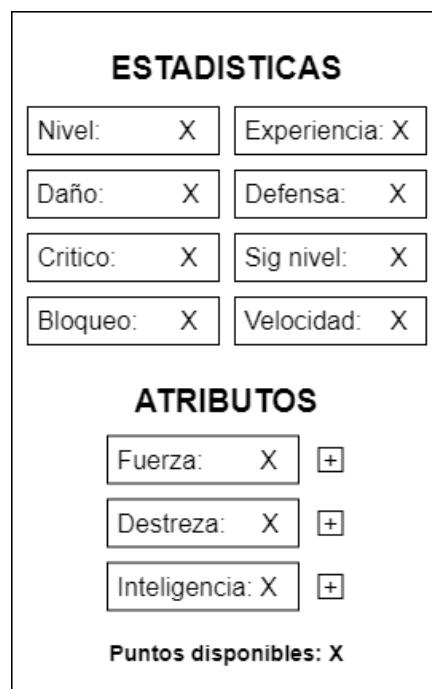


Ilustración 8 Boceto panel de estadísticas y atributos

Sistema de dinero

La mecánica del dinero es importante en este juego, pues lo necesitaremos para conseguir objetos que hagan más fuerte a nuestro personaje principal. En este juego la divisa serán las monedas de oro y empezaremos la partida con 10. Podemos aumentar nuestra cantidad de oro completando misiones y, si tenemos suerte, también podremos conseguirlo al matar algunos enemigos.

Podemos usar el oro en las tiendas para obtener armas y objetos, en cuyo caso se restará la cantidad de oro usada de nuestro oro total.

El oro no se resetea al morir, por lo que podemos acumularlos, aunque nos maten y tengamos que empezar desde el punto de partida.

Podemos ver el oro del que disponemos en el panel de información del personaje que se muestra en la esquina superior izquierda de la pantalla. Esta cantidad aparece a la derecha de un ícono de moneda.

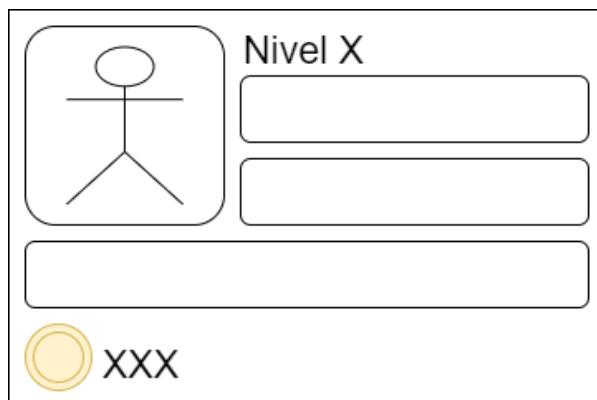


Ilustración 9 Boceto dinero protagonista

Inventario

El personaje principal tendrá asociado un inventario con un tamaño predeterminado de 24 huecos, al que se le asignarán objetos de inventario cuando el personaje obtenga nuevos objetos. Al empezar la partida, el inventario estará vacío. Además, si el personaje muere, no perderá los objetos de inventario que tenga almacenados en él.

En cuanto a los objetos de inventario, existen de distintos tipos: armas, pociones, ingredientes y otros objetos. Cada objeto debe tener asociado un ID, un nombre, una imagen y una descripción. Además, podremos marcar si ese objeto se puede acumular en stack (varios en el mismo hueco del inventario) indicando en ese caso su número máximo de acumulación, y si es consumible (como en el caso de las pociones, para poderlas usar y que desaparezcan del inventario o su número se reduzca). Un objeto tendrá asignadas, además, algunas características especiales si son del tipo consumible o arma:

- Consumible: efecto que causa sobre el personaje al consumirlo, como recuperar ciertos puntos de vida o maná
- Arma: características que se suman a las del personaje, mientras se tenga el arma equipada, para las batallas: daño, probabilidad de crítico...

Existen dos tipos de armas, físicas y mágicas, que se desarrollarán en el apartado de combate.

Podemos acceder al inventario a través del panel situado en la parte inferior izquierda de la pantalla, representado con un zurrón. El menú se abre y se cierra al hacer clic en este botón.

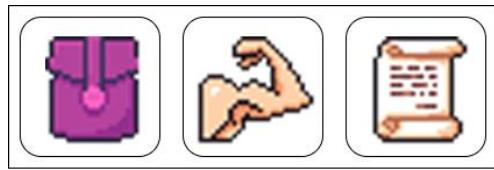


Ilustración 10 Boceto panel de accesos

Una vez accedemos al menú, podremos ver los huecos y los objetos de inventario almacenados. También tendremos acceso a tres botones llamados “usar”, “equipar” y “quitar” que podremos usar cuando estemos seleccionando un objeto de inventario dentro de este. El botón de “usar” se podrá usar sobre objetos consumibles para tener acceso a sus beneficios, mientras el botón “equipar” se podrá usar sobre objetos de tipo armas para asociarlas a nuestro personaje y que haga uso de ellas (solamente puede tener equipada un arma a la vez). El botón “quitar” nos desequipará el arma que llevemos equipada en ese momento.

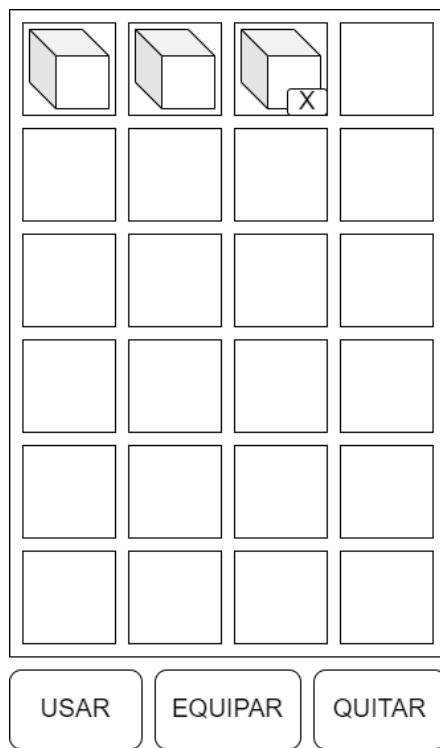


Ilustración 11 Boceto inventario

Para navegar por los objetos del inventario y ver sus características, simplemente tendremos que hacer clic sobre ellos. Al seleccionar un objeto, su casilla cambiará de color para indicar que está seleccionada y se abrirá una ventana con los datos de este objeto.

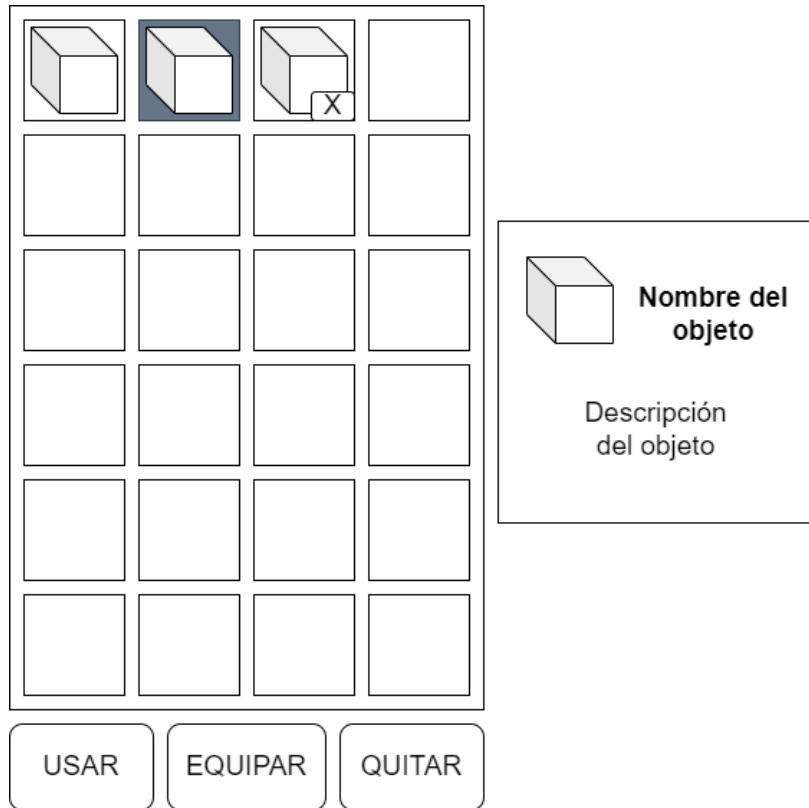


Ilustración 12 Boceto inventario extendido

Protagonista

Además de tener asociados los elementos explicados anteriormente de vida, maná, experiencia, estadísticas, atributos e inventario; el protagonista podrá moverse por el mapa e interactuar con cosas a su alrededor.

Idle

Cuando el protagonista no está en movimiento, realiza una animación de espera, imitando una respiración mientras está de pie.



Ilustración 13 Sprites protagonista idle

Moverse

El protagonista puede moverse libremente por el mundo mientras realiza una animación simulando correr.



Ilustración 14 Sprites protagonista corriendo

Interactuar

Puede interactuar con NPCs, enemigos y otros elementos a través de su collider y el de los elementos interactivos.

NPCs

Los NPC (Non Playable Character, o Personaje No Jugable en español) pueden aparecer en las escenas de forma estática o moviéndose. En el caso de aparecer de forma estática, no tienen animación de Idle como el protagonista; sin embargo, si aparecen moviéndose por la escena, si tienen su propia animación. Algunos ejemplos de animaciones de NPCs:



Ilustración 15 Sprites NPC femenino corriendo



Ilustración 16 Sprites NPC masculino corriendo

Algunos ejemplos de personajes estáticos:



Ilustración 17 Sprite NPC con gorro



Ilustración 18 Sprite NPC anciano

Diálogos

Los diálogos se pueden asignar a los NPCs, aunque no todos los NPCs tienen por qué tener un diálogo asignado. Los diálogos están formados por una lista de textos ordenados, al iniciar un diálogo sale el primer texto de la lista. Al darle al botón siguiente, se muestra el siguiente texto de la lista. Cuando no quedan textos por mostrar en ese diálogo, el botón siguiente cierra el menú del diálogo. Este menú de diálogo también muestra una imagen del NPC con el que estamos interactuando y su nombre.

Al crear el diálogo, además de asignarle un NPC y una lista de textos, podemos asignarle (de forma opcional) una interacción extra a elegir entre “misiones”, “tienda” y “crafting”. Estas interacciones se detallarán en su propio apartado.

Al asignarle un diálogo a un NPC, se le añade un botón encima que solo aparece cuando nuestro protagonista se acerca a este NPC. Este botón puede ser una “!” si es un diálogo normal o con interacción extra del tipo tienda o crafting, o una “?” si el diálogo tiene una interacción extra del tipo “misiones”.

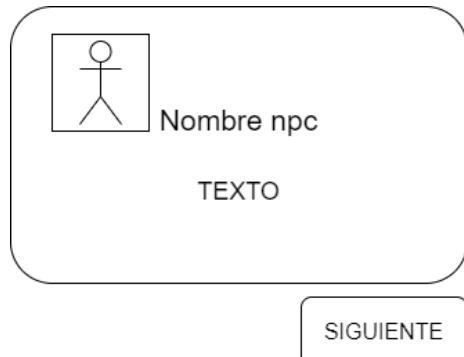


Ilustración 19 Boceto panel de diálogo

[Tienda](#)

Algunos NPCs tendrán asociados un diálogo con interacción extra del tipo tienda y podremos comprarles objetos. Para que se reconozca fácilmente qué personajes tienen este tipo de interacción, usaré siempre el mismo NPC llamativo sin importar que salga duplicado en varias escenas del juego:



Ilustración 20 Sprite NPC comerciante

Tendremos un manager para la tienda que nos permita agregar al inventario de la tienda los objetos que queramos que puedan comprar los jugadores. Para añadir estos objetos a la tienda, seleccionaremos qué objeto es (creado anteriormente), el nombre y su precio de venta. Cuando el personaje acceda a la tienda, verá la lista de los objetos disponibles con su imagen, nombre y precio. El usuario podrá seleccionar el número de objetos que quiere comprar de cada tipo usando los botones “+” y “-” al lado del objeto y luego haciendo clic en el botón comprar. No se podrán comprar objetos por un valor mayor al dinero disponible.

Para finalizar la compra, se debe hacer clic en un botón en la esquina superior derecha de la tienda con una “x”.

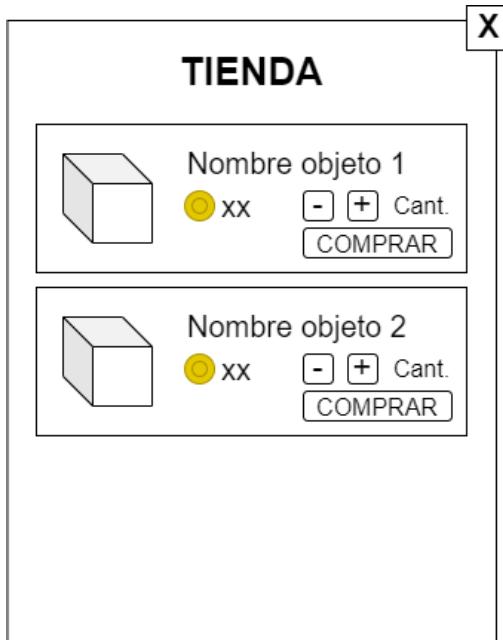


Ilustración 21 Boceto panel tienda

Misiones

Algunos NPCs tendrán asociados un diálogo con interacción extra del tipo misiones y podremos verlas y aceptarlas para obtener recompensas. Para que se reconozca fácilmente qué personajes tienen este tipo de interacción, cuando nuestro personaje se acerque a ellos aparecerá una "?" en la cabeza de estos NPCs.

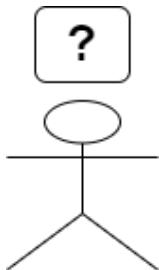


Ilustración 22 Boceto NPC con misiones

Las misiones serán creadas como objetos encriptables, y tendrán un id asociado, un nombre, una cantidad de enemigos a matar para completarla, una descripción y las recompensas asociadas. Las recompensas siempre serán un número de experiencia, otro de oro y, además, un número de objetos que se añadirán directamente al inventario tras completarla. Para controlar el progreso de la misión, se sumará un punto cada vez que matemos a un enemigo y se comprobará si la cantidad de enemigos matados es igual o mayor a la cantidad necesaria para cumplir la misión.

Una vez hablemos con el NPC con misiones, veremos un menú con las misiones disponibles. Este menú debe mostrar, para cada misión, su nombre, descripción y recompensa. Además, debe haber un botón de aceptar en cada una de las misiones.

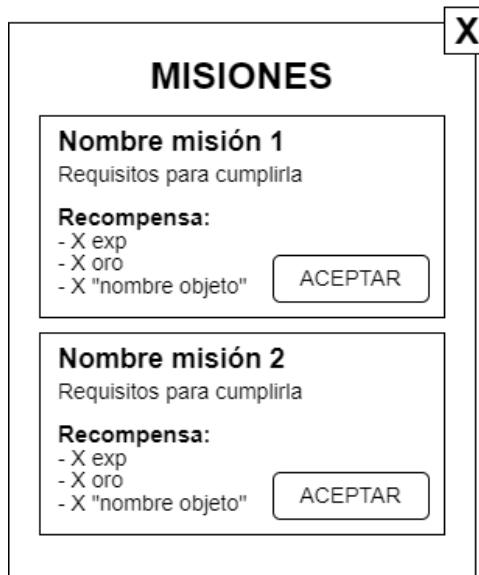


Ilustración 23 Boceto panel de misiones

Cuando las misiones sean aceptadas, desaparecerán del menú de misiones del NPC y podremos verla en nuestro menú de misiones activas. Podemos acceder al menú de misiones activas a través del panel situado en la parte inferior izquierda de la pantalla, representado con un pergamo. El menú se abre y se cierra al hacer clic en este botón.



Ilustración 24 Boceto panel de accesos

El menú de misiones activas debe mostrar, para cada misión activa, su nombre, descripción, las recompensas y el progreso de la misión (cuantos enemigos hemos matado y el total de enemigos que debemos matar para cumplir esa misión).

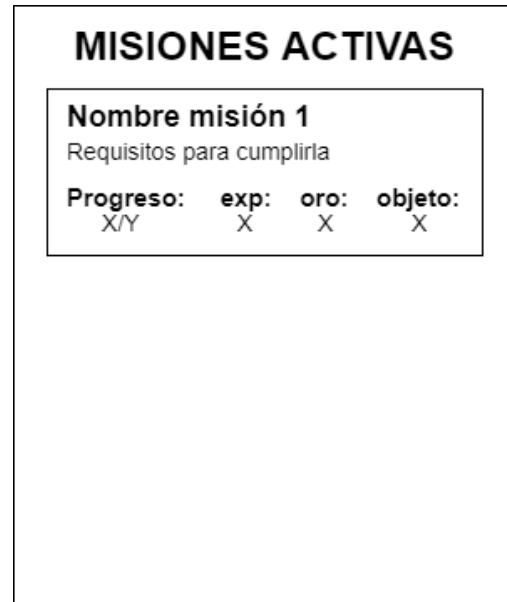


Ilustración 25 Boceto panel de misiones aceptadas

Cuando una misión se haya completado, aparecerá un panel emergente en la pantalla para avisarnos. Este panel mostrará el nombre de la misión completada y las recompensas de esta misión. Además, aparecerá un botón de reclamar para obtener las recompensas y cerrar el panel.



Ilustración 26 Boceto panel de misión completa

Craftear

Algunos NPCs tendrán asociados un diálogo con interacción extra del tipo craftear. Al hablar con él, nos permitirá usar objetos de nuestro inventario para crear objetos nuevos. Para controlar qué objetos podrá craftear el personaje principal y cómo, se crearán recetas. Estas recetas tendrán asociados 2 objetos necesarios para crear el nuevo objeto junto a la cantidad que necesitaremos de cada uno, además de tener asociado el objeto resultado y su cantidad.

Cuando hablemos con el NPC que nos permita craftear, veremos una ventana con las recetas disponibles.

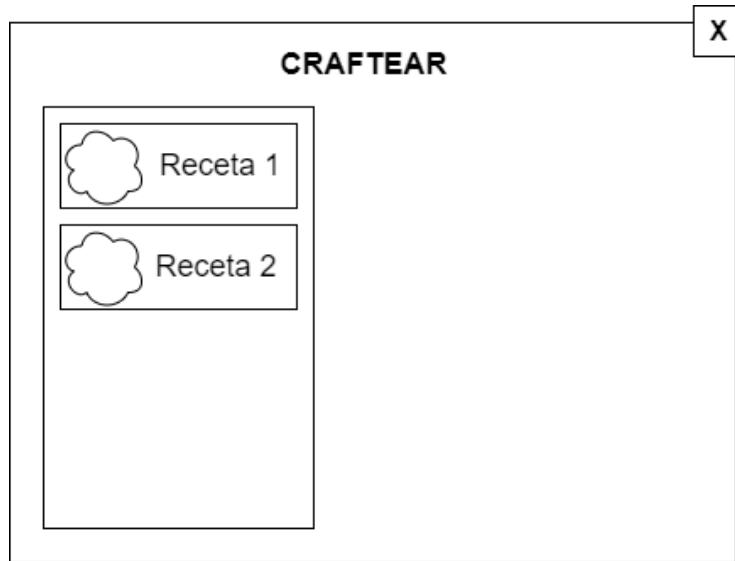


Ilustración 27 Boceto panel de crafteo

Para obtener más información de las recetas y poder crearlas, podemos hacer clic sobre cada una de ellas. Al hacer clic, veremos a la derecha los objetos necesarios para crearlas junto a la cantidad necesaria y la cantidad que tenemos. También veremos el objeto resultante con su descripción y un panel que cambiará en función de si tenemos o no los materiales necesarios.

Si no tenemos suficientes materiales, nos avisará y no podremos crear el objeto resultante.

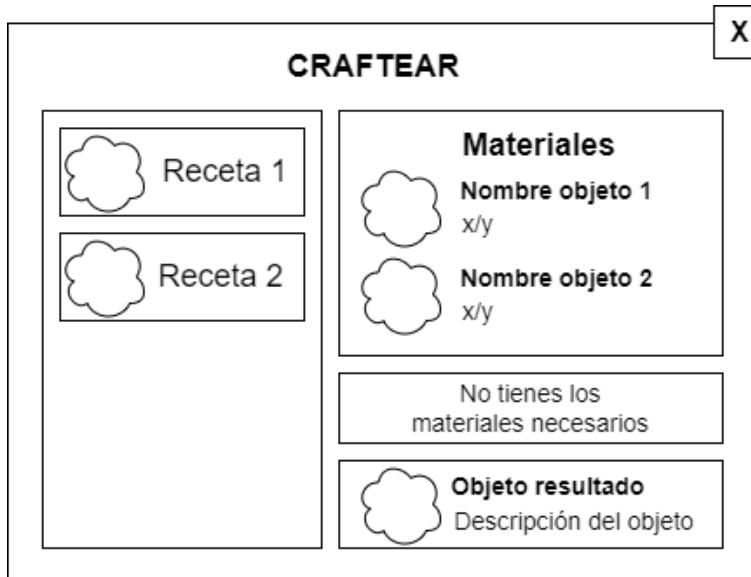


Ilustración 28 Boceto panel de crafteo sin materiales

Sin embargo, si tenemos los objetos necesarios, veremos un botón de “craft”. Al pulsarlo, crearemos el objeto resultante. Este objeto aparecerá en nuestro inventario y a la vez desaparecerán de él los objetos que hemos usado.

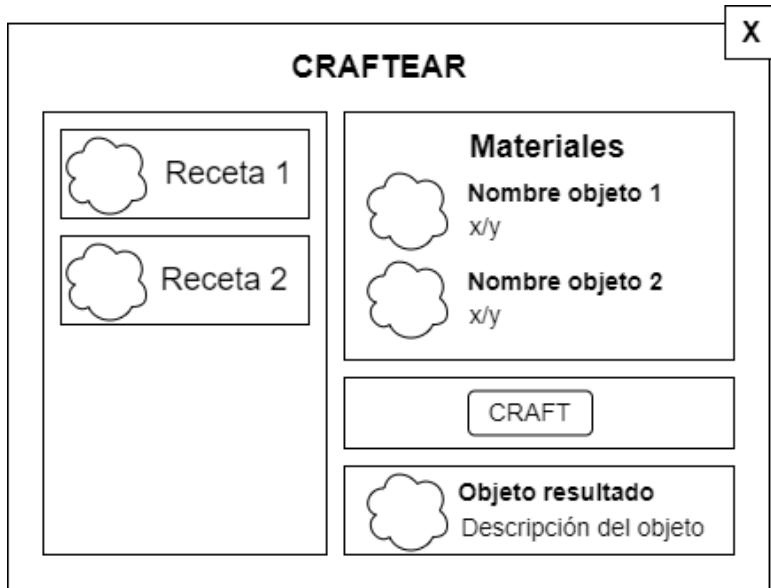


Ilustración 29 Boceto panel de crafteo con materiales

Podemos cerrar el menú de crafteo haciendo clic en la “x” de la parte superior derecha del propio menú.

Combate

Combatir contra enemigos nos permitirá hacerles daño hasta matarlos y ser dañados por ellos hasta morir.

Para que un enemigo nos dañe basta con entrar en su rango de visión y ser perseguido por él. Sin embargo, para que nuestro personaje pueda dañar a un enemigo, debe tener una arma equipada además de entrar en el rango de combate.

Existen dos tipos de armas, las mágicas y las físicas. Las armas físicas no gastarán puntos de maná pero nos exigirá estar más cerca de los enemigos para poder atacar. Por otro lado, las armas mágicas nos permitirán dañar a los enemigos guardando cierta distancia a cambio de usar puntos de maná. No podremos usar estas armas cuando no nos queden puntos de maná.

Todas las armas tienen asociado un número de daño que se suma al de nuestro personaje. Cuanto más avanzada la historia o más difícil sea de conseguir un arma, más daño otorgará al personaje.

A lo largo de un combate, además de poder ver la vida restante del enemigo y del personaje, podremos ver la cantidad de daño hecho en cada golpe; pues saldrán animaciones al lado del enemigo y del personaje con la cantidad de daño que ha recibido en cada golpe.

El personaje solo puede tener una arma equipada a la vez. Si se quiere equipar una arma distinta, primero debemos desequipar la arma anterior. Podemos ver si tenemos una arma equipada (y cual) en la esquina superior derecha de la pantalla. Si el personaje no tiene ninguna arma equipada, el cuadrado de la esquina aparecerá vacío.

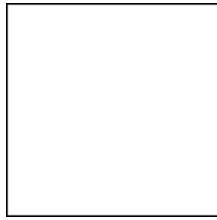


Ilustración 30 Boceto panel de arma sin equipar

Sin embargo, si tenemos un arma equipada, en este cuadrado veremos la imagen de esta arma.

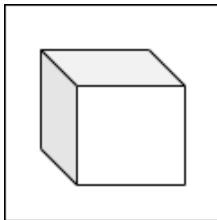


Ilustración 31 Boceto panel de arma equipada

Enemigos

Existirán distintos enemigos contra los que tendremos que luchar para poder fortalecer a nuestro protagonista. Realmente, no es necesario entrar en combate contra ellos y se podrían intentar esquivar, pero cuanto más avance la historia los enemigos que aparezcan serán más fuertes y peligrosos, por lo que es mejor ir preparados.

Para poder nivelar la dificultad del juego y de los enemigos, se programarán varios scripts asociados a estos, de forma que se puedan ajustar sus atributos y adecuarlos a cada tramo de la historia.

Movimiento

Al colocar un enemigo sobre un escenario, podremos hacer que se mueva por ella siguiendo unos patrones. Esto se hará indicándole el número de puntos del escenario por donde queremos que pase y las coordenadas de estos. Además, se podrá añadir la velocidad a la que se moverá por este recorrido.

Selección

Cuando nuestro personaje se acerque a un enemigo teniendo equipada un arma física, este enemigo se marcará automáticamente con un círculo rojo, indicando que podemos atacarle.

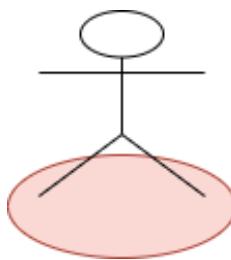


Ilustración 32 Boceto enemigo seleccionado con arma física

Sin embargo, si el arma es mágica, no tendremos que acercarnos tanto al enemigo, pero sí tendremos que hacerle clic para seleccionarlo antes de atacarle.

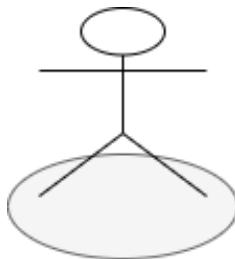


Ilustración 33 Boceto enemigo seleccionado con arma mágica

Si el personaje no lleva equipada ningún arma, no veremos ninguna marca de selección en los enemigos y no podremos defendernos.

IA

Los enemigos tendrán asociados varios parámetros que cambiarán su forma de comportarse. Entre ellos encontramos el rango de detección. Cuando el personaje entra en el rango de detección de un enemigo, el enemigo dejará de andar por su recorrido predeterminado y perseguirá al personaje hasta que se aleje lo suficiente como para salir de este rango. En ese caso, el enemigo volverá a su recorrido.

Además de este rango, podremos definir para cada enemigo su cantidad de daño, su velocidad de ataque, si su ataque es de tipo básico (desde cerca) o cargado (desde lejos), el rango desde el que pueden atacarte y su vida.

Loot

Podremos asociar a los enemigos un botín. Este botín tendrá asociada una cantidad de experiencia que conseguiremos de forma inmediata al vencer al enemigo. Además, al ser vencidos, los enemigos se desactivan y aparece una sombra gris en el lugar donde los hemos matado. Podemos interactuar con esta sombra para recoger otros objetos del botín que hemos asociado a este enemigo.



Ilustración 34 Boceto enemigo muerto

Para asignar objetos al botín de cada enemigo, tendremos que indicar qué objetos son de los creados anteriormente, su nombre, la cantidad que recibiremos de este objeto, y el porcentaje de probabilidad de que este objeto aparezca en el botín del enemigo al morir.

Sprites

Los enemigos tendrán animaciones mientras andan por el escenario y nos atacan. En cuanto al aspecto, se podrá elegir entre varios prefabricados, aunque es lo menos importante de los enemigos, ya que independientemente de su aspecto tendrán atributos más o menos fuertes dependiendo del momento del juego, no de su apariencia.

Algunos ejemplos de enemigos:



Ilustración 35 Sprites enemigo slime azul



Ilustración 36 Sprites enemigo slime verde



Ilustración 37 Sprites enemigo slime naranja



Ilustración 38 Sprites enemigo llama blanca



Ilustración 39 Sprites enemigo llama roja

Diagrama de mecánicas

Adjunto un diagrama a modo de resumen de las mecánicas y como interactúan entre ellas, a grandes rasgos para que sea más visual.

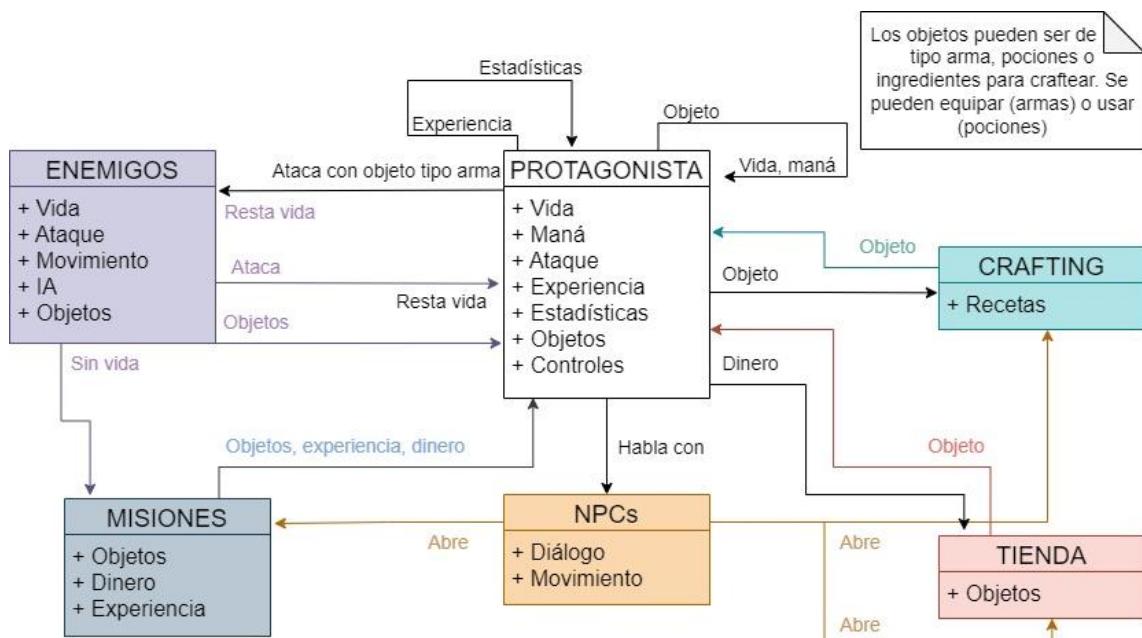


Ilustración 40 Diagrama mecánicas

Flujo y menús

Cuando abramos el juego veremos una pantalla de inicio desde la que podremos acceder a configuraciones o comenzar la partida. Cuando hayamos iniciado la partida, podremos abrir el menú de pausa desde el que podremos acceder a configuraciones o volver al menú de inicio. Cuando estemos jugando y lleguemos al final de la partida, veremos un menú de fin desde donde podremos volver a la partida de inicio.

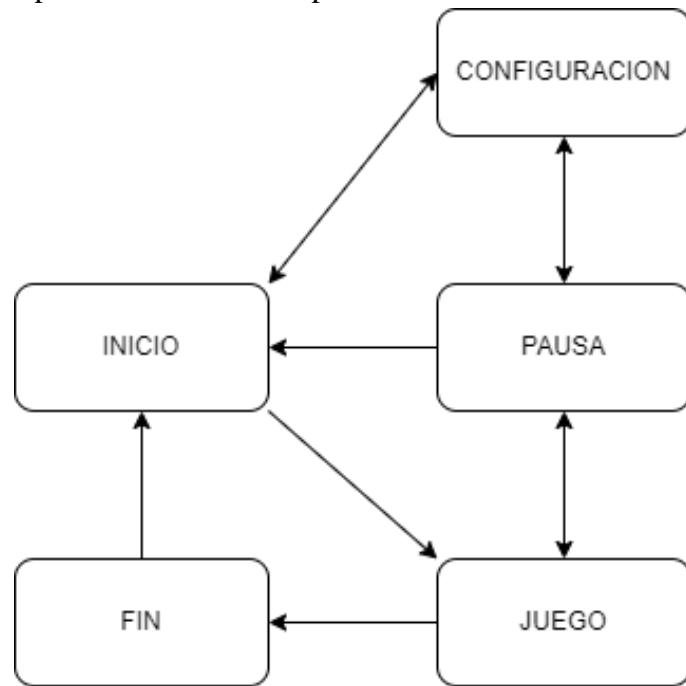


Ilustración 41 Boceto flujo general

Menú de inicio

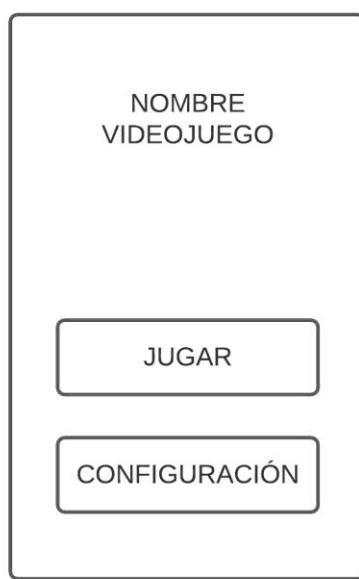


Ilustración 42 Boceto menú de inicio

Menú de pausa



Ilustración 43 Boceto menú de pausa

Menú de configuración

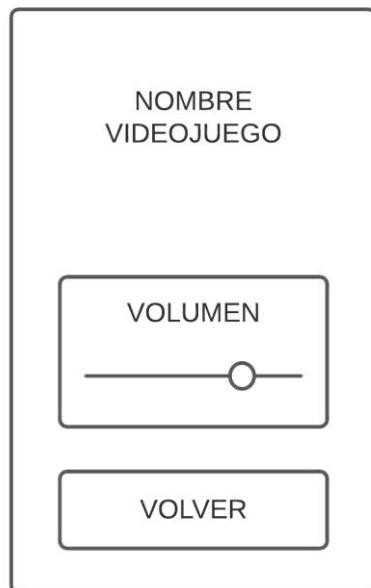


Ilustración 44 Boceto menú de configuración

Menú de fin

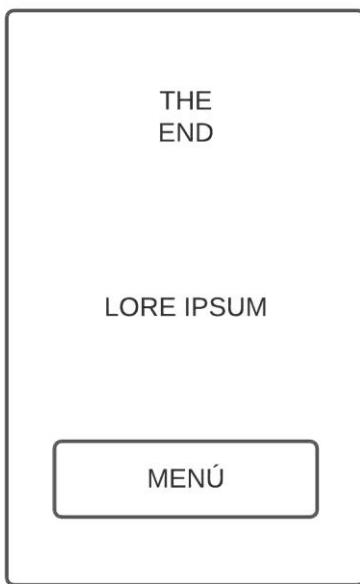


Ilustración 45 Boceto menú de fin

Público objetivo

El juego está pensado para llegar a los amantes de videojuegos RPG de todas las edades. Este género atrae una gran audiencia; sin embargo, los videojuegos RPG suelen ser demasiado largos y a veces cuesta compaginar el tiempo que se les dedica con el estilo de vida actual en el que muchas personas emplean demasiado tiempo en estudiar o trabajar.

Desarrollar un videojuego RPG corto para ordenador (plataforma a la que casi todo el mundo tiene acceso) le dará la oportunidad a esta audiencia de disfrutar una entretenida partida en cualquier momento en el que tenga un rato libre.

ADD

El videojuego será desarrollado para ordenador, por lo que tanto los niveles como las interfaces se desarrollarán pensando en que deben encajar bien en pantallas horizontales.

En cuanto al estilo del arte, se desarrollará en 2D y con arte píxel. Para esto se usarán spritesheets que encajen con la estética (rural) y usando la herramienta de tilemap para facilitar el trabajo. También se usarán spritesheets para los personajes y enemigos y sus correspondientes animaciones.

La interfaz no seguirá la estética pixel para diferenciarla bien de los espacios del juego y los personajes. Sin embargo, seguirá la estética del videojuego usando colores acordes, como tonos madera o tierra.

Personajes

Protagonista

La protagonista del videojuego está basada en mí, la desarrolladora del mismo. Por lo tanto, tiene el pelo rosa, es poco sociable y tiene muchos miedos. Sin embargo, siente gran empatía por los seres no humanos.

Para ella, luchar contra los monstruos simboliza luchar con sus miedos y atreverse a dar un paso adelante para crecer como persona. Sin embargo, esta tarea es muy dolorosa por ella debido a la empatía que siente por estos seres.

La idea de que el protagonista sea femenino, además de representarme a mí, es un símbolo de protesta contra los estereotipos de género: los protagonistas suelen ser masculinos y el público masculino tiene facilidad para identificarse con ellos, pero es hora de darle la misma sensación a la otra mitad de la audiencia.



Ilustración 46 Sprites protagonista

Comerciantes

Personaje típico en los juegos de rol que ayudan al desarrollo del protagonista vendiéndole objetos. Habrá uno en cada aldea, tendrán todos el mismo aspecto y serán de una raza no humana.



Ilustración 47 Sprite NPC comerciante

Otros NPCs

Deben seguir la misma estética que el personaje principal y no tener un aspecto demasiado llamativo, así la atención se centrará en el protagonista. Otros aspectos sobre los NPCs son irrelevantes.

Entorno

El entorno en general tiene un aspecto rural, hay naturaleza por todos sitios y parece un poco antiguo. Las construcciones suelen ser de madera.

Aldeas

En todas las aldeas podemos encontrar elementos comunes como un comerciante, una taberna o herrería para craftear y alguien que nos ofrezca misiones.

Algunos ejemplos de construcciones con la estética a seguir para las aldeas:

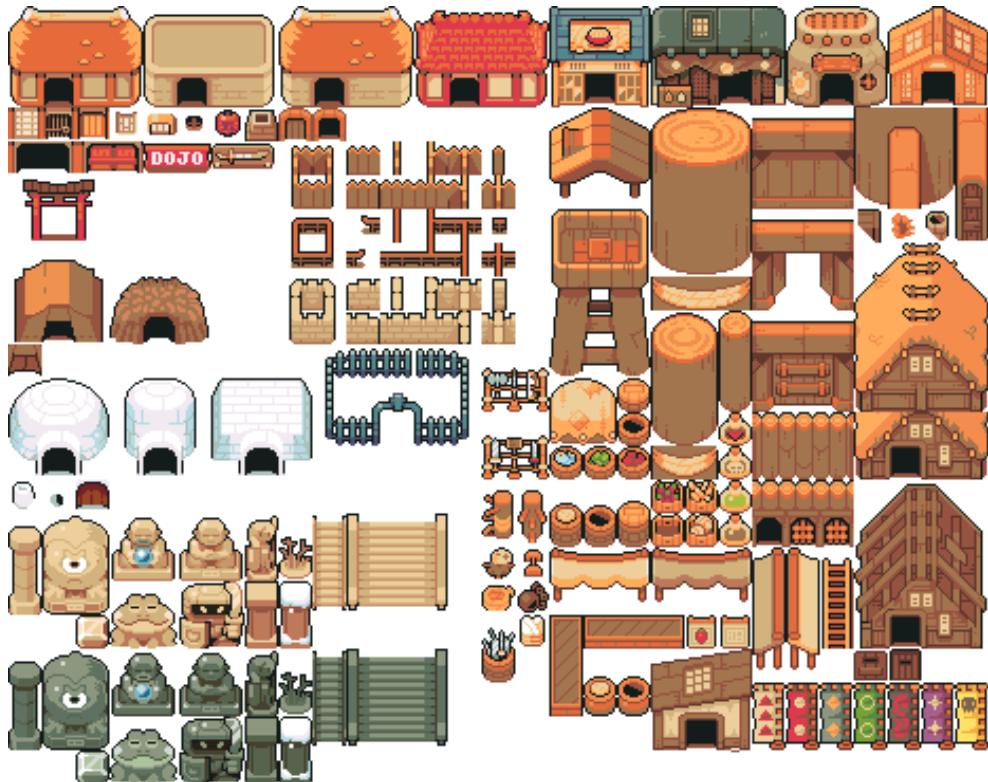


Ilustración 48 Spritesheet aldeas

Bosques

Sirven de rutas entre aldeas. Tienen la misma estética rural con mucha naturaleza, pero no tienen apenas construcciones, ya que los humanos no utilizan estos espacios. Solo encontraremos monstruos en estos sitios.

Algunos ejemplos de elementos con la estética a seguir para las zonas de campo:



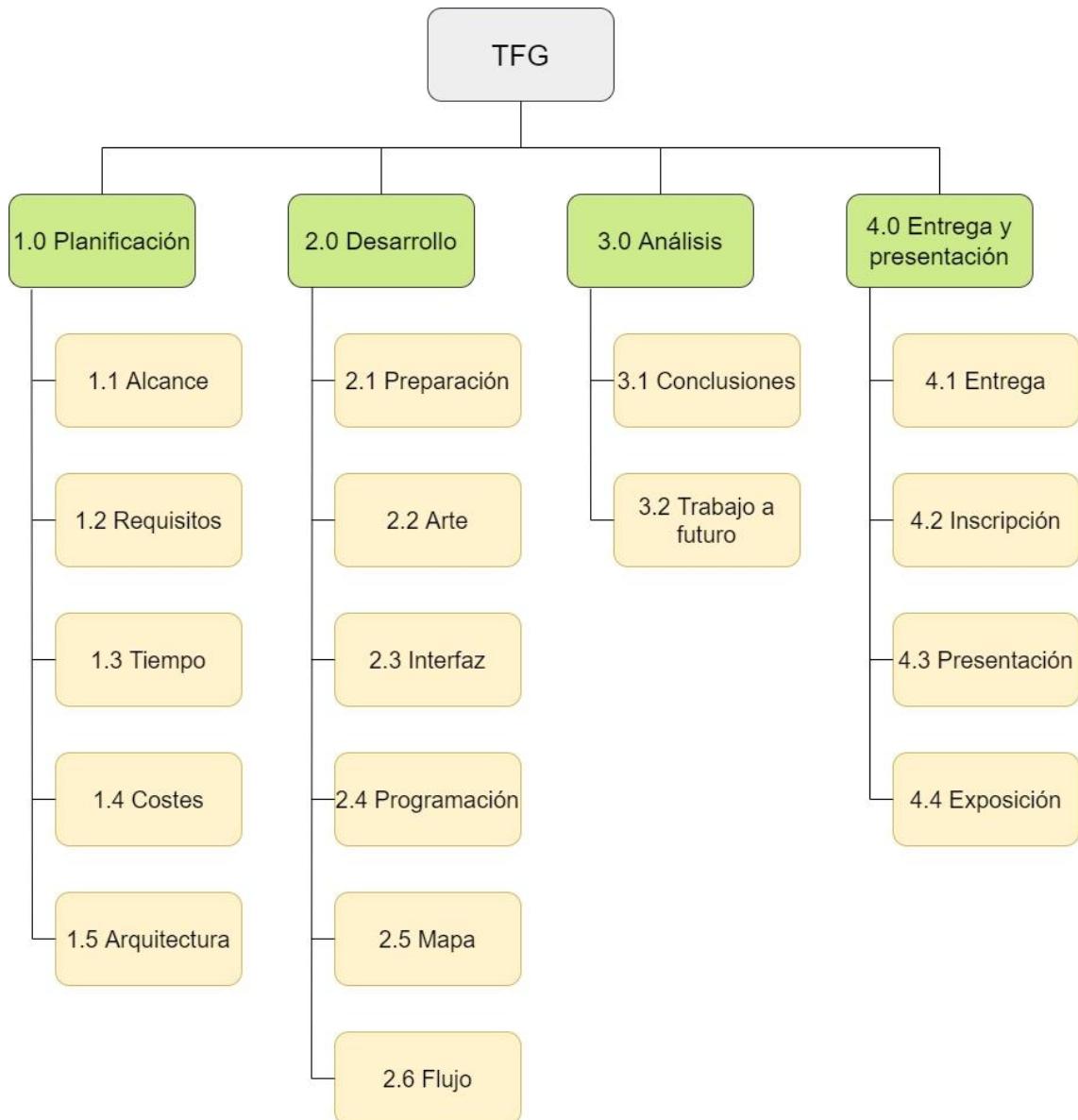
Ilustración 49 Sprite árbol celeste



Ilustración 50 Sprite árbol rojo



Ilustración 51 Sprite puente de madera



Diccionario de la EDT

Planificación

Entrada	1.1 Alcance
Descripción	Documento que define de forma simple los objetivos que se intentan alcanzar en el desarrollo del proyecto. En el caso de los videojuegos, el GDD es una buena forma de definir el alcance.
Estimación	1 semana
Riesgos	No estimar bien los requisitos ni organizar bien el tiempo y el coste del proyecto. No organizar las tareas de forma adecuada.

Entrada	1.2 Requisitos
Descripción	Expectativas que tenemos de un proyecto, propósitos que debe cumplir para que se considere exitoso.
Estimación	3 días
Riesgos	Si no se hace una lista de requisitos y se comprueba que el proyecto las cumpla, puede que tras ser terminado no cumpla con nuestras expectativas.

Entrada	1.3 Tiempo
Descripción	Definir el tiempo del que disponemos para la realización del proyecto, marcar las fechas importantes y los hitos, estimar las tareas necesarias para el desarrollo del proyecto y el tiempo que se tardará en realizarlas.
Estimación	1 semana
Riesgos	Atraso en algunas tareas, incluso no entregar el proyecto a tiempo.

Entrada	1.4 Costes
Descripción	Definir todos los recursos, humanos y no humanos, que necesitamos para desarrollar el proyecto. Una vez estén claros los recursos necesarios, calcular el coste de cada uno de ellos para conocer el coste total del proyecto.
Estimación	2 días
Riesgos	Gastar más dinero del necesario para el desarrollo del proyecto.

Entrada	1.5 Arquitectura
Descripción	Definir la arquitectura del entorno necesario para desarrollar el proyecto, qué softwares necesitamos y las configuraciones previas que debemos hacer.
Estimación	2 días
Riesgos	No contar con los medios para desarrollar el proyecto definido.

Desarrollo

Entrada	2.1 Preparación
Descripción	Preparar el entorno de trabajo definido previamente, interactuar con él para familiarizarnos y estudiar el funcionamiento de las tecnologías necesarias si no las hemos usado anteriormente, como es este caso.
Estimación	2 semanas
Riesgos	No poder desarrollar las tareas de forma satisfactoria o tardar demasiado en el desarrollo de las mismas.

Entrada	2.2 Arte
Descripción	Estudiar y definir el estilo artístico que queremos para el proyecto. Descargar las imágenes y los audios necesarios que cumplan con las características que hemos definido anteriormente. Usar estas imágenes para hacer los tilemaps y las animaciones que necesitaremos para el proyecto. Añadir audio al juego.
Estimación	2 semanas
Riesgos	Que el proyecto, aunque funcione correctamente, no tenga la estética que esperábamos.

Entrada	2.3 Interfaz
Descripción	Realizar los diseños de la interfaz, desarrollarlos en los canvas de Unity y comunicar esta interfaz con el juego para que sea usable.
Estimación	2 semanas
Riesgos	Que el videojuego desarrollado no sea accesible o usable, que sea incómodo de jugar o que incluso fracase por no poder ser jugado.

Entrada	2.4 Mapa
Descripción	Diseño previo del mapa en el que deseamos concurra la historia del juego. Crear las distintas partes de este mapa con los tilemaps preparados anteriormente. Añadir elementos interactivos al mapa, como NPCs y enemigos.
Estimación	1 semana
Riesgos	Que el jugador no pueda avanzar en el videojuego.

Entrada	2.5 Programación
Descripción	Crear los scripts necesarios para que funcionen, de forma adecuada y óptima, todas las mecánicas deseadas en el videojuego.
Estimación	4 meses
Riesgos	Mala optimización, errores en el videojuego.

Entrada	2.6 Flujo
Descripción	Poder pasar por todos los estados del juego, desde la pantalla de inicio hasta el menú de fin.
Estimación	1 semana
Riesgos	No poder pausar o configurar el juego, o incluso no poder empezarlo o terminarlo.

Análisis

Entrada	3.1 Lecciones aprendidas
Descripción	Conocimiento adquirido a lo largo del desarrollo del proyecto, tanto positivo como negativo.
Estimación	3 días
Riesgos	Volver a cometer errores pasados, no mejorar.

Entrada	3.2 Trabajo a futuro
Descripción	Especificar tareas que se pueden hacer tras la finalización del proyecto para mejorarlo.
Estimación	2 días
Riesgos	Ninguno

Entrega y presentación

Entrada	4.1 Entrega
Descripción	Entregar al tutor el trabajo terminado.
Estimación	1 día
Riesgos	No tener la aprobación para inscribir el proyecto.

Entrada	4.2 Inscripción
Descripción	Inscribir el trabajo terminado y aceptado por el tutor para poder hacer la presentación y exponerlo.
Estimación	1 día
Riesgos	No aprobar.

Entrada	4.3 Presentación
Descripción	Preparar la presentación de diapositivas que se utilizarán para exponer el trabajo.
Estimación	1 semana
Riesgos	No poder exponer el trabajo.

Entrada	4.4 Exposición
Descripción	Exponer el trabajo realizado y los resultados del mismo.
Estimación	1 día
Riesgos	No aprobar.

REQUISITOS

Requisitos funcionales

Protagonista

Requisito	RF001 – Moverse
Descripción	El protagonista debe poder moverse libremente por el mapa usando los botones del teclado A, S, D, W (hacia la izquierda, abajo, derecha y arriba respectivamente).

Requisito	RF002 – Idle
Descripción	El protagonista debe tener una animación de espera cuando esté quieto.

Requisito	RF003 – Correr
Descripción	El protagonista debe tener una animación de correr cuando se esté moviendo por el mapa.

Requisito	RF004 – Poder interactuar
Descripción	El protagonista debe tener un collider para poder interactuar con el medio.

Requisito	RF005 – Interactuar con NPC
Descripción	El protagonista debe poder interactuar con los NPC haciendo clic izquierdo con el ratón.

Requisito	RF006 – Seleccionar enemigos
Descripción	El protagonista debe poder seleccionar enemigos cuando lleva equipada un arma mágica con el clic izquierdo del ratón, y se deben seleccionar automáticamente cuando lleva equipada un arma física y se acerca a este.

Requisito	RF007 – Atacar
Descripción	El protagonista debe poder atacar a los enemigos seleccionados usando la barra espaciadora si tiene un arma equipada.

Requisito	RF008 – Aumentar dinero
Descripción	El protagonista debe poder conseguir dinero completando misiones.

Requisito	RF009 – Gastar dinero
Descripción	El protagonista debe poder usar su dinero para comprar objetos en la tienda.

Requisito	RF010 – Usar maná
Descripción	El protagonista debe poder usar maná para hacer uso de las armas mágicas.

Requisito	RF011 – Restaurar maná
Descripción	El protagonista debe poder restaurar maná con el paso del tiempo y usando pociones.

Requisito	RF012 – Perder vida
Descripción	El protagonista debe poder perder vida al ser atacado por un enemigo. Se debe mostrar la vida que el enemigo le ha quitado al protagonista.

Requisito	RF013 – Restaurar vida
Descripción	El protagonista debe poder restaurar vida al usar pociones.

Requisito	RF014 – Morir
Descripción	El protagonista debe morir si su vida baja de un punto.

Requisito	RF015 – Revivir
Descripción	Tras morir, el protagonista debe revivir volviendo a un punto de partida y resemando sus estadísticas.

Requisito	RF016 – Ganar experiencia
Descripción	El protagonista debe poder aumentar su experiencia matando enemigos y completando misiones.

Requisito	RF017 – Subir de nivel
Descripción	El protagonista debe poder aumentar su nivel cuando acumule cierta experiencia.

Requisito	RF018 – Ganar atributos
Descripción	El protagonista debe poder conseguir puntos de atributos cada vez que suba de nivel.

Requisito	RF019 – Aumentar estadísticas
Descripción	Las estadísticas de combate del protagonista, tales como su daño, defensa, velocidad, probabilidad de crítico o probabilidad de bloqueo; deben poder aumentar usando puntos de atributos.

Requisito	RF020 – Restablecer estadísticas
Descripción	Las estadísticas de combate del protagonista deben restablecerse cuando este muera.

Requisito	RF021 – Aceptar misiones
Descripción	El protagonista debe poder aceptar misiones.

Requisito	RF022 – Completar misiones
Descripción	El protagonista debe poder progresar en las misiones y completarlas matando al número de enemigos necesario. Al completar las misiones, el protagonista debe recibir recompensas.

Requisito	RF023 – Comprar
Descripción	El protagonista debe poder comprar objetos disponibles de la tienda por un valor igual o menor al del dinero que tiene.

Requisito	RF024 – Craftear
Descripción	El protagonista debe poder crear objetos nuevos usando materiales de su inventario.

Inventario y objetos

Requisito	RF025 – Inventario
Descripción	El protagonista debe tener acceso a un inventario donde almacenar objetos, ver sus descripciones e interactuar con ellos.

Requisito	RF026 – Equipar objeto
Descripción	El protagonista debe poder equiparse un objeto de tipo arma desde el inventario. Para equiparse un arma no puede tener otra equipada previamente. Al equiparse un arma, sus estadísticas se suman a las del protagonista.

Requisito	RF027 – Desequipar objeto
Descripción	El protagonista debe poder desequiparse un objeto de tipo arma desde el inventario.

Requisito	RF028 – Usar objeto
Descripción	El protagonista debe poder usar los objetos de tipo consumible y recibir inmediatamente sus beneficios.

Requisito	RF029 – Poción vida
Descripción	Debe existir un objeto consumible que restaure 20 puntos de vida del protagonista.

Requisito	RF030 – Poción maná
Descripción	Debe existir un objeto consumible que restaure 20 puntos de maná del protagonista.

Requisito	RF031 – Superpoción vida
Descripción	Debe existir un objeto consumible que restaure 60 puntos de vida del protagonista.

Requisito	RF032 – Superpoción maná
Descripción	Debe existir un objeto consumible que restaure 60 puntos de maná del protagonista.

Requisito	RF033 – Materiales
Descripción	Deben existir objetos varios no equipables ni consumibles que sirvan de material para crear objetos nuevos.

Requisito	RF034 – Arma física muy débil
Descripción	Debe existir un arma, equipable, de tipo físico, que otorgue al protagonista 4 puntos de ataque.

Requisito	RF035 – Arma física débil
Descripción	Debe existir un arma, equipable, de tipo físico, que otorgue al protagonista 8 puntos de ataque.

Requisito	RF036 – Arma física fuerte
Descripción	Debe existir un arma, equipable, de tipo físico, que otorgue al protagonista 20 puntos de ataque.

Requisito	RF037 – Arma física muy fuerte
Descripción	Debe existir un arma, equipable, de tipo físico, que otorgue al protagonista 50 puntos de ataque.

Requisito	RF038 – Arma mágica muy débil
Descripción	Debe existir un arma, equipable, de tipo mágico, que otorgue al protagonista 5 puntos de ataque a cambio de 4 puntos de maná.

Requisito	RF039 – Arma mágica débil
Descripción	Debe existir un arma, equipable, de tipo mágico, que otorgue al protagonista 12 puntos de ataque a cambio de 10 puntos de maná.

Requisito	RF040 – Arma mágica fuerte
Descripción	Debe existir un arma, equipable, de tipo mágico, que otorgue al protagonista 26 puntos de ataque a cambio de 15 puntos de maná.

Requisito	RF041 – Arma mágica muy fuerte
Descripción	Debe existir un arma, equipable, de tipo mágico, que otorgue al protagonista 60 puntos de ataque a cambio de 40 puntos de maná.

Enemigos

Requisito	RF042 – Animaciones enemigos
Descripción	Los enemigos tendrán asignadas animaciones de caminar cuando se estén moviendo por el mapa.

Requisito	RF043 – Movimiento enemigos
Descripción	Los enemigos deben poder moverse siguiendo caminos asignados previamente.

Requisito	RF044 – IA enemigos
Descripción	Los enemigos deben perseguir al protagonista cuando entre en su rango de visión y deben poder atacarle cuando entre en su rango de ataque.

Requisito	RF045 – Dañar enemigos
Descripción	Los enemigos deben perder vida al ser atacados por el protagonista. Se debe mostrar la vida que el protagonista le ha quitado al enemigo.

Requisito	RF046 – Matar enemigos
Descripción	El protagonista debe poder matar a los enemigos cuando la vida de estos baje de un punto. Los enemigos al morir se transforman en sombras.

Requisito	RF047 – Saquear enemigos
Descripción	El protagonista debe poder saquear a los enemigos muertos haciendo clic izquierdo en su sombra ver los objetos que contiene y recogerlos también con el clic izquierdo.

NPCs

Requisito	RF048 – Animaciones NPCs
Descripción	Los NPCs tendrán asignadas animaciones de caminar cuando se estén moviendo por el mapa.

Requisito	RF049 – Movimiento NPCs
Descripción	Los NPCs deben poder moverse siguiendo caminos asignados previamente.

Requisito	RF050 – Diálogos
Descripción	Los NPCs deben poder tener asignado un diálogo que mostrarán cuando el protagonista interactúe con ellos.

Requisito	RF051 – Interacciones extras
Descripción	Los diálogos de los NPCs podrán tener interacciones extras que nos den opción a acceder a la tienda, a las misiones o a craftear.

Requisito	RF052 – NPCs tienda
Descripción	En cada aldea debe haber, al menos, un NPC que nos permita acceder a la tienda.

Requisito	RF053 – NPCs misiones
Descripción	En cada aldea debe haber, al menos, un NPC que nos permita acceder a las misiones.

Requisito	RF054 – NPCs crafteo
Descripción	En cada aldea debe haber, al menos, un NPC que nos permita acceder a craftear.

Interfaz

Requisito	RF055 – IU inicio
Descripción	Debe existir un menú de inicio al abrir el videojuego.

Requisito	RF056 – IU pausa
Descripción	Se debe poder pausar el juego mediante un botón que esté presente siempre que estemos jugando, accediendo a la pantalla de pausa.

Requisito	RF057 – IU configuración
Descripción	Se debe poder acceder al menú de configuración de los ajustes del juego.

Requisito	RF058 – IU HUD
Descripción	Debe existir un panel que esté presente mientras el juego esté activo en el que se muestren los datos del protagonista, tales como su nivel, experiencia, vida, maná y dinero.

Requisito	RF059 – IU inventario
Descripción	Se debe poder acceder desde el juego a un panel donde ver el inventario del protagonista.

Requisito	RF060 – IU equipamiento
Descripción	Debe existir un panel que esté presente mientras el juego esté activo que muestre si el protagonista lleva equipada una arma y cuál es.

Requisito	RF061 – IU estadísticas
------------------	--------------------------------

Descripción	Se debe poder acceder desde el juego a un panel donde ver las estadísticas y atributos del protagonista.
-------------	--

Requisito	RF062 – IU diálogos
Descripción	Se debe poder acceder desde el juego a un panel donde se muestren las conversaciones con los NPC.

Requisito	RF063 – IU misiones
Descripción	Se debe poder acceder desde el juego a un panel donde ver y aceptar las misiones.

Requisito	RF064 – IU misiones aceptadas
Descripción	Se debe poder acceder desde el juego a un panel donde ver las misiones aceptadas y el progreso de cada una.

Requisito	RF065 – IU misión completa
Descripción	Cuando una misión se complete, se debe mostrar un panel indicando el fin de esta misión y debe permitirnos recoger las recompensas correspondientes.

Requisito	RF066 – IU tienda
Descripción	Se debe poder acceder desde el juego a un panel donde comprar objetos a cambio de dinero.

Requisito	RF067 – IU crafteo
Descripción	Se debe poder acceder desde el juego a un panel que nos permita usar los objetos de nuestro inventario para crear objetos nuevos.

Requisito	RF068 – IU loot
Descripción	Se debe poder acceder desde el juego a un panel donde ver los objetos que tenían los enemigos ya muertos y poder recogerlos.

Flujo

Requisito	RF069 – Inicio
Descripción	Al arrancar el juego, debemos ver el menú de inicio. Desde este menú debemos poder acceder a jugar y al menú de configuración. También debemos poder acceder a este menú desde el menú de pausa y desde el menú de fin.

Requisito	RF070 – Configuración
Descripción	Debemos poder acceder al menú de configuración desde inicio y desde pausa. Desde este menú debemos poder modificar el volumen y volver al menú de inicio o de pausa.

Requisito	RF071 – Juego
Descripción	Debemos poder comenzar a jugar desde el menú de inicio. Desde el juego, debemos poder acceder al menú de pausa. Tras completar el juego, se debe acceder automáticamente al menú de fin.

Requisito	RF072 – Fin
Descripción	Debemos poder acceder al menú de fin al terminar la partida. Desde este menú debemos poder acceder al menú de inicio

Requisito	RF073 – Pausa
Descripción	Debemos poder acceder al menú de pausa desde dentro del juego. Desde este menú debemos poder acceder al menú de inicio, al de configuración y volver a la partida.

Requisito	RF074 – Aldeas
Descripción	Durante la partida, el protagonista debe poder pasar por, al menos, tres aldeas. En estas aldeas se encontrarán NPC.

Requisito	RF075 – Bosques
Descripción	Durante la partida, el protagonista debe poder pasar por, al menos, cinco bosques. En estos bosques se encontrarán enemigos.

Requisito	RF076 – Edificio crafteo
Descripción	En cada aldea debe haber, al menos, un edificio para craftear. En este edificio se encontrará el NPC que nos permita craftear.

Otros

Requisito	RF077 – Música
Descripción	El juego debe tener música.

Requisito	RF078 – Cámara
Descripción	La cámara debe perseguir al protagonista.

Requisitos no funcionales

Requisito	RNF001 - Unity
Descripción	El videojuego debe desarrollarse con el motor Unity 2021.1.22f1.

Requisito	RNF002 - Plataforma
Descripción	El videojuego debe poder jugarse en Windows 10.

Requisito	RNF003 - Controles
Descripción	El videojuego debe poder manejarse usando ratón y teclado de ordenador.

Requisito	RNF004 - Estética
Descripción	El arte del videojuego debe ser simple, y tanto los personajes como los mapas deben ser estilo píxel.

Requisito	RNF005 - Protagonista
Descripción	La protagonista debe ser un personaje femenino con el pelo rosa.

PLANIFICACIÓN TEMPORAL

Tras realizar el alcance del proyecto, se necesita organizar las tareas necesarias para su desarrollo a lo largo del tiempo disponible, es decir, en 300 horas (horas equivalentes a los 12 créditos que componen el Trabajo de Fin de Grado).

El comienzo del desarrollo será el día 1 de Febrero de 2022, y realizaré jornadas de trabajo de 2 a 3 horas durante 20 días al mes hasta el día 31 de Julio de 2022. En total, serán 120 jornadas de 2,5 horas de media de trabajo cada una.

Organizando el proyecto de esta forma, debe estar terminado para cumplir con las fechas establecidas para la segunda convocatoria. Estas fechas son las siguientes:

- 1 de Septiembre de 2022: entrega de la memoria
- 2 de Septiembre de 2022: inscripción
- 7 de Septiembre de 2022: entrega de la presentación
- 12 al 16 de Septiembre de 2022: exposición oral

Hitos

Además de las fechas establecidas para esta convocatoria, he establecido fechas para organizar el desarrollo del trabajo. Dividiré las 100 jornadas de trabajo en 3 sprint, de forma que cada sprint cuente con 100 horas de trabajo. Las fechas establecidas como tope para cada uno de los sprint son las siguientes

- 28 de Marzo de 2022: fin del primer Sprint
- 27 de Mayo de 2022: fin del segundo Sprint
- 26 de Julio de 2022: fin del tercer Sprint y entrega

A continuación, la lista de tareas necesarias explicadas junto a la estimación del tiempo que llevará desarrollarlas y los requisitos que satisface cada una.

Lista de tareas

Preparar el entorno de trabajo

Tarea	1.1 Instalar Unity
Descripción	Descargar e instalar Unity Hub, crear una cuenta de Unity e instalar Unity 2021.1.22f1.
Tareas anteriores	-
Estimación	½ jornada
Satisface	RNF001

Tarea	1.2 Instalar Aseprite
Descripción	Descargar e instalar Aseprite.
Tareas anteriores	-
Estimación	½ jornada
Satisface	RNF005

Familiarizarse con las tecnologías

Tarea	2.1 Familiarizarse con Unity
Descripción	Crear un proyecto de prueba con Unity desde 0, probar las distintas herramientas que este motor nos ofrece y familiarizarnos con los elementos de la interfaz.
Tareas anteriores	1.1 Instalar Unity
Estimación	2 jornadas
Satisface	RNF001

Tarea	2.2 Familiarizarse con Aseprite
Descripción	Probar las distintas funcionalidades que ofrece Aseprite y hacer algunos sprites con este programa para familiarizarnos.
Tareas anteriores	1.2 Instalar Aseprite
Estimación	½ jornada
Satisface	RNF005

Tarea	2.3 Estudio de C#
Descripción	Estudiar las nociones básicas del lenguaje de programación C#. Hacer algunos scripts sencillos con este lenguaje dentro de Unity para familiarizarnos y comprender las herramientas básicas.
Tareas anteriores	1.1 Instalar Unity
Estimación	3 jornadas
Satisface	RNF001

Tarea	2.4 Organizar el entorno de Unity
Descripción	Para organizar el futuro trabajo y ser ordenados desde el principio, crear las carpetas donde guardaremos los distintos tipos de assets: <ul style="list-style-type: none"> - Animaciones - Prefabs - Escenas - Scripts - Scriptable Objects - Sonidos - Sprites
Tareas anteriores	2.1 Familiarizarse con Unity
Estimación	½ jornada
Satisface	-

Elementos de arte

Tarea	3.1 Descargar sprites
Descripción	Buscar y descargar spritesheets de estilo píxel que sean coherentes entre ellos, con estilo rural, de los que se puedan extraer elementos como campos, ríos, flores, edificios de madera, enemigos y personajes.
Tareas anteriores	-
Estimación	1 jornada
Satisface	RNF004

Tarea	3.2 Crear sprite protagonista
Descripción	Crear los sprites necesarios para crear un personaje principal femenino y con el pelo rosa, siendo coherente con la estética de los sprites descargados anteriormente.
Tareas anteriores	2.2 Familiarizarse con Aseprite 3.1 Descargar sprites
Estimación	1 jornada
Satisface	RNF005

Tarea	3.3 Organizar sprites
Descripción	Crear carpetas para los distintos tipos de sprites que se usarán y organizar cada spritesheet en su carpeta correspondiente. Estas carpetas estarán dentro de la carpeta general de sprites y serán: <ul style="list-style-type: none"> - Entorno - Personajes - Objetos - Interfaz
Tareas anteriores	2.4 Organizar el entorno de Unity 3.2 Crear Sprite protagonista
Estimación	½ jornada
Satisface	-

Tarea	3.4 Preparar sprites
Descripción	<p>Seleccionar los sprites y editar sus propiedades desde el editor.</p> <p>Debemos editar los siguientes campos:</p> <ul style="list-style-type: none"> - Texture Type (sprite 2D and UI) - Sprite Mode (multiple) - Pixels per Unit (16) - Filter Mode (no filter) - Compression (none) <p>Acceder a Sprite Editor y recortar en cuadrados de 16 por 16</p>
Tareas anteriores	3.3 Organizar sprites
Estimación	1 jornada
Satisface	-

Tarea	3.5 Descargar música
Descripción	Buscar y descargar música relajante que se pueda usar en bucle.
Tareas anteriores	-
Estimación	½ jornada
Satisface	-

Interfaz de usuario

Tarea	4.1 Preparar canvas
Descripción	Crear un objeto canvas en la jerarquía de la escena dentro de Unity para añadir dentro de este todas las interfaces.
Tareas anteriores	3.4 Preparar sprites
Estimación	½ jornada
Satisface	-

Tarea	4.2 Menú inicio
Descripción	<p>Crear un panel para el menú de inicio dentro del canvas donde aparezca:</p> <ul style="list-style-type: none"> - Nombre del juego - Botón para jugar - Botón para acceder al menú de configuración
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF055

Tarea	4.3 Menú pausa
Descripción	Crear un panel para el menú de pausa dentro del canvas donde aparezca: <ul style="list-style-type: none"> - Nombre del juego - Botón para volver al menú de inicio - Botón para acceder al menú de configuración - Botón para volver al juego
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF056

Tarea	4.4 Botón de pausa
Descripción	Crear un panel para añadir el botón de pausa mientras el juego está activo.
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF056

Tarea	4.5 Menú configuración
Descripción	Crear un panel para el menú de configuración dentro del canvas donde aparezca: <ul style="list-style-type: none"> - Nombre del juego - Barra para controlar el volumen - Botón para volver al menú de inicio o de pausa
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF057

Tarea	4.6 Menú fin
Descripción	Crear un panel para el menú de fin de partida dentro del canvas donde aparezca: <ul style="list-style-type: none"> - Texto de fin - Botón para volver al menú de inicio
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF072

Tarea	4.7 HUD protagonista
Descripción	<p>Crear un panel dentro del canvas para mostrar la información actualizada del personaje a lo largo de la partida. Este panel estará situado en la esquina superior izquierda de la pantalla y mostrará la siguiente información:</p> <ul style="list-style-type: none"> - Imagen del personaje - Nivel - Barra de vida - Barra de maná - Barra de experiencia para alcanzar el siguiente nivel - Dinero disponible
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF058

Tarea	4.8 Panel de accesos
Descripción	<p>Crear un panel dentro del canvas que se mostrará a lo largo del juego en la esquina inferior izquierda de la pantalla. Este panel tendrá 3 botones que permitirán al usuario acceder a:</p> <ul style="list-style-type: none"> - Inventario (representado por un zurrón) - Estadísticas (representado por un brazo con músculos) - Misiones activas (representado por un pergamo) <p>Al clicar cada uno de estos botones se abrirán o cerrarán los paneles correspondientes.</p>
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF059, RF061, RF064

Tarea	4.9 Panel inventario
Descripción	<p>Crear un panel para el inventario del canvas donde aparezca:</p> <ul style="list-style-type: none"> - 24 huecos cuadrados donde se muestren los objetos - Botón de usar - Botón de equipar - Botón de quitar - Panel con la imagen, el nombre y la descripción del objeto seleccionado
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF063

Tarea	4.10 Panel estadísticas
Descripción	<p>Crear un panel para mostrar las estadísticas del personaje dentro del canvas donde aparezca información sobre estas estadísticas como:</p> <ul style="list-style-type: none"> - Nivel - Experiencia - Experiencia necesaria para alcanzar el siguiente nivel - Daño - Defensa - Velocidad - Probabilidad de daño crítico - Probabilidad de bloqueo <p>También debe mostrar información sobre los atributos:</p> <ul style="list-style-type: none"> - Puntos de fuerza - Puntos de destreza - Puntos de inteligencia - Puntos disponibles - Botones para añadir los puntos disponibles a cada uno de los atributos
Tareas anteriores	4.1 Preparar canvas
Estimación	1 jornada
Satisface	RF061

Tarea	4.11 Panel misiones aceptadas
Descripción	<p>Crear un panel dentro del canvas donde aparezcan las misiones aceptadas por el personaje. Debe contener un panel para cada misión aceptada, y cada uno de ellos debe mostrar:</p> <ul style="list-style-type: none"> - Nombre de la misión - Progreso (enemigos matados / enemigos necesarios para completar la misión) - Experiencia de recompensa - Dinero de recompensa - Objeto de recompensa
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF064

Tarea	4.12 Panel arma equipada
Descripción	Crear un panel Dentro de canvas que nos muestre la imagen del arma que lleva equipada nuestro personaje. Si el personaje no lleva equipada ningún arma, no habrá ninguna imagen dentro de este panel. Debe mostrarse en la esquina superior derecha de la pantalla durante el juego.
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF060

Tarea	4.13 Panel diálogo
Descripción	<p>Crear un panel para los diálogos en el canvas donde aparezca:</p> <ul style="list-style-type: none"> - Imagen del NPC con el que hablamos - Nombre del NPC con el que hablamos - Texto - Botón de siguiente <p>Este panel se abrirá al interactuar con un NPC con diálogo asignado. En el texto aparecerá el primer texto de la lista, e irá avanzando al darle al botón siguiente. Cuando no queden textos por mostrar, este botón cerrará el panel.</p>
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF062

Tarea	4.14 Botón diálogo
Descripción	<p>Crear un panel en el canvas para interactuar con los NPCs con menú asignado. Será un panel cuadrado habrá dos versiones distintas</p> <ul style="list-style-type: none"> - “!” en el interior del cuadrado - “?” en el interior del cuadrado
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF062

Tarea	4.15 Panel misiones
Descripción	<p>Crear un panel dentro del canvas para mostrar las misiones. Debe mostrar un panel para cada una de las misiones y cada uno de estos debe mostrar:</p> <ul style="list-style-type: none"> - Nombre de la misión - Requisitos para cumplir la misión - Experiencia de recompensa - Oro de recompensa - Objeto de recompensa - Botón para aceptar la misión <p>Además, debe haber un botón en la parte superior derecha del panel para cerrarlo.</p>
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF063

Tarea	4.16 Panel misión completa
Descripción	<p>Crear un panel en el canvas para mostrar las misiones que se van completando. Debe mostrar:</p> <ul style="list-style-type: none"> - Texto de misión completa - Nombre de la misión - Experiencia ganada - Oro ganado - Objeto ganado - Botón para reclamar la recompensa y cerrar el panel
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF065

Tarea	4.17 Panel tienda
Descripción	<p>Crear un panel para la tienda dentro del canvas que muestre un panel para cada objeto disponible en la tienda. Cada panel debe mostrar:</p> <ul style="list-style-type: none"> - Imagen del objeto - Nombre del objeto - Precio del objeto - Botón para añadir cantidad - Botón para restar cantidad - Cantidad seleccionada del objeto - Botón para comprar <p>Además, debe haber un botón para cerrar la tienda en la parte superior derecha del panel.</p>
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF066

Tarea	4.18 Panel crafteo
Descripción	<p>Crear un panel de crafteo en el canvas. Debe estar dividido en dos partes:</p> <ul style="list-style-type: none"> - Panel izquierdo con paneles para cada una de las recetas disponibles, que muestren: <ul style="list-style-type: none"> o Imagen del objeto resultante o Nombre del objeto resultante - Panel derecho, con información de la receta seleccionada, que muestren: <ul style="list-style-type: none"> o Materiales necesarios: imagen de los objetos necesarios, sus nombres y la cantidad que tenemos y necesitamos de este objeto o Texto indicando que no tenemos los materiales necesarios o botón para crear el objeto si tenemos los materiales necesarios o Objeto resultante: imagen, nombre y descripción <p>Además, debe haber un botón en la parte superior derecha del panel para cerrarlo.</p>
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF067

Tarea	4.19 Panel loot
Descripción	<p>Crear un panel en el canvas donde aparezcan un panel por cada uno de los objetos que podemos saquear de los personajes derrotados. Cada uno debe mostrar:</p> <ul style="list-style-type: none"> - Imagen del objeto - Nombre del objeto - Cantidad <p>Estos paneles deben funcionar a forma de botón para recoger estos objetos al hacer clic en ellos.</p> <p>Además, debe haber un botón en la parte superior derecha del panel para cerrarlo.</p>
Tareas anteriores	4.1 Preparar canvas
Estimación	½ jornada
Satisface	RF068

Inputs

Tarea	5.1 Input para controles
Descripción	<p>Crear el sistema de input que nos permita usar el teclado y el ratón del ordenador para controlar el videojuego. Los controles deben ser:</p> <ul style="list-style-type: none"> - W: mover el personaje hacia arriba - A: mover el personaje hacia la izquierda - S: mover el personaje hacia abajo - D: mover el personaje hacia la derecha - Espacio: atacar con armas mágicas - Clic izquierdo: seleccionar e interactuar
Tareas anteriores	2.4 Organizar el entorno de Unity
Estimación	½ jornada
Satisface	RF001, RF005, RF006, RF007

Protagonista

Tarea	6.1 Prefab protagonista
Descripción	Crear un prefab para el protagonista con su sprite asignado y guardarlo en la carpeta de prefabs dentro de assets. Todas las tareas que se hagan para el protagonista se deben reflejar en este prefab para que siempre esté actualizado.
Tareas anteriores	3.4 Preparar sprites
Estimación	½ jornada
Satisface	-

Tarea	6.2 Mover protagonista
Descripción	Crear scripts necesarios para leer los inputs y mover con ellos al protagonista
Tareas anteriores	3.4 Preparar sprites 5.1 Input para controles 6.1 Prefab protagonista
Estimación	½ jornada
Satisface	RF001

Tarea	6.3 Animar protagonista
Descripción	Usar los sprites correspondientes para crea las animaciones de: - Idle - Correr Configurar el Animator con las animaciones creadas y sus transiciones. Crear scripts necesarios para comprobar qué animación hay que mostrar y girar los sprites según la dirección
Tareas anteriores	6.1 Prefab protagonista
Estimación	1 jornada
Satisface	RF002, RF003

Tarea	6.4 Detectar protagonista
Descripción	Añadir collider al prefab del protagonista para poder detectarlo y que pueda interactuar.
Tareas anteriores	6.1 Prefab protagonista
Estimación	½ jornada
Satisface	RF004

Sistema de vida

Tarea	7.1 Sistema de vida general
Descripción	Crear un script para gestionar la vida de los personajes en el juego, con el que se pueda asignar un valor de vida máxima, un valor inicial, permita recibir daño y compruebe el daño recibido y si el personaje sigue vivo.
Tareas anteriores	2.4 Organizar el entorno de Unity
Estimación	1 jornada
Satisface	RF012, RF014, RF045, RF046

Tarea	7.2 Sistema de vida protagonista
Descripción	Crear un script para gestionar la vida del protagonista. Además de heredar del script de vida general, debe permitirnos restaurar su salud y revivirlo.
Tareas anteriores	7.1 Sistema de vida general
Estimación	1 jornada
Satisface	RF012, RF013, RF014

Tarea	7.3 Revivir protagonista
Descripción	Crear los scripts necesarios para que cuando el protagonista muera sea llevado a un punto inicial y todos sus atributos vuelvan a tener el valor inicial.
Tareas anteriores	7.2 Sistema de vida protagonista
Estimación	1 jornada
Satisface	RF015

Tarea	7.4 Actualizar interfaz vida
Descripción	Crear los scripts necesarios para comunicar los datos actuales de vida del protagonista con la interfaz donde se muestra la barra de vida, de forma que siempre esté actualizada.
Tareas anteriores	4.7 HUD protagonista 7.3 Revivir protagonista
Estimación	1 jornada
Satisface	RF058

Tarea	7.5 Asignar vida al protagonista
Descripción	Añadir al prefab del protagonista su script de vida y asignarle valores.
Tareas anteriores	6.1 Prefab protagonista 7.2 Sistema de vida protagonista
Estimación	½ jornada
Satisface	RF012, RF013, RF014, RF015

Sistema de maná

Tarea	8.1 Sistema de maná general
Descripción	Crear un script para gestionar el maná de los personajes en el juego, con el que se pueda asignar un valor de maná máximo, un valor inicial, un valor de regeneración por segundo y permita usar maná y recuperarlo. Restaurar los valores cuando el protagonista muera.
Tareas anteriores	2.4 Organizar el entorno de Unity 7.3 Revivir protagonista
Estimación	1 jornada
Satisface	RF010, RF011

Tarea	8.2 Actualizar interfaz maná
Descripción	Crear los scripts necesarios para comunicar los datos actuales sobre maná del protagonista con la interfaz donde se muestra la barra de maná, de forma que siempre esté actualizada.
Tareas anteriores	4.7 HUD protagonista 8.1 Sistema de maná general
Estimación	1 jornada
Satisface	RF058

Tarea	8.3 Asignar maná al protagonista
Descripción	Añadir al prefab del protagonista el script de maná y asignarle valores.
Tareas anteriores	6.1 Prefab protagonista 8.1 Sistema de maná general
Estimación	½ jornada
Satisface	RF010, RF011

Sistema de experiencia

Tarea	9.1 Sistema de experiencia general
Descripción	Crear un script para gestionar la experiencia de los personajes en el juego, con el que se pueda asignar un valor de nivel máximo, un valor inicial, un valor de experiencia hasta el siguiente nivel, un valor de incremento de experiencia necesaria en los siguientes niveles y nos permita ganar experiencia.
Tareas anteriores	2.4 Organizar el entorno de Unity
Estimación	2 jornadas
Satisface	RF016, RF017

Tarea	9.2 Actualizar interfaz experiencia
Descripción	Crear los scripts necesarios para comunicar los datos actuales sobre la experiencia del protagonista con la interfaz donde se muestra la barra de experiencia y el nivel, de forma que siempre esté actualizada.
Tareas anteriores	4.7 HUD protagonista 9.1 Sistema de experiencia general
Estimación	1 jornada
Satisface	RF058

Tarea	9.3 Asignar experiencia al protagonista
Descripción	Añadir al prefab del protagonista el script de experiencia y asignarle valores.
Tareas anteriores	6.1 Prefab protagonista 9.1 Sistema de experiencia general
Estimación	½ jornada
Satisface	RF016, RF017

Sistema de estadísticas

Tarea	10.1 Sistema de estadísticas general
Descripción	Crear un script de ScriptableObject para gestionar las estadísticas de los personajes en el juego, con el que se pueda asignar valores de daño, defensa, velocidad, probabilidad de crítico y probabilidad de bloqueo. Debe permitir aumentar los valores de las estadísticas y también sobrescribir datos de nivel, experiencia actual y experiencia hasta el siguiente nivel. Además, debe permitir asignar unos valores por defecto y restaurarlos cuando el personaje muera.
Tareas anteriores	2.4 Organizar el entorno de Unity 7.3 Revivir protagonista 9.1 Sistema de experiencia general
Estimación	2 jornadas
Satisface	RF019, RF020

Tarea	10.2 Crear ScriptableObject de estadísticas
Descripción	Crear el objeto ScriptableObject de estadísticas del protagonista en su carpeta correspondiente de ScriptableObjects dentro de assets. Asignarle valores.
Tareas anteriores	10.1 Sistema de estadísticas general
Estimación	1 jornada
Satisface	RF020

Tarea	10.3 Actualizar interfaz estadísticas
Descripción	Crear los scripts necesarios para comunicar los datos actuales sobre las estadísticas del protagonista con la interfaz donde se muestran estos datos, de forma que siempre esté actualizada.
Tareas anteriores	4.10 Panel estadísticas 10.1 Sistema de estadísticas general
Estimación	1 jornada
Satisface	RF061

Tarea	10.4 Asignar estadísticas al protagonista
Descripción	Añadir al prefab del protagonista el ScriptableObject de estadísticas configurado.
Tareas anteriores	6.1 Prefab protagonista 10.2 Crear ScriptableObject de estadísticas
Estimación	½ jornada
Satisface	RF019, RF020

Sistema de atributos

Tarea	11.1 Sistema de atributos general
Descripción	<p>Crear un script para gestionar los atributos de los personajes en el juego, con el que se pueda asignar valores de fuerza, destreza, inteligencia y puntos disponibles para canjear. Debe permitir añadir puntos disponibles a estos atributos, y con esto aumentar las estadísticas, dependiendo del atributo aumentado:</p> <ul style="list-style-type: none"> - Fuerza: aumenta daño, defensa y porcentaje de bloqueo - Destreza: aumenta daño, porcentaje de bloqueo, porcentaje de crítico y velocidad - Inteligencia: aumenta defensa, porcentaje de bloqueo y porcentaje de crítico <p>El protagonista debe poder obtener puntos disponibles de atributo cuando sube de nivel. Los valores se restaurarán si el protagonista muere.</p>
Tareas anteriores	10.1 Sistema de estadísticas general
Estimación	2 jornadas
Satisface	RF018, RF019

Tarea	11.2 Actualizar interfaz atributos
Descripción	Crear los scripts necesarios para comunicar los datos actuales sobre los atributos del protagonista con la interfaz donde se muestran estos datos, de forma que siempre esté actualizada.
Tareas anteriores	4.10 Panel estadísticas 10.1 Sistema de estadísticas general
Estimación	1 jornada
Satisface	RF061

Tarea	11.3 Asignar atributos al protagonista
Descripción	Añadir al prefab del protagonista el script de atributos.
Tareas anteriores	6.1 Prefab protagonista 11.1 Sistema de atributos general
Estimación	½ jornada
Satisface	RF018

Objetos

Tarea	12.1 Crear objetos
Descripción	Crear script de tipo ScriptableObject para los objetos del juego. Debe tener información sobre qué tipo de objeto es (armas, paciones u otros), un identificador, un nombre, una imagen, una descripción, si el objeto es consumible, acumulable, y en este último caso la cantidad máxima que se puede acumular. Los objetos se deben poder copiar, usar, equipar y desequipar.
Tareas anteriores	3.4 Preparar sprites
Estimación	1 jornada
Satisface	RF029, RF030, RF031, RF032, RF033,

Tarea	12.2 Objeto poción vida
Descripción	Crear script para las pociones que restauran vida y crear el ScriptableObject en la carpeta correspondiente dentro de assets configurando sus atributos. Deben restaurar 20 puntos de salud y se pueden acumular hasta 60.
Tareas anteriores	12.1 Crear objetos
Estimación	½ jornada
Satisface	RF029

Tarea	12.3 Objeto poción maná
Descripción	Crear script para las pociones que restauran maná y crear el ScriptableObject en la carpeta correspondiente dentro de assets configurando sus atributos. Deben restaurar 20 puntos de maná y se pueden acumular hasta 60.
Tareas anteriores	12.1 Crear objetos
Estimación	½ jornada
Satisface	RF030

Tarea	12.4 Otros objetos
Descripción	Crear ScriptableObjects en la carpeta correspondiente dentro de assets para el resto de objetos que aparecerán en el juego. Se debe crear, al menos: <ul style="list-style-type: none"> - Pociones mejoradas de vida - Pociones mejoradas de maná - 6 distintos ingredientes para craftear
Tareas anteriores	12.1 Crear objetos
Estimación	1 jornada
Satisface	RF031, RF032, RF033

Sistema de inventario

Tarea	13.1 Crear inventario
Descripción	Crear los scripts necesarios para el inventario, que contendrá un número de slots que podemos definir y contener objetos en estos. Definir 24 huecos por el momento y asociar inventario al protagonista. En el inventario se debe poder añadir objetos y verlos.
Tareas anteriores	12.1 Crear objetos
Estimación	2 jornadas
Satisface	RF025

Tarea	13.2 Crear slots
Descripción	Crear los scripts necesarios para gestionar los huecos que estarán dentro del inventario y contendrán a los objetos del mismo. Deben tener acceso a la información del objeto que contiene. Al abrir el inventario, cada slot mostrará la imagen y cantidad de su objeto. Guardar como prefab en la carpeta correspondiente para asociarlos al inventario y llamar a los slots necesarios dependiendo del número de huecos que definamos.
Tareas anteriores	12.1 Crear objetos
Estimación	2 jornadas
Satisface	RF025

Tarea	13.3 Interactuar con los slots
Descripción	Crear los scripts necesarios para poder interactuar con cada hueco del inventario. Los huecos pueden estar seleccionados o deseleccionados. Cuando un slot se selecciona, se abre un panel con información sobre el objeto que contiene y nos dará acceso a interactuar con él.
Tareas anteriores	4.9 Panel inventario 13.2 Crear slots
Estimación	1 jornada
Satisface	RF025

Tarea	13.4 Información objetos en inventario
Descripción	Crear los scripts necesarios para que al seleccionar un slot se abra un panel y muestre el resto de información del objeto que contiene: nombre y descripción, además de la imagen del mismo.
Tareas anteriores	13.3 Interactuar con los slots
Estimación	1 jornada
Satisface	RF025

Tarea	13.5 Interactuar con objetos en inventario
Descripción	Crear los scripts necesarios para que, al seleccionar un slot, podamos usar el objeto (si son pociones), equiparlos y desequiparlos (si son armas).
Tareas anteriores	13.3 Interactuar con los slots
Estimación	1 jornada
Satisface	RF026, RF027, RF028

Tarea	13.6 Actualizar interfaz inventario
Descripción	Crear los scripts necesarios para comunicar los datos actuales sobre el inventario, sus slots y contenido con la interfaz de inventario, de forma que siempre esté actualizada.
Tareas anteriores	4.9 Panel inventario 13.1 Crear inventario 13.2 Crear slots
Estimación	1 jornada
Satisface	RF059

NPCs

Tarea	14.1 Animaciones NPCs
Descripción	Crear las animaciones de andar de cada NPC con sus sprites correspondientes.
Tareas anteriores	3.4 Preparar sprites
Estimación	1 jornada
Satisface	RF048

Tarea	14.2 Movimiento NPCs
Descripción	Crear un script para los NPCs que permita asignarles un recorrido por el que se moverán en bucle.
Tareas anteriores	2.4 Organizar el entorno de Unity
Estimación	1 jornada
Satisface	RF049

Tarea	14.3 Crear NPCs
Descripción	Añadir npcs a la escena y asignarles sus recorridos.
Tareas anteriores	14.1 Animaciones NPCs 14.2 Movimiento NPCs 24.4 Crear aldeas 24.6 Crear edificio crafteo
Estimación	1 jornada
Satisface	RF052, RF053, RF054

Tarea	14.4 Detectar NPCs
Descripción	Añadir colliders a los NPCs para poder interactuar con ellos.
Tareas anteriores	14.3 Crear NPCs
Estimación	½ jornada
Satisface	RF005

Sistema de diálogo

Tarea	15.1 Sistema de diálogo general
Descripción	Crear un script de tipo ScriptableObject para gestionar los diálogos. Este script debe contener tener información sobre el NPC que participa (icono y nombre). Debe tener también un saludo, una despedida y una lista de textos que conforman la conversación. Además, a un diálogo se le puede asignar una interacción extra de tipo tienda, misiones o crafteo.
Tareas anteriores	2.4 Organizar el entorno de Unity
Estimación	2 jornadas
Satisface	RF050, RF051

Tarea	15.2 Interacción NPC
Descripción	Hacer los scripts necesarios para asignar un diálogo a un NPC y gestionar mediante colliders que cuando el protagonista se acerque a un NPC con diálogo asignado salga un botón para interactuar con él y que se abra el diálogo.
Tareas anteriores	4.14 Botón diálogo 6.4 Detectar protagonista 14.4 Detectar NPCs 15.1 Sistema de diálogo general
Estimación	1 jornada
Satisface	RF050, RF062

Tarea	15.3 Actualizar interfaz diálogos
Descripción	Crear los scripts necesarios para comunicar los datos de los diálogos con la interfaz donde se muestran. El botón de siguiente en la interfaz nos ayudará a avanzar por los textos de la conversación y terminarla.
Tareas anteriores	4.13 Panel diálogo 15.1 Sistema de diálogo general
Estimación	1 jornada
Satisface	RF062

Tarea	15.4 Crear y asignar diálogos
Descripción	Crear los ScriptableObjects en su carpeta correspondiente dentro de assets para cada diálogo (sin interacción extra) que se mostrará en el juego. Asignar estos diálogos a sus correspondientes NPCs.
Tareas anteriores	14.2 Crear NPCs 15.1 Sistema de diálogo general
Estimación	1 jornada
Satisface	RF052, RF053, RF054

Sistema de dinero

Tarea	16.1 Sistema de dinero general
Descripción	Crear los scripts necesarios para gestionar el dinero del protagonista. El dinero serán monedas de oro. Debemos poder ganar oro y usarlo.
Tareas anteriores	2.4 Organizar el entorno de Unity
Estimación	1 jornadas
Satisface	RF008, RF009

Tarea	16.2 Actualizar interfaz dinero
Descripción	Crear los scripts necesarios para comunicar los datos actuales de dinero con la interfaz donde se muestra la cantidad, de forma que siempre esté actualizada.
Tareas anteriores	4.7 HUD protagonista 16.1 Sistema de dinero general
Estimación	½ jornada
Satisface	RF058

Misiones

Tarea	17.1 Sistema de misiones general
Descripción	Crear un script de tipo ScriptableObject para gestionar las misiones. Este script debe contener tener información sobre las misiones: nombre, id, descripción, cantidad de enemigos que necesitamos matar para completar la misión y recompensas con sus cantidades (experiencia, oro y objeto). Se debe poder añadir progreso a las misiones y comprobar si se han completado.
Tareas anteriores	9.1 Sistema de experiencia general 12.1 Crear objetos 16.1 Sistema de dinero general
Estimación	2 jornadas
Satisface	RF022

Tarea	17.2 Actualizar interfaz misiones
Descripción	Crear los scripts necesarios para comunicar los datos de las misiones con la interfaz donde se muestran las misiones disponibles. El panel de tienda se abrirá tras interactuar con un npc con diálogo con interacción extra del tipo misiones, y se cerrará al clicar en el botón de la esquina superior derecha.
Tareas anteriores	4.15 Panel misiones 17.1 Sistema de misiones general
Estimación	1 jornada
Satisface	RF063

Tarea	17.3 Aceptar misiones
Descripción	Crear los scripts necesarios para poder aceptar misiones. Cuando una misión se acepte, se activará, desaparecerá del panel de misiones y aparecerá en el panel de misiones aceptadas
Tareas anteriores	17.1 Sistema de misiones general
Estimación	1 jornada
Satisface	RF021

Tarea	17.4 Actualizar interfaz misiones aceptadas
Descripción	Crear los scripts necesarios para comunicar los datos de las misiones con la interfaz donde se muestran las misiones aceptadas, con la información actualizada de cada una de ellas y su progreso. Este panel se abrirá al clicar sobre su botón correspondiente en el menú inferior y se cerrará de la misma forma.
Tareas anteriores	4.11 Panel misiones aceptadas 17.3 Aceptar misiones
Estimación	1 jornada
Satisface	RF064

Tarea	17.5 Actualizar interfaz misión completa
Descripción	Crear los scripts necesarios para comunicar los datos de las misiones con la interfaz donde se muestra que se han completado. Cuando una misión se haya completado, desaparecerá del panel de misiones aceptadas y aparecerá un panel emergente para avisarnos de que la hemos terminado y poder recoger la recompensa.
Tareas anteriores	4.16 Panel misión completa 17.3 Aceptar misiones
Estimación	½ jornada
Satisface	RF065

Tarea	17.6 Conseguir recompensas
Descripción	Hacer los scripts necesarios para que cuando completemos una misión recibamos la experiencia, el oro y los objetos asociados a su recompensa. Tanto la experiencia como el oro deben actualizarse inmediatamente, mientras que los objetos deben aparecer en nuestro inventario al recoger la recompensa del panel de misión completa.
Tareas anteriores	17.1 Sistema de misiones general 17.3 Aceptar misiones
Estimación	1 jornada
Satisface	RF022

Tarea	17.7 Crear misiones
Descripción	Crear los ScriptableObjects de las misiones del juego en su carpeta correspondiente dentro de assets. Configurar los datos de todas las misiones creadas.
Tareas anteriores	17.1 Sistema de misiones general
Estimación	1 jornada
Satisface	RF021, RF022

Tarea	17.8 Crear y asignar diálogo misiones
Descripción	Crear los ScriptableObjects en su carpeta correspondiente dentro de assets para los diálogos con interacción extra de tipo misiones. Asignar estos diálogos a sus correspondientes npcs.
Tareas anteriores	14.2 Crear npcs 15.1 Sistema de diálogo general 17.1 Sistema de misiones general
Estimación	½ jornada
Satisface	RF053

Tienda

Tarea	18.1 Sistema de tienda general
Descripción	Crear los scripts necesarios para gestionar el sistema de la tienda. Debe contener una lista con los objetos disponibles en la tienda y poder acceder a su información.
Tareas anteriores	12.1 Crear objetos
Estimación	2 jornadas
Satisface	RF023

Tarea	18.2 Añadir objetos a tienda
Descripción	Crear los scripts necesarios para añadir objetos a la tienda, pasándole un objeto como referencia. Debe contener información de la imagen, el nombre, el precio y la cantidad máxima que se puede comprar.
Tareas anteriores	16.1 Sistema de dinero general 18.1 Sistema de tienda general
Estimación	1 jornada
Satisface	RF023

Tarea	18.3 Comprar objetos
Descripción	Crear los scripts necesarios para que el protagonista pueda comprar objetos de la tienda. Debe poder comprar objetos por valor igual o menor al dinero del que dispone. Se pueden comprar varios objetos a la vez usando los botones para sumar y restar cantidad de objetos. Estos objetos se añadirán directamente a su inventario y se restará el dinero.
Tareas anteriores	18.2 Añadir objetos a tienda
Estimación	1 jornada
Satisface	RF023

Tarea	18.4 Actualizar interfaz tienda
Descripción	Crear los scripts necesarios para comunicar los datos de la tienda y sus objetos con su interfaz correspondiente. Para cada objeto se debe mostrar un panel con información del objeto (imagen, nombre y precio) y un botón para comprarlos. El panel de tienda se abrirá tras interactuar con un npc con diálogo con interacción extra del tipo tienda, y se cerrará al clicar en el botón de la esquina superior derecha.
Tareas anteriores	4.17 Panel tienda 18.3 Comprar objetos
Estimación	1 jornada
Satisface	RF066

Tarea	18.5 Crear y asignar diálogo tienda
Descripción	Crear los ScriptableObjects en su carpeta correspondiente dentro de assets para los diálogos con interacción extra de tipo tienda. Asignar estos diálogos a sus correspondientes npcs.
Tareas anteriores	14.2 Crear npcs 15.1 Sistema de diálogo general 18.1 Sistema de tienda general
Estimación	½ jornada
Satisface	RF052

Crafeo

Tarea	19.1 Sistema recetas
Descripción	Crear los scripts de tipo ScriptableObject necesarios crear recetas, en su carpeta correspondiente dentro de assets. Las recetas deben tener un nombre y también deben tener asociados dos objetos necesarios para crearlas y un objeto resultante.
Tareas anteriores	12.1 Crear objetos
Estimación	1 jornada
Satisface	RF024

Tarea	19.2 Sistema de crafteo general
Descripción	Crear los scripts necesarios para gestionar el sistema de crafteo. Debe contener una lista de recetas y poder acceder a la información de ellas (información de los materiales necesarios y resultantes) y debe permitirnos cargar recetas creadas y mostrarlas.
Tareas anteriores	19.1 Sistema recetas
Estimación	2 jornadas
Satisface	RF024

Tarea	19.3 Craftear recetas
Descripción	Crear los scripts necesarios para que el protagonista pueda crear objetos desde el panel de crafteo. Debe poder acceder a los objetos necesarios y la cantidad de cada uno tanto como al inventario del protagonista para comprobar que tiene los materiales suficientes (de lo contrario no le dejará craftear y le avisará de que le faltan materiales). Al craftear un objeto, desaparecerán los materiales necesarios del inventario del protagonista y el objeto resultante aparecerá en su inventario.
Tareas anteriores	13.5 Interactuar con objetos en inventario 19.2 Sistema de crafteo general
Estimación	2 jornadas
Satisface	RF024

Tarea	19.4 Actualizar interfaz crafteo
Descripción	Crear los scripts necesarios para comunicar los datos de la tienda y sus objetos con su interfaz correspondiente. Para cada objeto se debe mostrar un panel con información del objeto (imagen, nombre y precio) y un botón para comprarlos. El panel de tienda se abrirá tras interactuar con un NPC con diálogo con interacción extra del tipo tienda, y se cerrará al clicar en el botón de la esquina superior derecha.
Tareas anteriores	4.18 Panel crafteo 19.2 Sistema de crafteo general
Estimación	1 jornada
Satisface	RF067

Tarea	19.5 Crear recetas
Descripción	Configurar los ScriptableObjects de recetas en su carpeta correspondiente, asignando los objetos resultados y materiales variados. Se deben poder craftear distintos objetos como pociones o armas.
Tareas anteriores	16.1 Sistema recetas
Estimación	1 jornada
Satisface	RF024

Tarea	19.6 Crear y asignar diálogo crafteo
Descripción	Crear los ScriptableObjects en su carpeta correspondiente dentro de assets para los diálogos con interacción extra de tipo tienda. Asignar estos diálogos a sus correspondientes NPCs.
Tareas anteriores	14.2 Crear NPCs 15.1 Sistema de diálogo general 19.2 Sistema de crafteo general
Estimación	½ jornada
Satisface	RF054

Enemigos

Tarea	20.1 Animaciones enemigos
Descripción	Crear las animaciones de movimiento y ataque de cada enemigo con sus sprites correspondientes. Se deben animar, al menos, los siguientes enemigos: <ul style="list-style-type: none"> - Slime azul - Slime verde - Slime naranja - Llama roja - Llama blanca
Tareas anteriores	3.4 Preparar sprites
Estimación	1 jornada
Satisface	RF042

Tarea	20.2 Movimiento enemigos
Descripción	Crear un script para los enemigos que permita asignarles un recorrido por el que se moverán en bucle.
Tareas anteriores	2.4 Organizar el entorno de Unity
Estimación	½ jornada
Satisface	RF043

Tarea	20.3 IA enemigos
Descripción	Crear los scripts necesarios para los enemigos que permita asignarles un rango a su alrededor y perseguir al protagonista cuando entre dentro de este rango (dejando de seguir el camino que previamente se les había asignado). Además, también debe permitir asignarles un rango a partir del cual harán daño al protagonista. Cuando el protagonista salga de estos rangos, el enemigo volverá a seguir su camino asignado. Usar colliders para detectar al protagonista.
Tareas anteriores	6.4 Detectar protagonista 20.2 Movimiento enemigos
Estimación	1 jornada
Satisface	RF044

Tarea	20.4 Ataque enemigos
Descripción	Crear los scripts necesarios para que el enemigo ataque. Se debe poder asignar un tipo de ataque: normal o cargado. Además, se le asignará el daño que hace al atacar y la velocidad de ataque. Los ataques deben restar puntos de vida al protagonista mientras este siga con vida.
Tareas anteriores	20.3 IA enemigos
Estimación	1 jornada
Satisface	RF012, RF044

Tarea	20.5 Crear y configurar enemigos
Descripción	Añadir enemigos con sus animaciones a la escena y asignarles sus recorridos. Añadir también los scripts creados para ellos de IA y ataque.
Tareas anteriores	20.1 Animaciones enemigos 20.4 Ataque enemigos 24.5 Crear bosques
Estimación	1 jornada
Satisface	RF075

Tarea	20.6 Detectar enemigos
Descripción	Añadir colliders a los enemigos para que puedan interactuar con el protagonista y el entorno.
Tareas anteriores	20.5 Crear y configurar enemigos
Estimación	½ jornada
Satisface	RF007

Tarea	20.7 Asignar vida enemigos
Descripción	Añadir a los enemigos el script general creado anteriormente para gestionar la vida, de forma que puedan ser dañados y matados por el protagonista.
Tareas anteriores	7.1 Sistema de vida general 20.5 Crear y configurar enemigos
Estimación	½ jornada
Satisface	RF045, RF046

Tarea	20.8 Configurar vida enemigos
Descripción	Configurar la cantidad de vida que tiene cada enemigo. Los enemigos que aparezcan al principio de la aventura tendrán la salud más baja (24), mientras que los enemigos que aparezcan al final de esta tendrán la salud mucho más alta (+100).
Tareas anteriores	20.7 Asignar vida enemigos
Estimación	½ jornada
Satisface	RF045, RF046

Armas

Tarea	21.1 Crear armas
Descripción	Crear script de tipo ScriptableObject para las armas del juego, que hereden del script de objeto. Además de la información básica de los objetos, debe tener información sobre qué tipo de arma es (física o mágica), el daño que aumenta al protagonista, y las probabilidades que aumenta de daño crítico y de bloqueo. Si el arma es de tipo mágica, debe asignarse el maná que usa y se debe poder asignar un sprite con el efecto que lanza.
Tareas anteriores	12.1 Crear objetos
Estimación	1 jornadas
Satisface	RF034, RF035, RF036, RF037, RF038, RF039, RF040, RF041

Tarea	21.2 Arma física espada
Descripción	<p>Crear el ScriptableObject de arma en su carpeta correspondiente.</p> <p>Configurar esta arma con los siguientes datos:</p> <ul style="list-style-type: none"> - Su imagen - Tipo: física - Daño: 4 - Probabilidad de crítico: 1 - Probabilidad de bloqueo: 0.2
Tareas anteriores	21.1 Crear armas
Estimación	¼ jornada
Satisface	RF034

Tarea	21.3 Arma física arco
Descripción	<p>Crear el ScriptableObject de arma en su carpeta correspondiente.</p> <p>Configurar esta arma con los siguientes datos:</p> <ul style="list-style-type: none"> - Su imagen - Tipo: física - Daño: 8 - Probabilidad de crítico: 2 - Probabilidad de bloqueo: 1
Tareas anteriores	21.1 Crear armas
Estimación	¼ jornada
Satisface	RF035

Tarea	21.4 Arma física mazo
Descripción	<p>Crear el ScriptableObject de arma en su carpeta correspondiente.</p> <p>Configurar esta arma con los siguientes datos:</p> <ul style="list-style-type: none"> - Su imagen - Tipo: física - Daño: 20 - Probabilidad de crítico: 4 - Probabilidad de bloqueo: 2
Tareas anteriores	21.1 Crear armas
Estimación	¼ jornada
Satisface	RF036

Tarea	21.5 Arma física hueso
Descripción	<p>Crear el ScriptableObject de arma en su carpeta correspondiente.</p> <p>Configurar esta arma con los siguientes datos:</p> <ul style="list-style-type: none"> - Su imagen - Tipo: física - Daño: 50 - Probabilidad de crítico: 20 - Probabilidad de bloqueo: 10
Tareas anteriores	21.1 Crear armas
Estimación	¼ jornada
Satisface	RF037

Tarea	21.6 Arma mágica libro
Descripción	<p>Crear el ScriptableObject de arma en su carpeta correspondiente.</p> <p>Configurar esta arma con los siguientes datos:</p> <ul style="list-style-type: none"> - Su imagen - Imagen de su efecto: bola de fuego - Tipo: mágica - Daño: 5 - Maná: 4 - Probabilidad de crítico: 1.5 - Probabilidad de bloqueo: 0.5
Tareas anteriores	21.1 Crear armas
Estimación	¼ jornada
Satisface	RF038

Tarea	21.7 Arma mágica bastón
Descripción	<p>Crear el ScriptableObject de arma en su carpeta correspondiente.</p> <p>Configurar esta arma con los siguientes datos:</p> <ul style="list-style-type: none"> - Su imagen - Imagen de su efecto: pico de hierba - Tipo: mágica - Daño: 12 - Maná: 10 - Probabilidad de crítico: 3 - Probabilidad de bloqueo: 2
Tareas anteriores	21.1 Crear armas
Estimación	¼ jornada
Satisface	RF039

Tarea	21.8 Arma mágica vara
Descripción	<p>Crear el ScriptableObject de arma en su carpeta correspondiente.</p> <p>Configurar esta arma con los siguientes datos:</p> <ul style="list-style-type: none"> - Su imagen - Imagen de su efecto: pico de hielo - Tipo: mágica - Daño: 26 - Maná: 15 - Probabilidad de crítico: 10 - Probabilidad de bloqueo: 8
Tareas anteriores	21.1 Crear armas
Estimación	1/4 jornada
Satisface	RF040

Tarea	21.9 Arma mágica tridente
Descripción	<p>Crear el ScriptableObject de arma en su carpeta correspondiente.</p> <p>Configurar esta arma con los siguientes datos:</p> <ul style="list-style-type: none"> - Su imagen - Imagen de su efecto: bola de energía - Tipo: mágica - Daño: 60 - Maná: 40 - Probabilidad de crítico: 30 - Probabilidad de bloqueo: 20
Tareas anteriores	21.1 Crear armas
Estimación	1/4 jornada
Satisface	RF041

Tarea	21.10 Usar armas
Descripción	<p>Crear los scripts necesarios para que el protagonista pueda equiparse y desequiparse armas. Cuando tenga un arma equipada, las estadísticas del arma se sumarán a las del protagonista.</p> <p>Las armas se deben poder equipar y desequipar.</p>
Tareas anteriores	10.4 Asignar estadísticas al protagonista 21.1 Crear armas
Estimación	1 jornada
Satisface	RF026

Tarea	21.11 Actualizar interfaz armas
Descripción	Crear los scripts necesarios para comunicar los datos de armas equipadas con la interfaz correspondiente. Cuando haya un arma equipada, se verá su imagen en el panel de la zona superior derecha de la pantalla.
Tareas anteriores	4.12 Panel arma equipada 21.10 Usar armas
Estimación	½ jornada
Satisface	RF060

Combate

Tarea	22.1 Seleccionar enemigos
Descripción	Crear los scripts necesarios para poder seleccionar a un enemigo con el clic izquierdo del ratón o entrando en su rango. Se podrán seleccionar cuando tengamos un arma equipada. Si tenemos equipada un arma mágica, podremos seleccionarlo desde cualquier distancia y se mostrará un círculo blanco alrededor del enemigo. Sin embargo, si tenemos equipada un arma física, se seleccionará de rojo directamente al entrar en su rango de ataque.
Tareas anteriores	20.6 Detectar enemigos 21.10 Usar armas
Estimación	2 jornadas
Satisface	RF006

Tarea	22.2 Atacar enemigos
Descripción	Crear los scripts necesarios para poder dañar a un enemigo, usando la barra espaciadora, cuando un enemigo sea seleccionado y siga vivo. El daño hecho será calculado con las estadísticas de las armas y del protagonista.
Tareas anteriores	20.7 Asignar vida enemigos 21.10 Usar armas 22.1 Seleccionar enemigos
Estimación	1 jornada
Satisface	RF007, RF045

Tarea	22.3 Animación magia
Descripción	Crear los scripts necesarios para que al usar un arma mágica se muestre una animación.
Tareas anteriores	21.1 Crear armas 22.1 Seleccionar enemigos
Estimación	½ jornada
Satisface	-

Tarea	22.4 Ser atacados por enemigos
Descripción	Crear los scripts necesarios para poder ser dañado por un enemigo, mientras este y el protagonista sigan vivos y el protagonista esté dentro de su rango de ataque. El daño hecho será el asignado previamente al enemigo atacante.
Tareas anteriores	7.5 Asignar vida al protagonista 20.4 Ataque enemigos
Estimación	1 jornada
Satisface	RF012

Tarea	22.5 Ver daño
Descripción	Crear los scripts necesarios para que cuando un enemigo o el protagonista reciba daño se vea el número de vida que se ha quitado.
Tareas anteriores	22.2 Atacar enemigos 22.4 Ser atacados por enemigos
Estimación	1 jornada
Satisface	-

Loot

Tarea	23.1 Matar enemigos
Descripción	Crear los scripts necesarios para que cuando un enemigo muera se desactive y salga una sombra en su lugar.
Tareas anteriores	20.7 Asignar vida enemigos
Estimación	1 jornada
Satisface	RF046

Tarea	23.2 Loot enemigos
Descripción	Crear los scripts necesarios para poder asignar a un enemigo los puntos de experiencia y los objetos que podremos obtener al matarlos. Además, los objetos deben tener asignados la probabilidad de aparecer cuando el enemigo es derrotado.
Tareas anteriores	9.1 Sistema de experiencia general 21.1 Crear objetos
Estimación	2 jornadas
Satisface	RF047

Tarea	23.3 Configurar loot enemigos
Descripción	Asignar a los enemigos el script creado para asignarles un loot y configurar la cantidad de experiencia y los objetos que tendrán cada uno.
Tareas anteriores	20.5 Crear y configurar enemigos 23.2 Loot enemigos
Estimación	1 jornada
Satisface	RF047

Tarea	23.4 Sistema de loot general
Descripción	Crear los scripts necesarios para que cuando un enemigo muera se generen los objetos asociados a su loot y podamos verlos y recogerlos.
Tareas anteriores	23.1 Matar enemigos 23.2 Loot enemigos
Estimación	1 jornada
Satisface	RF047

Tarea	23.5 Actualizar interfaz loot
Descripción	Crear los scripts necesarios para comunicar los datos de loot actualizados de los enemigos al morir con su interfaz correspondiente, de forma que podamos interactuar con la sombra del enemigo para obtener los objetos de su loot al clicarlos. El panel se abre al clicar en la sombra de un enemigo y se cierra en el botón superior derecho.
Tareas anteriores	4.19 Panel loot 23.4 Sistema de loot general
Estimación	1 jornada
Satisface	RF068

Tarea	23.6 Obtener loot
Descripción	Crear los scripts necesarios para que, al matar a un enemigo, la experiencia de su loot se sume automáticamente a la de nuestro protagonista. Además, al seleccionar los objetos de su loot estos desaparecerán del panel y aparecerán en el inventario del protagonista.
Tareas anteriores	13.5 Interactuar con objetos en inventario 23.4 Sistema de loot general
Estimación	1 jornadas
Satisface	RF047

Mapa

Tarea	24.1 Crear capas
Descripción	Crear un grid con distintas capas donde se pintará el mapa y se colocarán los elementos. Estas capas, en orden de más lejano a más cercano, serán: <ul style="list-style-type: none"> - Foreground - Suelo - Edificios - Límites - Props - Protagonista - Enemigos - UI
Tareas anteriores	2.4 Organizar el entorno de Unity
Estimación	½ jornada
Satisface	-

Tarea	24.2 Crear tilemaps
Descripción	Usar la herramienta de tilemaps de Unity para configurar los sprites que usaremos para crear el mapa y poder usarlos de forma más cómoda.
Tareas anteriores	3.4 Preparar sprites
Estimación	½ jornada
Satisface	-

Tarea	24.3 Diseñar mapa
Descripción	Hacer el diseño del mapa del juego. Servirá de guía para crear cada una de las zonas. Debe contar con, al menos: <ul style="list-style-type: none"> - 3 aldeas - 5 bosques
Tareas anteriores	-
Estimación	1 jornada
Satisface	-

Tarea	24.4 Crear aldeas
Descripción	Crear las aldeas diseñadas previamente con los sprites preparados. Las aldeas deben tener, al menos: <ul style="list-style-type: none"> - Un NPC de misiones - Un NPC de tienda - Un edificio donde se encuentre un NPC de crafteo
Tareas anteriores	24.1 Crear capas 24.2 Crear tilemaps 24.3 Diseñar mapa
Estimación	1 jornada
Satisface	RF074

Tarea	24.5 Crear bosques
Descripción	Crear los bosques diseñados previamente con los sprites preparados. Los bosques deben estar llenos de enemigos.
Tareas anteriores	24.1 Crear capas 24.2 Crear tilemaps 24.3 Diseñar mapa
Estimación	1 jornada
Satisface	RF075

Tarea	24.6 Crear edificio crafteo
Descripción	Crear el interior de una casa con los sprites preparados. Debe tener aspecto rústico y contener un npc que nos permita craftear.
Tareas anteriores	24.1 Crear capas 24.2 Crear tilemaps
Estimación	½ jornada
Satisface	RF076

Tarea	24.7 Cámara
Descripción	Crear la cámara que siga al protagonista a lo largo del mapa. Configurar los límites a partir de los que la cámara se moverá para perseguir al protagonista o permanecerá estática mientras el protagonista se mueve libremente.
Tareas anteriores	2.4 Organizar el entorno de Unity
Estimación	½ jornada
Satisface	RF078

Tarea	24.8 Limitaciones
Descripción	Usar colliders para establecer los límites en los mapas. Debemos establecer límites para que el protagonista no pueda salirse de él, ni pasar por encima de edificios, npcs o agua. También se limitará la zona por la que se puede mover la cámara.
Tareas anteriores	24.4 Crear aldeas 24.5 Crear bosques 24.6 Crear edificio crafteo 24.7 Cámara
Estimación	½ jornada
Satisface	RF074, RF075, RF076, RF078

Tarea	24.9 Conectar mapa
Descripción	Crear los scripts necesarios y ayudarnos de los colliders para permitir al protagonista moverse por las distintas zonas del mapa y entrar y salir de los edificios. La cámara se moverá a la nueva zona a la que entre el protagonista.
Tareas anteriores	24.8 Limitaciones
Estimación	½ jornada
Satisface	RF001

Tarea	24.10 Zona inicio
Descripción	Establecer la zona del mapa donde aparecerá el personaje al empezar la partida o tras morir.
Tareas anteriores	6.1 Prefab protagonista 24.4 Crear aldeas 24.6 Cámara
Estimación	½ jornada
Satisface	RF015

Flujo

Tarea	25.1 Configurar menú inicio
Descripción	Configurar los botones del menú de inicio, de forma que desde cada uno de ellos se pueda acceder a: <ul style="list-style-type: none"> - Botón jugar: comenzar el juego - Botón configuración: menú de configuración
Tareas anteriores	4.2 Menú inicio
Estimación	1 jornada
Satisface	RF069, RF071

Tarea	25.2 Configurar menú pausa
Descripción	Configurar los botones del menú de pausa, de forma que desde cada uno de ellos se pueda acceder a: <ul style="list-style-type: none"> - Botón inicio inicio: menú de inicio - Botón configuración: menú de configuración - Botón volver: acceder al juego en el mismo punto donde se pausó
Tareas anteriores	4.3 Menú pausa
Estimación	½ jornada
Satisface	RF071, RF073

Tarea	25.3 Configurar botón pausa
Descripción	Configurar el botón de pausa que se mostrará mientras la partida esté en curso de forma que nos permita acceder al menú de pausa.
Tareas anteriores	4.4 Botón pausa
Estimación	½ jornada
Satisface	RF071, RF073

Tarea	25.4 Configurar menú configuración
Descripción	Configurar los botones del menú de configuración, de forma que desde cada uno de ellos se pueda acceder a: <ul style="list-style-type: none"> - Barra de volumen: subir y bajar el volumen de la música del juego - Botón inicio: menú de inicio
Tareas anteriores	4.5 Menú configuración
Estimación	1 jornada
Satisface	RF070

Tarea	25.5 Configurar menú fin
Descripción	Configurar los botones del menú de fin, de forma que desde cada uno de ellos se pueda acceder a: - Botón inicio: menú de inicio
Tareas anteriores	4.6 Menú fin
Estimación	1 jornada
Satisface	RF071, RF072

Tarea	25.6 Configurar fin
Descripción	Configurar un fin de partida desde el cual se acceda al menú de fin.
Tareas anteriores	4.6 Menú fin
Estimación	1 jornada
Satisface	RF072

Música

Tarea	26.1 Añadir música
Descripción	Añadir la música descargada a las escenas del juego para que suene de fondo.
Tareas anteriores	3.5 Descargar música 24.4 Crear aldeas 24.5 Crear bosques 24.6 Crear edificio crafteo
Estimación	1 jornada
Satisface	RF077

Cronograma

He creado el cronograma a seguir para realizar las tareas en orden teniendo en cuenta la división del trabajo en tres Sprint. Cada uno de los Sprint está dividido en dos meses, y cada mes tiene 20 días de trabajo. Por lo tanto, mi cronograma está dividido en 6 bloques de trabajo divididos en 20 días, todo esto organizado en orden cronológico.

A continuación, muestro las tablas correspondientes a cada bloque de trabajo indicando las fechas de cada día y las tareas en las que trabajé en cada una de ellas. Algunas tareas aparecen repetidas, pues ocuparán más de una jornada de trabajo. La primera fecha en la que aparece una tarea es el día de comienzo de esta, y la última fecha en la que aparece es la fecha final para tenerla terminada.

Primer Sprint (1 de Febrero de 2022 a 28 de Marzo de 2022)

Febrero

Día	Tarea
01/02/2022	1.1 Instalar Unity 1.2 Instalar Aseprite
02/02/2022	2.1 Familiarizarse con Unity
03/02/2022	2.1 Familiarizarse con Unity
04/02/2022	2.2 Familiarizarse con Aseprite 2.3 Estudio de C#
07/02/2022	2.3 Estudio de C#
08/02/2022	2.3 Estudio de C#
09/02/2022	2.3 Estudio de C# 2.4 Organizar el entorno de Unity
10/02/2022	3.1 Descargar sprites
11/02/2022	3.2 Crear sprite protagonista
14/02/2022	3.3 Organizar sprites 3.5 Descargar música
15/02/2022	3.4 Preparar sprites
16/02/2022	4.1 Preparar canvas 4.7 HUD protagonista
17/02/2022	5.1 Input para controles 6.1 Prefab protagonista
18/02/2022	6.2 Mover protagonista 6.3 Animar protagonista
21/02/2022	6.3 Animar protagonista 6.4 Detectar protagonista
22/02/2022	7.1 Sistema de vida general
23/02/2022	7.2 Sistema de vida protagonista
24/02/2022	7.3 Revivir protagonista
25/02/2022	7.5 Asignar vida al protagonista 7.4 Actualizar interfaz vida
28/02/2022	7.4 Actualizar interfaz vida 8.1 Sistema de maná general

Marzo

Día	Tarea
01/03/2022	8.1 Sistema de maná general 8.3 Asignar maná al protagonista
02/03/2022	8.2 Actualizar interfaz maná
03/03/2022	9.1 Sistema de experiencia general
04/03/2022	9.1 Sistema de experiencia general
07/03/2022	9.3 Asignar experiencia al protagonista 9.2 Actualizar interfaz experiencia
08/03/2022	9.2 Actualizar interfaz experiencia 4.8 Panel accesos
09/03/2022	4.10 Panel estadísticas
10/03/2022	10.1 Sistema de estadísticas general
11/03/2022	10.1 Sistema de estadísticas general
14/03/2022	10.2 Crear ScriptableObject de estadísticas
15/03/2022	10.4 Asignar estadísticas al protagonista 10.3 Actualizar interfaz estadísticas
16/03/2022	10.3 Actualizar interfaz estadísticas 11.1 Sistema de atributos general
17/03/2022	11.1 Sistema de atributos general
18/03/2022	11.1 Sistema de atributos general 11.3 Asignar atributos al protagonista
21/03/2022	11.2 Actualizar interfaz atributos
22/03/2022	24.1 Crear capas 24.2 Crear tilemaps
23/03/2022	24.3 Diseñar mapa
24/03/2022	24.4 Crear aldeas
25/03/2022	24.5 Crear bosques
28/03/2022	24.7 Cámara 24.8 Limitaciones

Segundo Sprint (1 de Abril de 2022 a 27 de Mayo de 2022)

Abril

Día	Tarea
01/04/2022	24.9 Conectar mapa 24.10 Zona inicio
04/04/2022	12.1 Crear objetos
05/04/2022	12.2 Objeto poción vida 12.3 Objeto poción maná
06/04/2022	12.4 Otros objetos
07/04/2022	4.9 Panel inventario 13.2 Crear slots
08/04/2022	13.2 Crear slots
11/04/2022	13.2 Crear slots 13.1 Crear inventario
12/04/2022	13.1 Crear inventario
13/04/2022	13.1 Crear inventario 13.3 Interactuar con los slots
14/04/2022	13.3 Interactuar con los slots 13.4 Información objetos en inventario
15/04/2022	13.4 Información objetos en inventario 13.5 Interactuar con objetos en inventario
18/04/2022	13.5 Interactuar con objetos en inventario 13.6 Actualizar interfaz inventario
19/04/2022	13.6 Actualizar interfaz inventario 14.1 Animaciones NPCs
20/04/2022	14.1 Animaciones NPCs 14.2 Movimiento NPCs
21/04/2022	14.2 Movimiento NPCs 14.3 Crear NPCs
22/04/2022	14.3 Crear NPCs 14.4 Detectar NPCs
25/04/2022	4.13 Panel diálogo 4.14 Botón diálogo
26/04/2022	15.1 Sistema de diálogo general
27/04/2022	15.1 Sistema de diálogo general
28/04/2022	15.2 interacción NPC

Mayo

Día	Tarea
02/05/2022	15.3 Actualizar interfaz diálogos
03/05/2022	15.4 Crear y asignar diálogos
04/05/2022	16.1 Sistema de dinero general
05/05/2022	16.2 Actualizar interfaz dinero 20.1 Animaciones enemigos
06/05/2022	20.1 Animaciones enemigos 20.2 Movimiento enemigos
09/05/2022	20.3 IA enemigos
10/05/2022	20.4 Ataque enemigos
11/05/2022	20.5 Crear y configurar enemigos
12/05/2022	20.7 Asignar vida enemigos 20.8 Configurar vida enemigos
13/05/2022	20.6 Detectar enemigos 4.12 Panel arma equipada
16/05/2022	21.1 Crear armas
17/05/2022	21.2 Arma física espada 21.3 Arma física arco 21.4 Arma física mazo 21.5 Arma física hueso
18/05/2022	21.6 Arma mágica libro 21.7 Arma mágica bastón 21.8 Arma mágica vara 21.9 Arma mágica tridente
19/05/2022	21.10 Usar armas
20/05/2022	21.11 Actualizar interfaz armas 22.1 Seleccionar enemigos
23/05/2022	22.1 Seleccionar enemigos
24/05/2022	22.1 Seleccionar enemigos 22.2 Atacar enemigos
25/05/2022	22.2 Atacar enemigos 22.3 Animación magia
26/05/2022	22.4 Ser atacados por enemigos
27/05/2022	22.5 Ver daño

Tercer Sprint (1 de Junio de 2022 a 26 de Julio de 2022)

Junio

Día	Tarea
01/06/2022	4.19 Panel loot 23.1 Matar enemigos
02/06/2022	23.1 Matar enemigos 23.2 Loot enemigos
03/06/2022	23.2 Loot enemigos
06/06/2022	23.2 Loot enemigos 23.3 Configurar loot enemigos
07/06/2022	23.3 Configurar loot enemigos 23.4 Sistema de loot general
08/06/2022	23.4 Sistema de loot general 23.5 Actualizar interfaz loot
09/06/2022	23.5 Actualizar interfaz loot 23.6 Obtener loot
10/06/2022	23.6 Obtener loot 4.15 Panel misiones
13/06/2022	4.11 Panel misiones aceptadas 4.16 Panel misión completa
14/06/2022	17.1 Sistema de misiones general
15/06/2022	17.1 Sistema de misiones general
16/06/2022	17.2 Actualizar interfaz misiones
17/06/2022	17.3 Aceptar misiones
20/06/2022	17.4 Actualizar interfaz misiones aceptadas
21/06/2022	17.5 Actualizar interfaz misión completa 17.6 Conseguir recompensas
22/06/2022	17.6 Conseguir recompensas 17.7 Crear misiones
23/06/2022	17.7 Crear misiones 17.8 Crear y asignar diálogo misiones
24/06/2022	4.17 Panel tienda 18.1 Sistema de tienda general
27/06/2022	18.1 Sistema de tienda general
28/06/2022	18.1 Sistema de tienda general 18.2 Añadir objetos a tienda
29/06/2022	18.2 Añadir objetos a tienda 18.3 Comprar objetos
30/06/2022	18.3 Comprar objetos 18.5 Crear y asignar diálogo tienda

Julio

Día	Tarea
01/07/2022	18.4 Actualizar interfaz tienda
04/07/2022	4.18 Panel crafteo 19.1 Sistema recetas
05/07/2022	19.1 Sistema recetas 19.2 Sistema de crafteo general
06/07/2022	19.2 Sistema de crafteo general
07/07/2022	19.2 Sistema de crafteo general 19.3 Craftear recetas
08/07/2022	19.3 Craftear recetas
11/07/2022	19.3 Craftear recetas 19.5 Crear recetas
12/07/2022	19.5 Crear recetas 19.6 Crear y asignar diálogo crafteo
13/07/2022	19.4 Actualizar interfaz crafteo
14/07/2022	24.6 Crear edificio crafteo 4.2 Menú inicio
15/07/2022	4.3 Menú pausa 4.4 Botón de pausa
18/07/2022	4.5 Menú configuración 4.5 Menú fin
19/07/2022	25.1 Configuración menú inicio
20/07/2022	25.2 Configuración menú pausa 25.3 Configurar botón pausa
21/07/2022	25.4 Configurar menú configuración
22/07/2022	25.6 Configurar fin
25/07/2022	25.5 Configurar menú fin
26/07/2022	26.1 Añadir música

PLANIFICACIÓN DE COSTES

Costes directos

Los costes directos son los que se asignan específicamente a un producto o proyecto, en este caso al desarrollo del videojuego. El único coste directo de este proyecto será el salario del desarrollador, ya que, como veremos, los otros costes cubren elementos que se pueden usar para otros proyectos de forma simultánea.

Salarios

Para calcular el salario correspondiente a este trabajo voy a tener en cuenta que la única persona que lo desarrolla debe encargarse de todos los roles necesarios, por lo tanto tendrá un trabajo equivalente al Project Manager. Por otro lado, también hay que tener en cuenta que esta persona aún no tiene experiencia laboral, por lo que será un Project Manager Junior.

Según glassdoor.es^{iv}, el salario medio para el rol de un Project Manager Junior en España (lugar donde se está desarrollando este trabajo) es de **30.640€ anuales**. Sin embargo, ya que se trabajará en jornadas de 3 horas en lugar de en jornadas de 8 horas, es importante conocer el sueldo por hora para ser más precisos.

Para conocer el sueldo por horas, debemos dividir el sueldo anual por las horas que se trabajan a la semana (40 horas semanales) por las semanas que conforman el año (52 semanas). Por tanto, $30,640\text{€}/(40*52) = 14,73\text{€ por hora}$.

Por último, para conocer su salario bruto, debemos tener en cuenta que va a trabajar durante **300 horas**. Por lo tanto, $14,73\text{€}*300 = 4.419\text{€}$

Rol	Precio por hora	Horas de trabajo	Salario bruto
Project Manager	14,73€	300	4.419€

Tras conocer el salario bruto del desarrollador, podemos calcular los costes que se debe pagar a la Seguridad Social por este empleado. Para ello, hay que calcular el 29.9% del salario bruto: $4.419*0,299 = 1.321,28\text{€}$. Finalmente, podemos calcular el coste total que se asume por el trabajo de este desarrollador.

Rol	Salario bruto	Seguridad Social	Total
Project Manager	4.419€	1.321,28€	5.740,28€

Costes indirectos

Al contrario que en los costes directos, los costes indirectos no se asignan específicamente a un producto o proyecto, pero sí son necesarios para el desarrollo o producción de este. En este apartado entran los equipos informáticos, la oficina, el software y la formación de los trabajadores.

Amortización de los equipos

El costo del uso de los equipos se calcula haciendo uso de su precio de adquisición y teniendo en cuenta los años de amortización de este. Para un ordenador de sobremesa, la amortización suele ser de 5 años aproximadamente.

En el caso del ordenador usado para el desarrollo de este proyecto (Alienware Aurora R8) tuvo un precio de 1.000€, por lo que se divide entre los años de amortización para conocer el coste de su uso durante un año: $1.000\text{€}/5 = 200\text{€ anuales}$.

Por otro lado, para el desarrollo de este proyecto, no se usará el equipo durante un año entero. En concreto, y como se ha especificado en la planificación temporal, se usará durante el periodo comprendido entre el día 1 de Febrero de 2022 y el día 31 de Julio de 2022, lo que equivale a 5 meses.

Conocida la amortización anual del equipo y los meses de uso, podemos averiguar la amortización mensual y multiplicarla por los meses de uso para conocer el coste total de usar este equipo para este proyecto. Podemos calcular que $200\text{€}/12 = 16,67\text{€}$ mensuales y que $16,67\text{€} \times 5 = 83,35\text{€}$ es la amortización total del equipo en este proyecto.

Equipo	Tipo	Precio	Amortización
Alienware Aurora R8	Sobremesa	1.000€	83,35€

El resto de los elementos informáticos como las pantallas, altavoces, ratón o teclado no son amortizables, ya que sus valores individuales no sobrepasan los 300€.

Software

El motor gráfico elegido para el desarrollo del videojuego es Unity^v. A pesar de que Unity ofrece varios planes con distintos precios y prestaciones, podemos usarlo de forma gratuita para uso personal o para empresas que facturen menos de 100.000 dólares americanos al año. Yo en concreto usaré la licencia gratuita para estudiantes.

Por otro lado, también usaré un software para la creación de sprites. El software elegido es Aseprite^{vi}, cuyo precio es de 19,99€.

También me ayudaré de las prestaciones de otros softwares como Photoshop para editar fácilmente las spritesheets descargadas de internet. Concretamente, estaré usando la versión portable de Photoshop CS6^{vii}, la cual es gratuita.

Software	Precio
Unity 2021.1.22f1	0€
Adobe Photoshop CS6 Portable	0€
Aseprite	19,99€
Total: 19,99€	

Formación

Aunque no todos los proyectos necesitan realizar una inversión para la formación de sus desarrolladores en las tecnologías necesarias, este no es el caso. A pesar de tener conocimiento sobre muchos softwares y lenguajes de programación, nunca había desarrollado usando Unity ni C# (lenguaje de programación necesario en Unity). Por lo tanto, he realizado un desembolso en distintos cursos, desde más básicos hasta más avanzados, para comprender bien estas tecnologías y desenvolverme fácilmente con ellas para la creación de este proyecto.

También he usado para formación la propia documentación de Unity^{viii}, pero al ser gratuita no la he añadido en los costes.

En total, los cursos los cursos realizados han sido tres, a través de las plataformas Domestika^{ix} y Udemy^x. Estos cursos han sido sobre introducción a Unity 2D, introducción de desarrollo de videojuegos RPG y diseño avanzado de videojuegos RPG.

Curso	Precio
Introducción a Unity 2D	20,70€
Aprender a crear videojuegos RPG	12,99€
Diseño avanzado de videojuegos RPG	25,96€
Total: 59,65€	

Local de trabajo

Ya que este trabajo lo va a desarrollar una sola persona y puede trabajar cómodamente desde casa, no es necesario el desembolso que supone una oficina para trabajar en ella. Sin embargo, aunque trabajar en casa pueda hacer el proyecto más barato, el sitio de trabajo sigue sin ser gratis: es necesario tener en cuenta el alquiler mensual de la vivienda donde se desarrollará el trabajo. En este caso, el alquiler asciende a 400€ mensuales, con luz y agua incluidas.

Como ya se ha explicado anteriormente, el trabajo se desarrollará durante 5 meses, por lo que para conocer el presupuesto total para el local de trabajo se debe multiplicar el precio mensual por estos 5 meses: $400\text{€} \times 5 = \mathbf{2.000\text{€}}$

Lugar	Alquiler mensual	Duración	Total
Casa particular	400€	5 meses	2.000€

Coste total

Sumando todos los costes directos e indirectos podemos averiguar el coste total del proyecto.

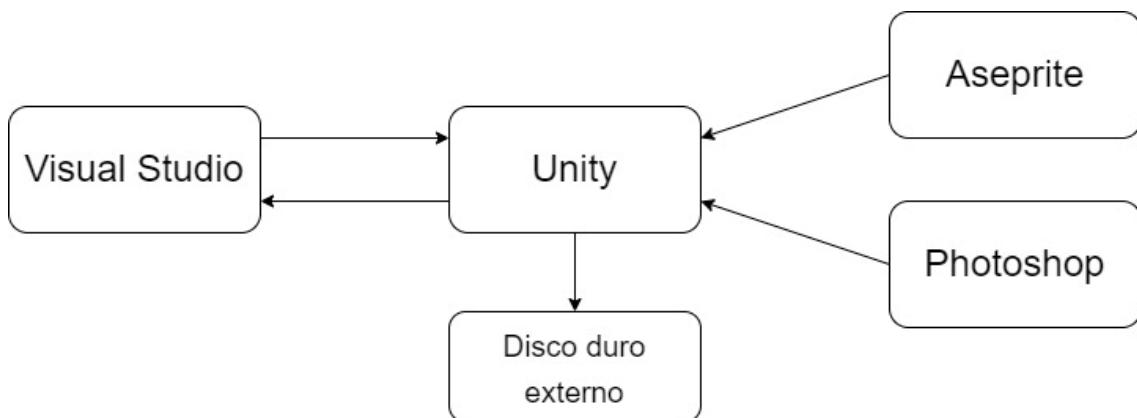
Concepto	Tipo	Coste
Salario	Directo	5.740,28€
Amortización de los equipos	Indirecto	83,35€
Software	Indirecto	19,99€
Formación	Indirecto	59,65€
Local de trabajo	Indirecto	2.000€
		Total: 7.903,27€

ARQUITECTURA DEL SISTEMA

Para desarrollar el videojuego necesitaré un motor de creación de videojuegos, un IDE de programación para realizar los scripts y los programas con los que realizar los elementos de arte, como los sprite, que usaré para la construcción del videojuego.

Los motores de videojuegos suelen incluir su propio repositorio local, y no hay necesidad de usar otro o conectarlo a internet si vamos a trabajar solos. En mi caso, usaré el repositorio de Unity (motor elegido, como se describe más adelante en este mismo apartado) y mantendré una copia de seguridad en un disco duro externo, siempre es bueno prevenir. Como IDE usaré Visual Studio, ya que se integra bien con Unity. Para realizar los elementos de artes, usaré Photoshop y Aseprite.

Añado un esquema con los elementos que componen la arquitectura del sistema y cómo interactúan entre ellos.



Cada uno de los elementos se describen a continuación.

Motor gráfico

Lo más importante en el entorno para desarrollar un videojuego, es tener un motor. Estos motores nos permiten ejecutar rutinas de programación y nos permiten diseñar y crear videojuegos con ellos.

Hoy en día, muchos estudios crean sus propios motores de videojuegos, pero también tenemos muchos Softwares que podemos descargar y utilizar directamente y que ya incluyen funcionalidades como físicas, inteligencia artificial o animación.

Elección del motor

Para esta propuesta de TFG, podía usar varias opciones gratuitas como Unity o Unreal. Antes de tomar una decisión de cuál usar, he investigado un poco de cada uno de ellos y los he comparado. Una de las páginas que me ha sido útil para esta comparación es soloempleo^{xi}.

Optimización

Unity permite realizar videojuegos para plataformas móviles de forma más fácil, ya que optimiza más los juegos desarrollados que Unreal. Aunque mi idea para este videojuego

no es hacerlo en una plataforma móvil, me gusta tener la opción de poder hacerlo en un futuro. Además, como soy principiante en el desarrollo de videojuegos, prefiero tener un motor que me ayude a optimizar más las cosas.

Curva de aprendizaje

Según la experiencia de los usuarios, la curva de aprendizaje de Unity hace que sea más fácil aprender cuando no tenemos ninguna experiencia que Unreal. Esta curva de aprendizaje en Unity no se debe solo a la estructura del motor, también por la cantidad de documentación disponible que podemos consultar.

Lenguaje de programación

Unity usa C#, mientras que Unreal usa C++ o Blueprints. Aunque no tenía experiencia con ninguno de ellos, estuve mirando el funcionamiento de estos tres lenguajes y C# me pareció más cómodo a primera vista.

Elección

Por estas razones, y en especial por la curva de aprendizaje, decidí usar Unity para el desarrollo del videojuego de mi TFG.

Configuración del motor

Una vez elegido el motor, es hora de instalarlo y hacer las configuraciones necesarias para su uso.

El primer paso es acceder a la web oficial de Unity y descargar Unity Hub^{xi}, que es el asistente de instalación. En mi caso, descargué la versión para Windows.

1. 1. Descargar Unity Hub

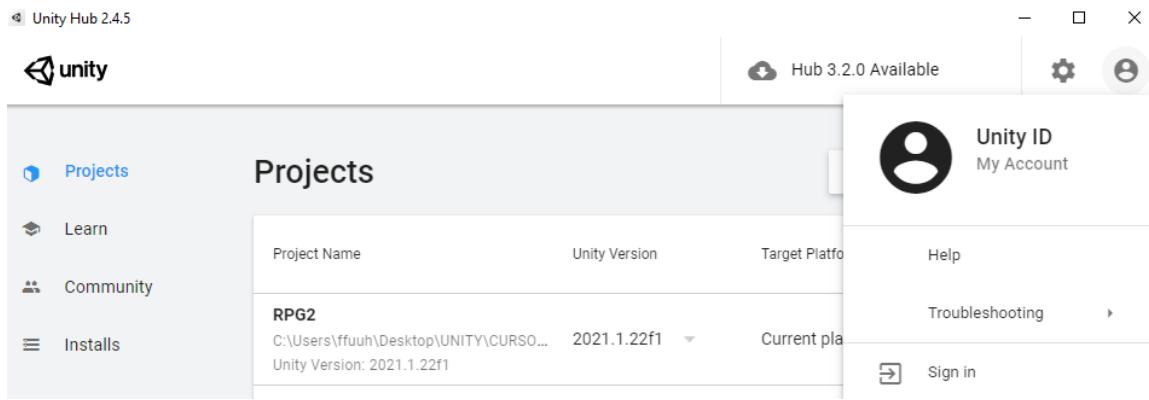
Sigue las instrucciones en la pantalla para obtener orientación para el proceso de instalación y configuración.

[Descargar para Windows](#) [Descargar para Mac](#) [Instrucciones para Linux](#)

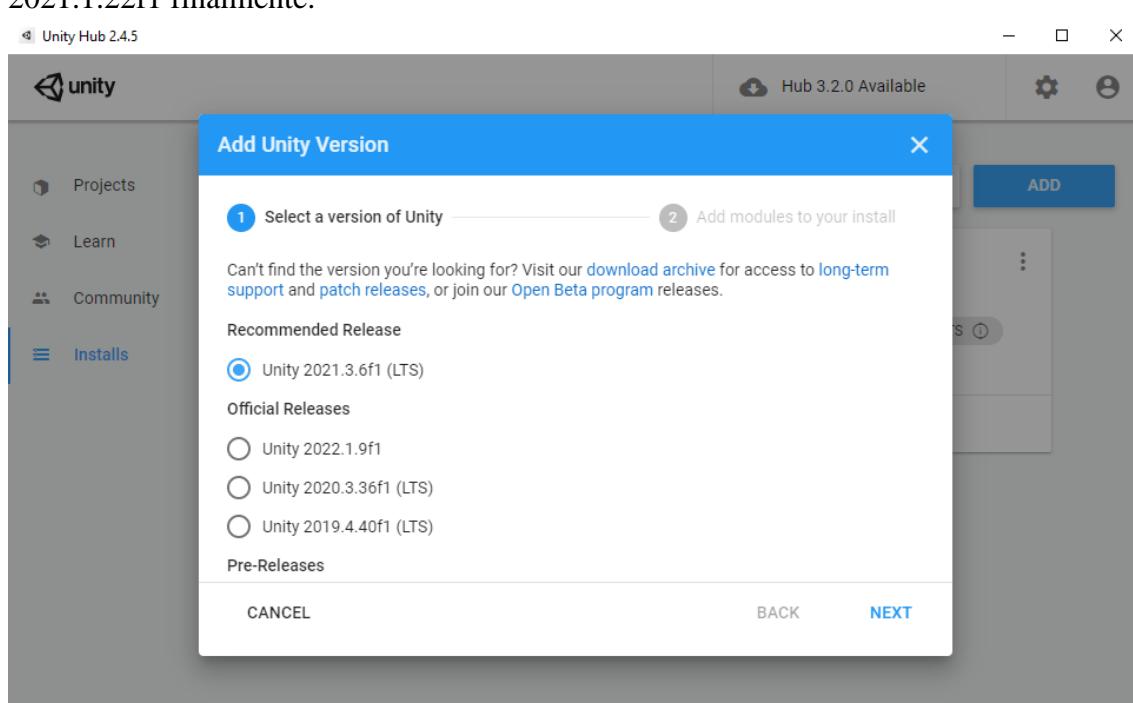
Ilustración 53 Panel de descarga de Unity

Simplemente hay que ejecutarlo y seguir las instrucciones que aparecen en pantalla.

Una vez instalado, abrimos Unity Hub e iniciamos sesión. Yo tuve que crear una cuenta nueva porque no tenía ninguna, pero la creé con mi correo corporativo de la Universidad de Sevilla para tener el plan gratuito para estudiantes.



A través de Unity Hub, en la pestaña de instalaciones, podemos elegir la versión de Unity que queremos descargar. En mi caso, aunque probé varias, decidí usar la versión 2021.1.22f1 finalmente.



Por último, en la pestaña de proyectos de Unity Hub, podemos ver los proyectos de Unity que existen en nuestro ordenador y abrirlo; pero también podemos crear un proyecto nuevo. Cuando creamos este proyecto, nos dará varias opciones de plantillas, de las cuales depende la configuración inicial de Unity. En mi caso, seleccioné la plantilla de proyecto 2D.

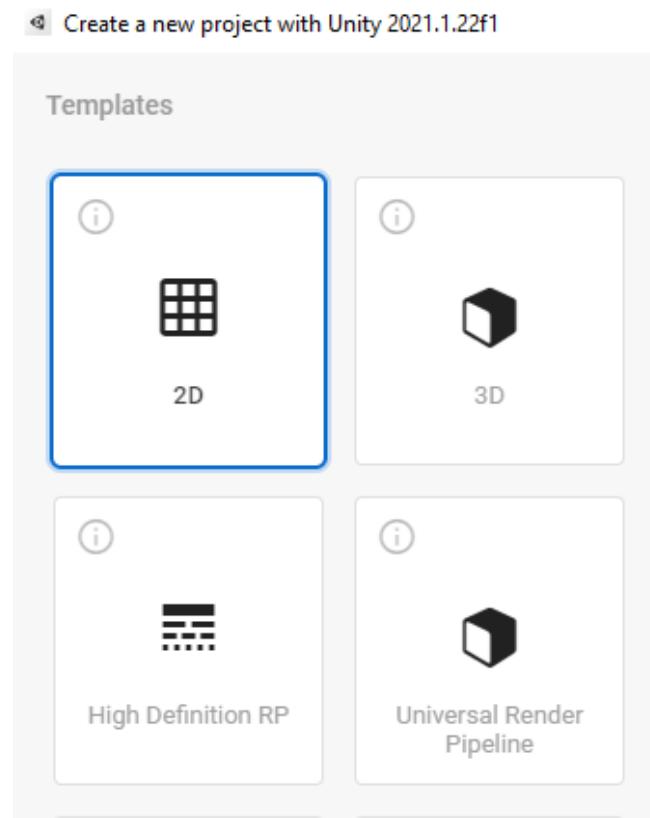


Ilustración 56 Unity Hub crear proyecto

Una vez que nuestro proyecto se haya creado, se abrirá con la versión de Unity que tengamos instalada. Podemos acceder a este proyecto siempre que queramos desde Unity Hub.

Con Unity ya abierto, podemos modificar las ventanas para tener a mano las que más usemos. Aunque esto se puede modificar en cualquier momento, yo he estado trabajando casi todo el tiempo con la misma configuración para estar más cómoda. He puesto dos ventanas grandes en el centro, una con el juego y otra con la escena y el animador. En la parte de abajo, he puesto el proyecto para poder acceder rápidamente a cualquier recurso. Por último, en la derecha, he puesto la jerarquía y el inspector.

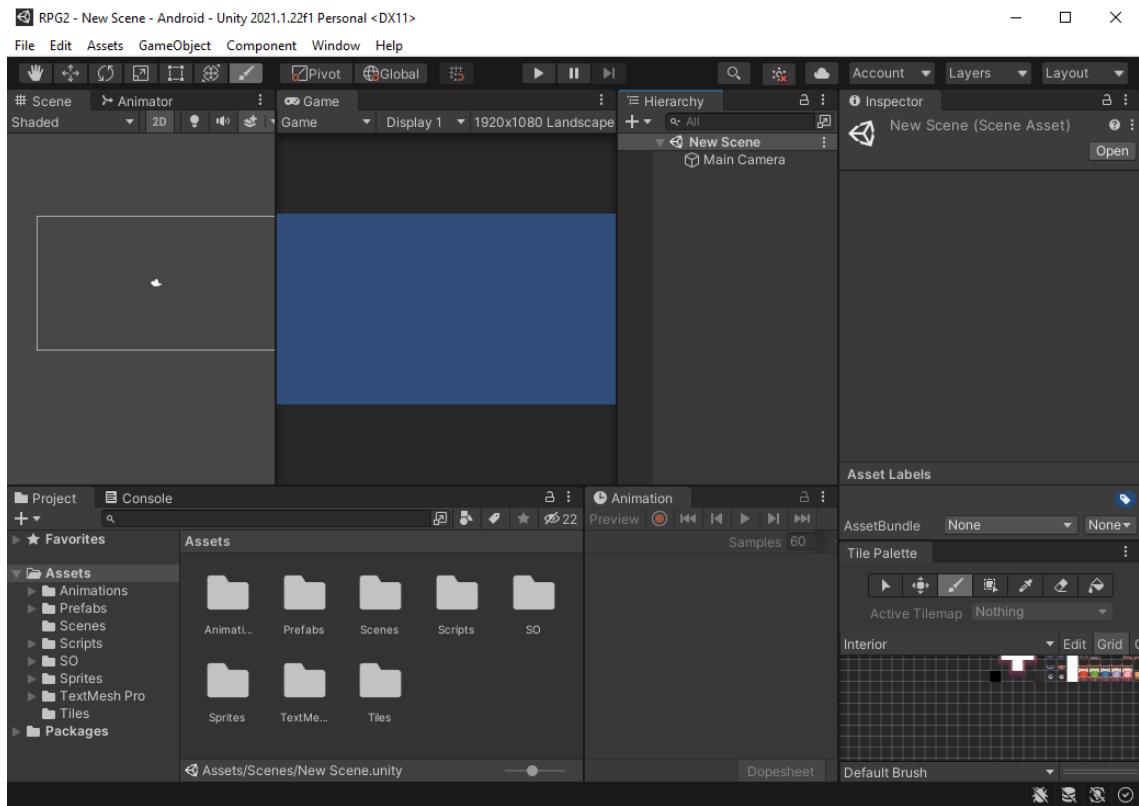


Ilustración 57 Unity ventanas

IDE

Para crear los scripts que utilizará Unity, necesitaba un IDE de programación. Al elegir Unity como motor, no tuve que darle muchas vueltas a qué motor usar, ya que Unity viene ya con el IDE Visual Studio, con el que es perfectamente compatible.

Aunque en algunos casos es necesario realizar algunas configuraciones o instalar este IDE aparte, no fue mi caso, se quedó todo correctamente instalado y funcional simplemente con instalar Unity.

Programas de arte

Además de Unity y el IDE, también se debe tener en el entorno Aseprite y Adobe Photoshop CS6 (portable, en mi caso en un disco duro extraíble), aunque son menos importantes para los aspectos técnicos de este proyecto.

Aseprite se puede descargar desde su propia página web tras pagar el precio de venta, y se instala rápidamente siguiendo los pasos del instalador. No hace falta realizar ninguna configuración previa para trabajar con él.

En cuanto a Photoshop, para usar la versión portable se puede descargar desde la página de Adobe sin tener que realizar ningún pago. Tampoco es necesario configurar nada previamente para usarlo. En mi caso, y gracias a que el programa es portable, lo guardaré en un disco duro extraíble además de en el propio equipo de trabajo.

Copia de seguridad

Ya que trabajaré de forma individual con Unity, haciendo uso de su repositorio local, guardaré una copia de seguridad que actualizaré al final de cada jornada en un disco duro extraíble para evitar posibles pérdidas del trabajo realizado.

DESARROLLO

Sprint 1

La primera tarea, y la más importante para desarrollar el videojuego, es instalar y configurar Unity; tal como se describe en el apartado de *Entorno*. He desarrollado esta tarea siguiendo los pasos indicados en la propia web de Unity. Aunque en algunos casos es necesario instalar Visual Studio aparte para poder programar en Unity, yo no tuve que hacerlo (se descargó automáticamente).

Tras crear un nuevo proyecto 2D, configuro otras propiedades teniendo en cuenta que mi juego será un RPG píxel para Windows. Tuve que cambiar las configuraciones de la build para seleccionar la plataforma para la que quería desarrollar el juego (Windows) y la resolución y forma de la pantalla en la que está pensado jugarlo.

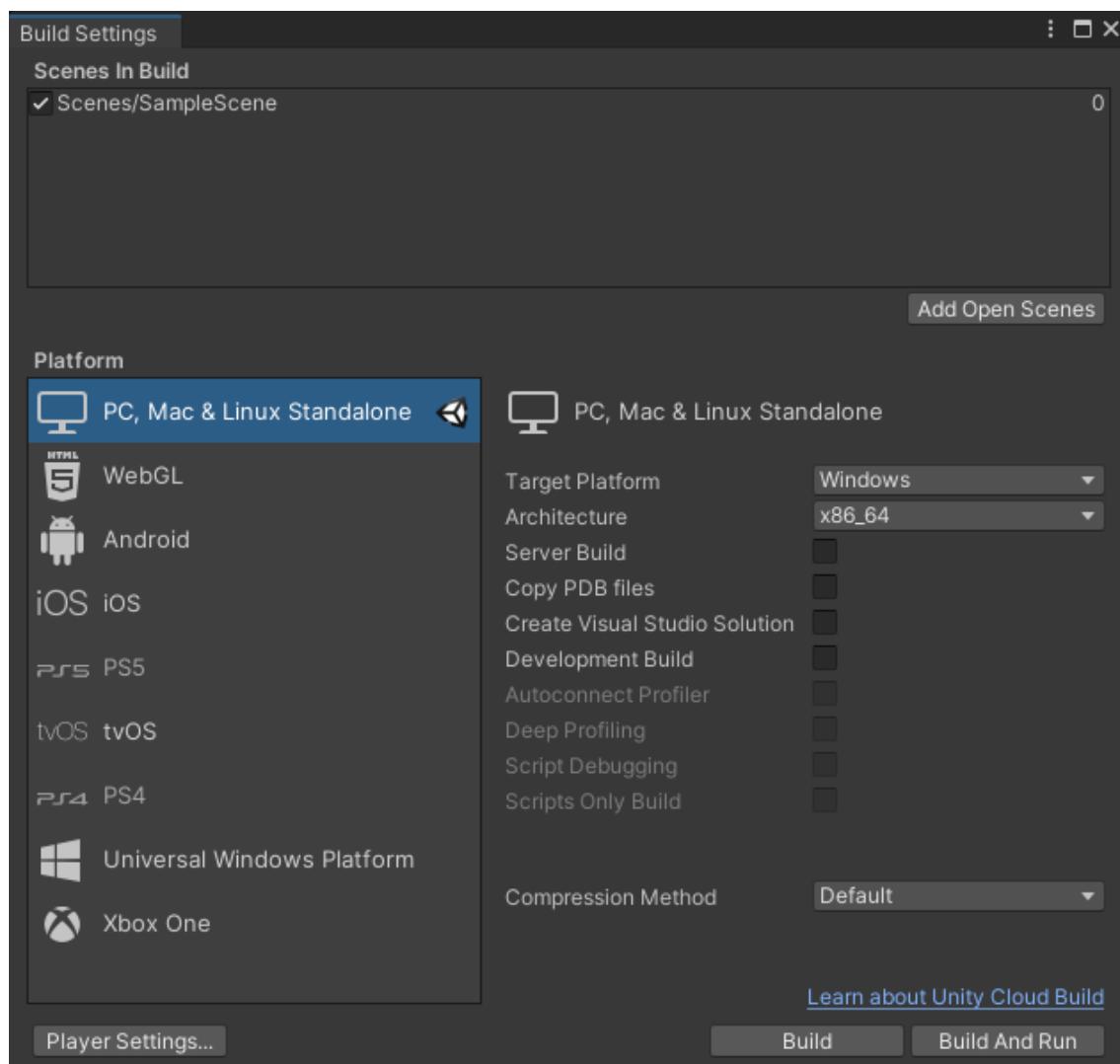


Ilustración 58 Build Settings plataforma

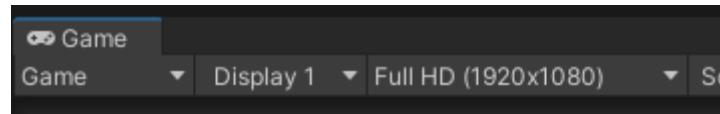


Ilustración 59 Resolución pantalla

A continuación, edité la configuración del proyecto para indicar el nombre del juego y la resolución y presentación (concretamente, que el juego se ejecute en pantalla completa).

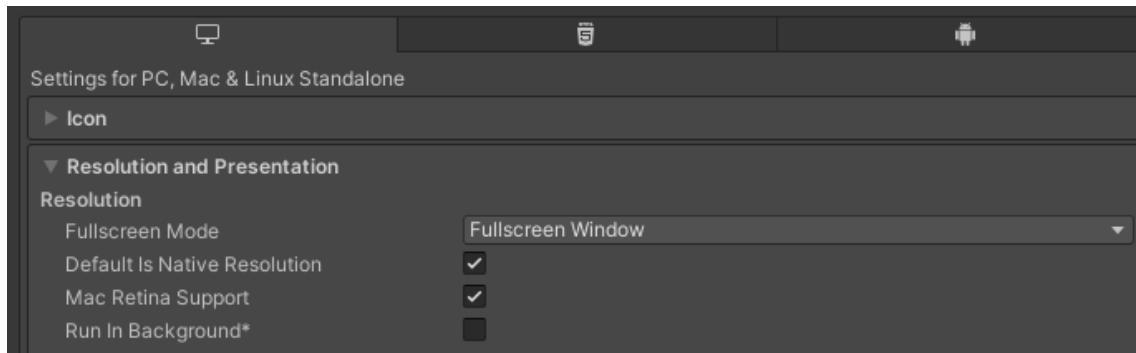


Ilustración 60 Project Settings resolución

Tras preparar Unity, instalé Aseprite y me aseguré de tener a mano la versión portable de Adobe Photoshop CS6.

Con todo instalado, consulté documentación disponible sobre C# y algunos tutoriales online para familiarizarme con este lenguaje de programación y dar mis primeros pasos con él.

Una vez que todo estaba preparado y tenía unas nociones básicas de todas las tecnologías necesarias, era hora de organizar el entorno de trabajo dentro de Unity. En la parte principal, puse las ventanas de jerarquía, juego y escena. La ventana de escena se puede cambiar por la de animator.

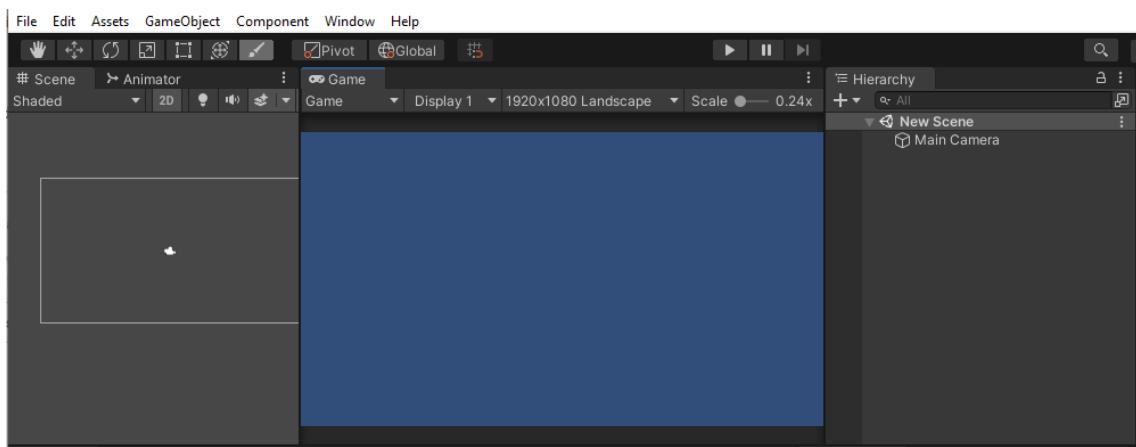


Ilustración 61 Entorno Unity parte superior

Justo abajo, para poder acceder cómodamente, situé las ventanas de proyecto y de animación. La ventana de proyecto se puede cambiar por la consola para ver el

funcionamiento del código cuando se ejecuta el juego o poder ver los errores. Además, creé varias carpetas en la ventana de proyecto para organizar todos los elementos: animaciones, prefabs, escenas, scripts, objetos encriptables, sprites, TextMesh Pro y tilemaps.

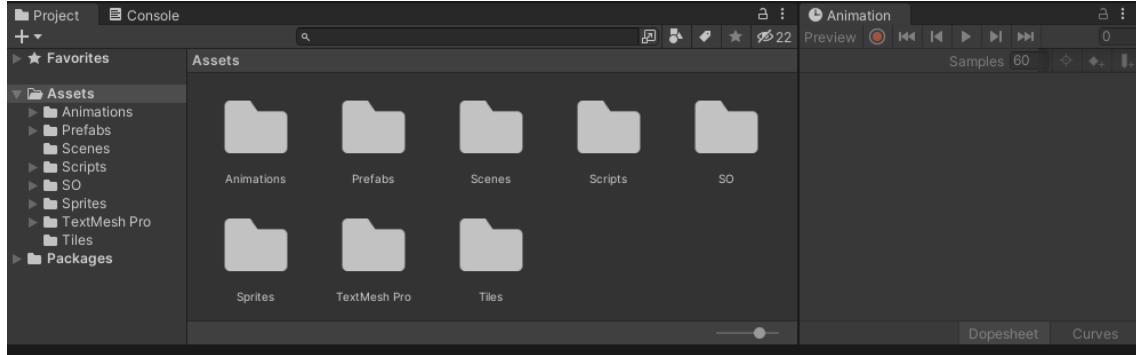


Ilustración 62 Entorno Unity parte inferior

A la derecha, puse las ventanas de inspector y la paleta de tilemaps, para poder ver información de los elementos y “pintar” en la escena de forma cómoda.

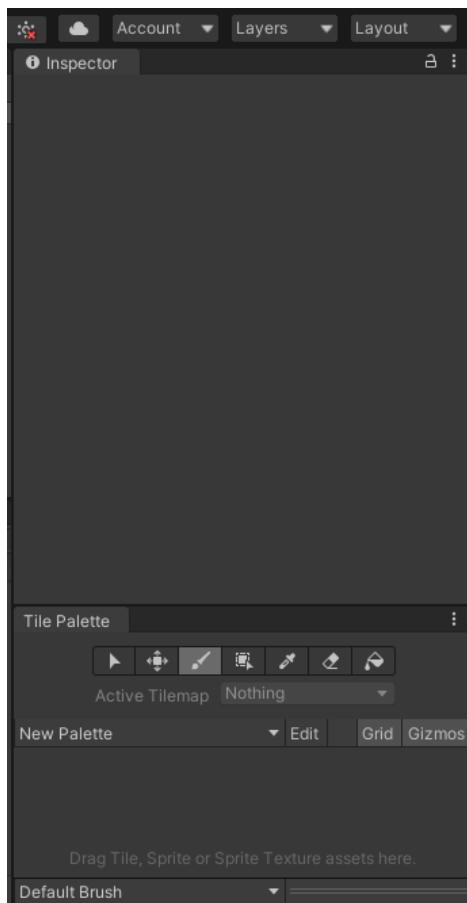


Ilustración 63 Entorno Unity parte derecha

Tras organizar el entorno de Unity, empecé con algunas tareas de arte. Me informé sobre sitios donde poder descargar sprites de forma gratuita y busqué varias hojas de sprites que encajaran con el aspecto que quería darle a mi videojuego, además de que encajaran entre

ellos y tuvieran un estilo de píxel similar. Encontré estos recursos en páginas de distintos artistas dentro de la plataforma Itch.io^{xiii}

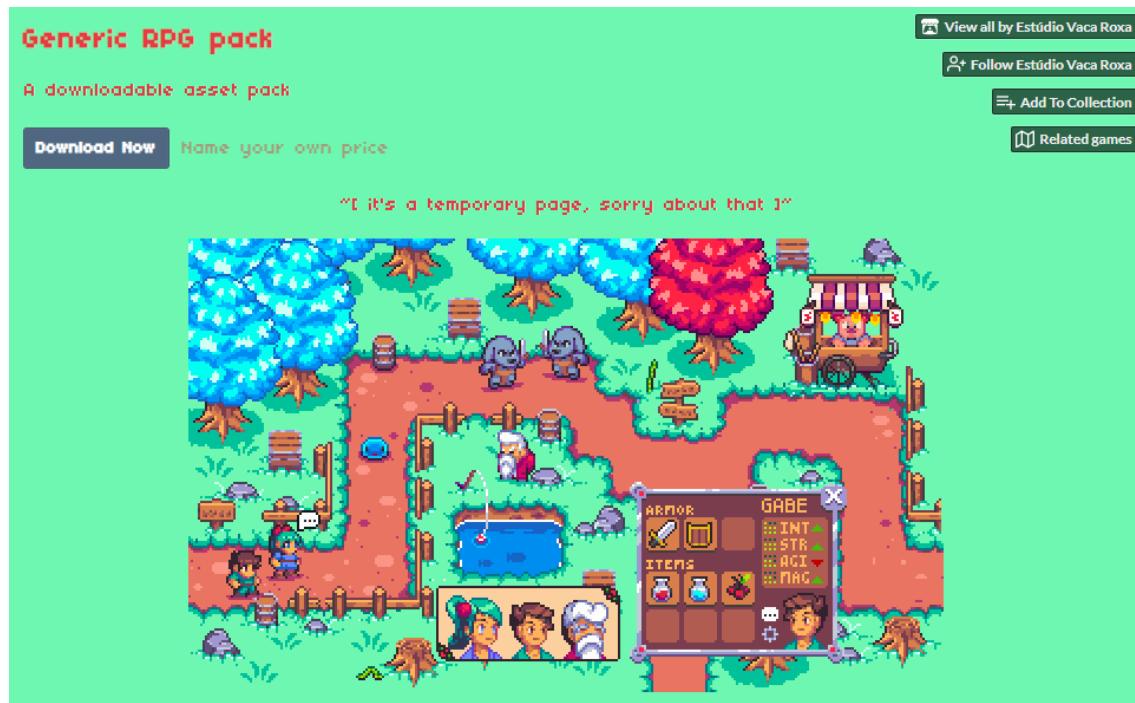


Ilustración 64 Página de descarga de sprites

Ayudándome de los sprites descargados y haciendo uso de Aseprite y Photoshop, creé los sprites de mi protagonista, siguiendo la estética del resto de sprites e intentando hacer que su aspecto fuera similar al mío, teniendo en cuenta las restricciones del arte píxel.



Ilustración 65 Sprites protagonista

Pero antes de poder usar todos los sprites, tenía que organizarlos en sus carpetas correspondientes de Unity para no perder tiempo buscándolos cuando quisiera usarlos. Además, tenía que configurar algunos parámetros de estos sprites a través de la ventana inspector.

En cuanto a la organización, creé cuatro carpetas dentro de la carpeta principal de Sprites: una para el entorno, otra para los objetos, otra para los personajes y una última para los elementos de la interfaz de usuario.



Ilustración 66 Carpetas sprites

Dentro de la carpeta de entorno, hice distinción entre los elementos de interior, de exterior (suelo) y de elementos decorativos que irían sobre el suelo.



Ilustración 67 Carpeta sprites entorno

Dentro de la carpeta de objetos, creé distintas categorías para los objetos de armas, ingredientes, pergaminos y tesoros.



Ilustración 68 Carpetas sprites objetos

Y por último, dentro de la carpeta de personajes, distinguí entre los sprites del protagonista y de otros personajes, como los distintos enemigos y los NPCs.

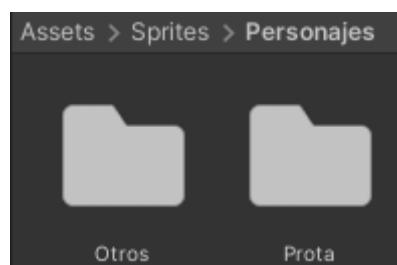


Ilustración 69 Carpeta sprites personajes

Para configurar los sprites, seleccioné todos los elementos a la vez dentro de cada una de estas carpetas, y desde el inspector cambié sus propiedades por las siguientes: tipo de textura sprite, modo de sprite múltiple (para poder dividir las hojas en distintos sprites), 16 píxeles por unidad (para que todos los elementos tuvieran un tamaño acorde) modo de

filtro sin filtro (para que Unity no intente suavizar los píxeles y se vea borroso) y ninguna compresión (los sprites de tipo píxel ocupan poco, no necesitan ser comprimidos).

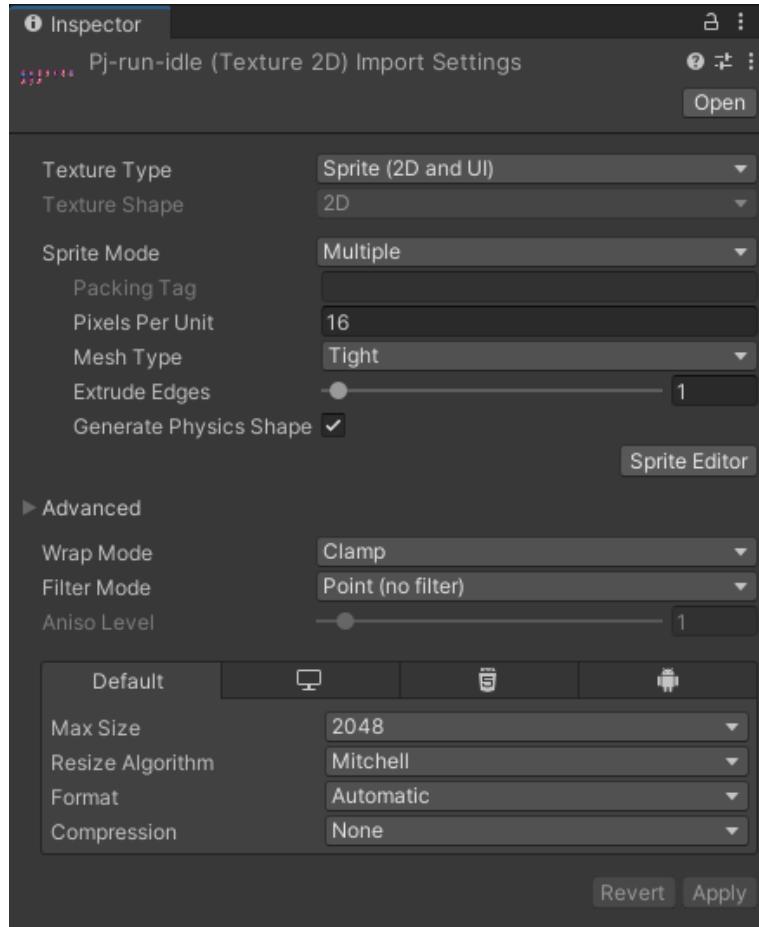


Ilustración 70 Inspector sprites

Por último, para terminar de preparar los sprites que usaría en el futuro, accedería uno por uno al editor de sprite para recortar cada hoja en los sprites individuales, recortando en cuadrados de 16 por 16 píxeles.

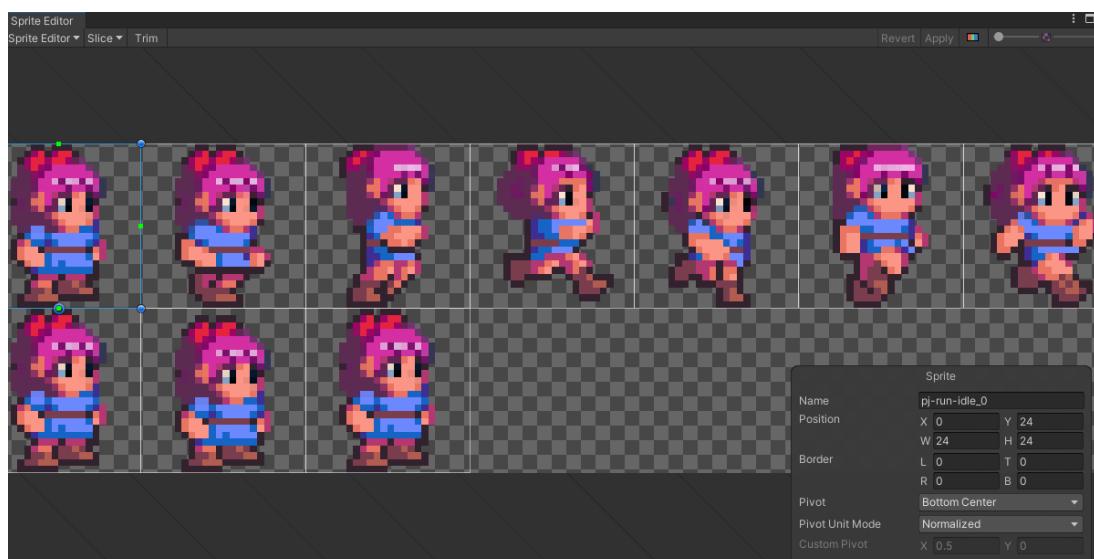


Ilustración 71 Sprite editor

Una vez que todos los elementos de arte estaban organizados y listos para ser usados, había que organizar el “lienzo”. En la jerarquía, creé un nuevo objeto de tipo canvas, donde irían todos los elementos de la interfaz.

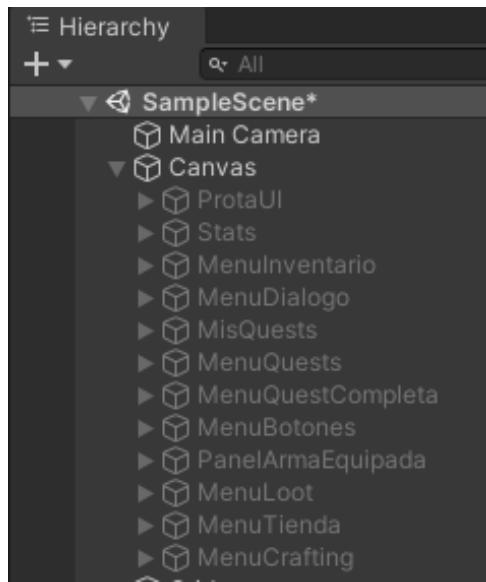


Ilustración 72 Jerarquía canvas

Protagonista

El primer elemento del videojuego que decidí empezar a desarrollar fue la protagonista, así podría familiarizarme con los controles o las animaciones. Además, desarrollar primero al protagonista me permitiría después poder probar el resto de las mecánicas con él.

Decidí empezar por la interfaz donde se verían reflejados los elementos del protagonista, como su vida o dinero, y más tarde vincularlos para que reflejen los datos reales una vez estuvieran programadas estas mecánicas.

Creé en el canvas un nuevo objeto para el HUD del protagonista. Este HUD estaría situado en la parte izquierda superior de la pantalla, siempre visible, para que el jugador pueda ver rápidamente los datos importantes sobre él. En este HUD aparecen una imagen del protagonista, su nivel, su dinero y barras rellenables para la vida, el maná y la experiencia. Además, en la barra de experiencia, aparecerá con texto el número de puntos de experiencia que se requiere para alcanzar el siguiente nivel y cuántos ha conseguido.

He usado el color verde para la barra de vida y el celeste para la barra de maná, ya que son colores que los jugadores de RPG ya tienen asociados a cada una de estas características.



Ilustración 73 HUD

Para poder manejar al protagonista es necesario establecer los controles del juego. Las últimas versiones de Unity traen ya instalado un sistema de inputs donde podemos seleccionar el tipo de controles que queremos usar y los botones o teclas con los que queremos realizar cada acción. En mi caso, lo configuré para jugar con el teclado y el ratón del ordenador.

Para configurar estos inputs, solo tenemos que crear mapas de acciones, y dentro de estos mapas crear las distintas acciones que queremos que se puedan realizar. En mi caso creé distintas acciones para el movimiento, interactuar y atacar.

En cuanto al movimiento, creé un sistema de vector 2D que usa las teclas W, A, S y D para moverse hacia arriba, hacia la izquierda, hacia abajo y hacia la derecha respectivamente.

Para interactuar con los elementos del juego como los NPCs, asigné la tecla E; y, para atacar a los enemigos, la barra espaciadora.

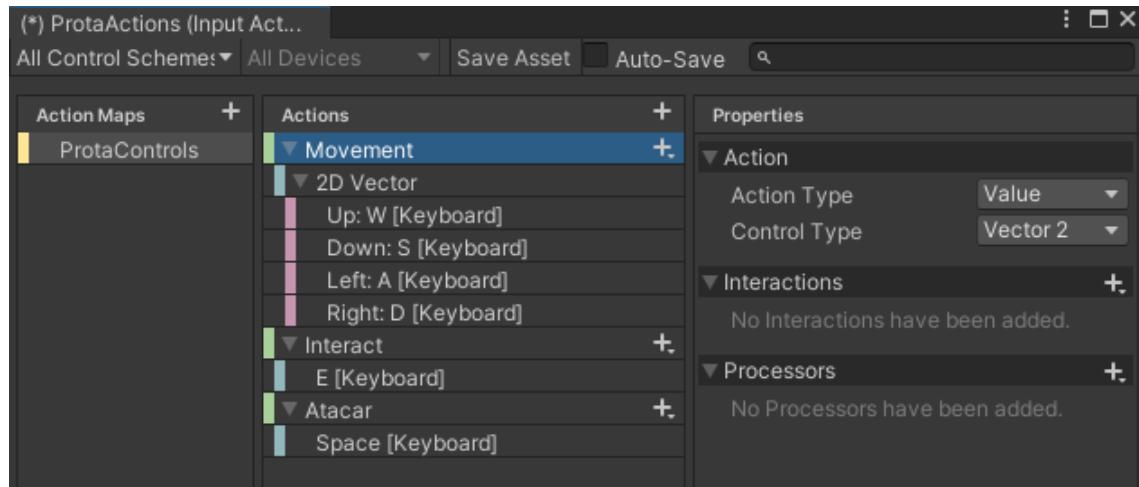


Ilustración 74 Inputs

Es hora de empezar con el protagonista. Lo primero, es crear un prefab para el protagonista en la carpeta correspondiente.



Ilustración 75 Prefab protagonista

Crear un prefab es una forma segura de editar a nuestro protagonista y que los cambios se guarden, ya que, si cambiamos las cosas directamente sobre el objeto de la jerarquía y necesitamos eliminar el objeto por algún motivo, todos los cambios se perderán. Si el objeto añadido a la jerarquía es un prefab, podemos eliminarlo o duplicarlo las veces que queramos y siempre tendrán la misma configuración. Es importante recordar que, cuando queramos editar el prefab, hay que abrirlo y asegurarnos de que editamos el prefab en sí, no el objeto de la jerarquía. Para asegurarnos de que estamos editando el prefab, tenemos que comprobar que en la parte superior de la jerarquía salga el nombre del prefab y el cubo en celeste.

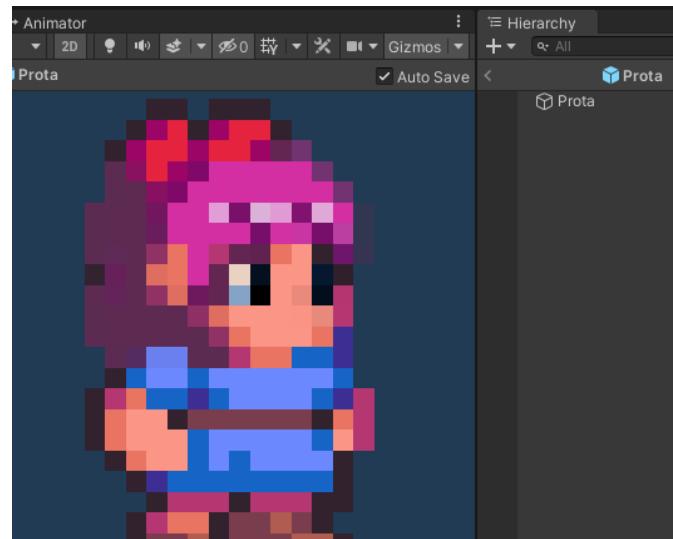


Ilustración 76 Jerarquía prefab

Unos de los primeros cambios que debemos hacerle al prefab, es asignarle el sprite del protagonista para poder distinguirlo, y asegurarnos de que está en el punto de coordenadas 0.

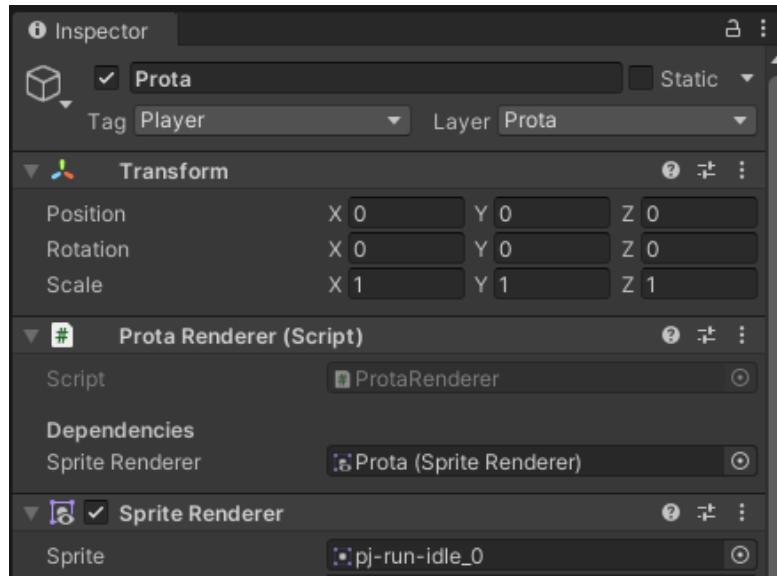


Ilustración 77 Inspector sprite protagonista

En este punto ya es hora de empezar a programar creando scripts. Para poder mover al protagonista, tuve que crear y asignarle un controlador haciendo referencia a los inputs creados anteriormente y a una velocidad que se puede editar desde el inspector.

```
private void FixedUpdate()
{
    rigidbody.velocity = _input * speed;
}

0 referencias
public void OnMovement(InputAction.CallbackContext value)
{
    _input = value.ReadValue<Vector2>();
```

Ilustración 78 Controlador

Además, creé un script para que el sprite del protagonista girara hacia la izquierda o a hacia la derecha al moverse en cada una de esas direcciones.

```

public void OnMovement(InputAction.CallbackContext value)
{
    Vector2 input = value.ReadValue<Vector2>();
    if (input.x > 0f && PlayerIsLookingLeft())
    {
        spriteRenderer.flipX = false;
    }
    else if (input.x < 0f && !PlayerIsLookingLeft())
    {
        spriteRenderer.flipX = true;
    }
}

2 referencias
public bool PlayerIsLookingLeft()
{
    return spriteRenderer.flipX;
}

```

Ilustración 79 Girar sprite

Además, para que estos scripts funcionen, hay que añadirlos al prefab del protagonista desde el inspector. También hay que añadirle un Rigidbody2D y asignarle el sistema de inputs creado anteriormente. En el Rigidbody2D añadido al protagonista, es importante bloquear que pueda girar en el eje Z, ya que es un juego 2D donde queremos que el sprite siempre se vea de frente. También hay que quitarle la gravedad para que el protagonista no se “caiga” de la escena, pues no es un juego de plataformas.

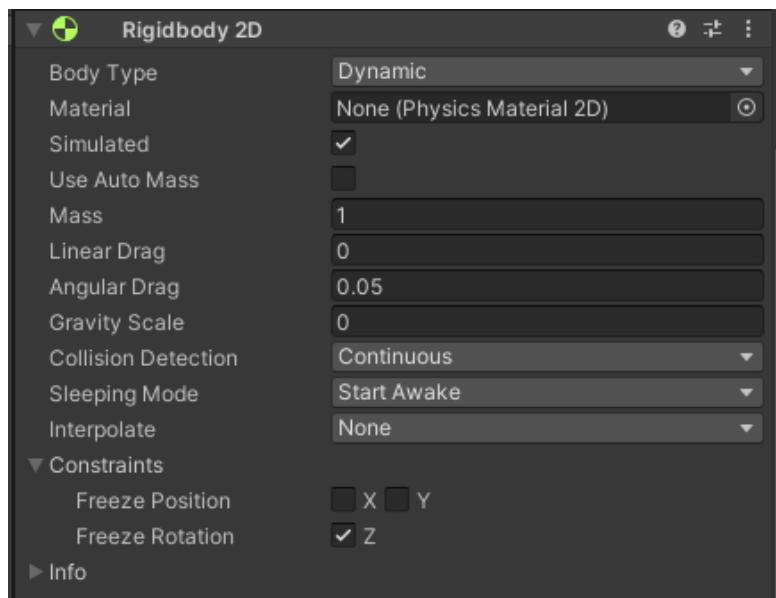


Ilustración 80 Rigidbody protagonista

Nuestro protagonista ya se puede mover libremente por la escena, así que toca animarlo. Creé dos animaciones distintas usando la ventana Animation: una para cuando el personaje está parado en un mismo sitio (idle), simulando una respiración, y otra para cuando el personaje estuviera en movimiento, de forma que parezca que está corriendo.

Para crear estas animaciones desde Animation, es tan fácil como crear una animación nueva y arrastrar los sprites que queremos que conformen la animación, seleccionando en el tiempo que queremos que cambie un sprite por otro.

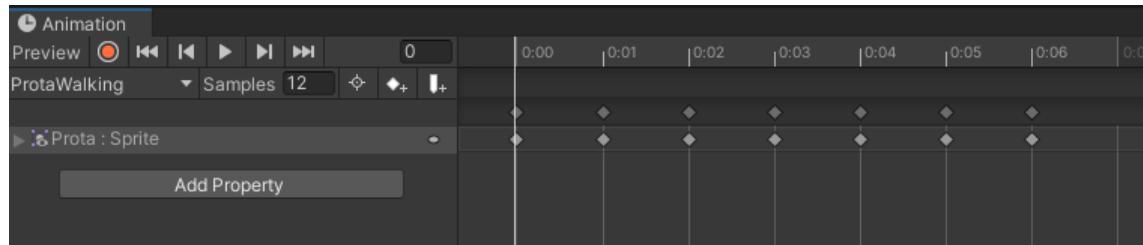


Ilustración 81 Animation protagonista

Una vez que las distintas animaciones están creadas, hay que acceder al árbol de animaciones en la ventana Animator. En esta ventana, seleccionaremos las transiciones entre cada una de las animaciones y las condiciones que se tienen que cumplir para que se dé esta transición.

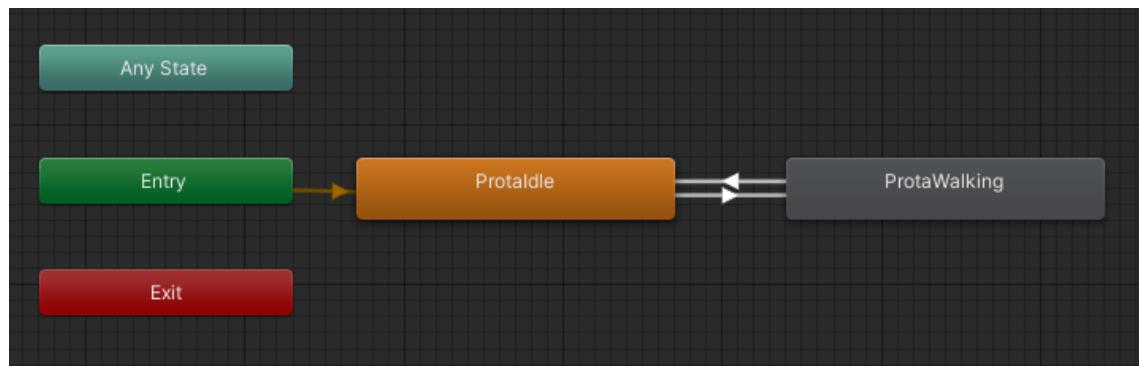


Ilustración 82 Animator protagonista

En mi caso, las condiciones para cambiar entre una animación y otra han sido muy simples: el protagonista empieza con la animación de idle, y sigue en ella mientras no se mueva. Al moverse, la animación cambia por la de correr. La animación de correr se sigue mostrando hasta que el protagonista vuelve a estar quieto.

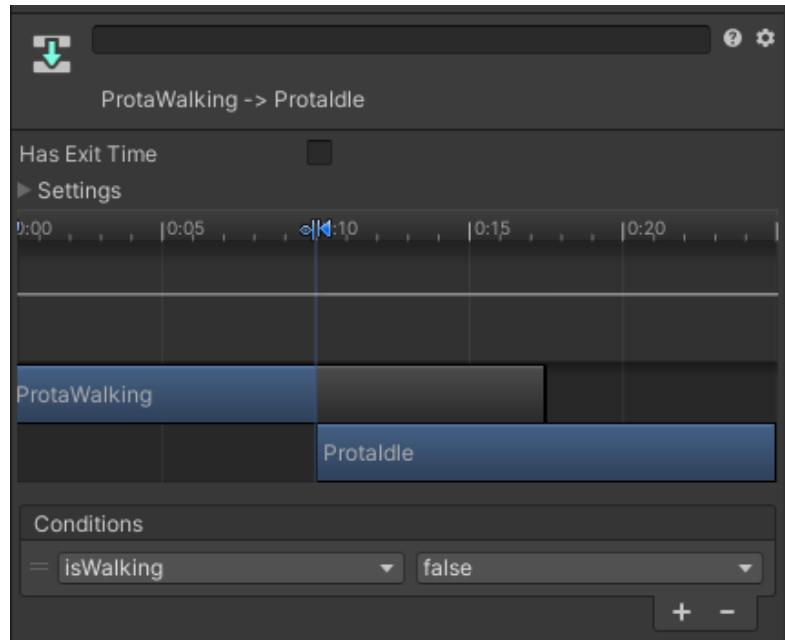


Ilustración 83 Inspector Animator transición andar a idle

Para poder comprobar si se dan estas condiciones, creé un script donde se definían cada una de ellas.

```
public class ProtaAnimation : MonoBehaviour
{
    [Header("Dependencies")]
    public Animator animator;

    0 referencias
    public void OnMovement(InputAction.CallbackContext value)
    {
        float input = value.ReadValue<Vector2>().magnitude;
        if (input > 0f)
        {
            animator.SetBool("isWalking", true);
        }
        else
        {
            animator.SetBool("isWalking", false);
        }
    }
}
```

Ilustración 84 Script condiciones Animator

Por último, para que las animaciones funcionen, hay que asignarle al prefab del protagonista el árbol de animaciones creado en el Animator y el script de las condiciones.

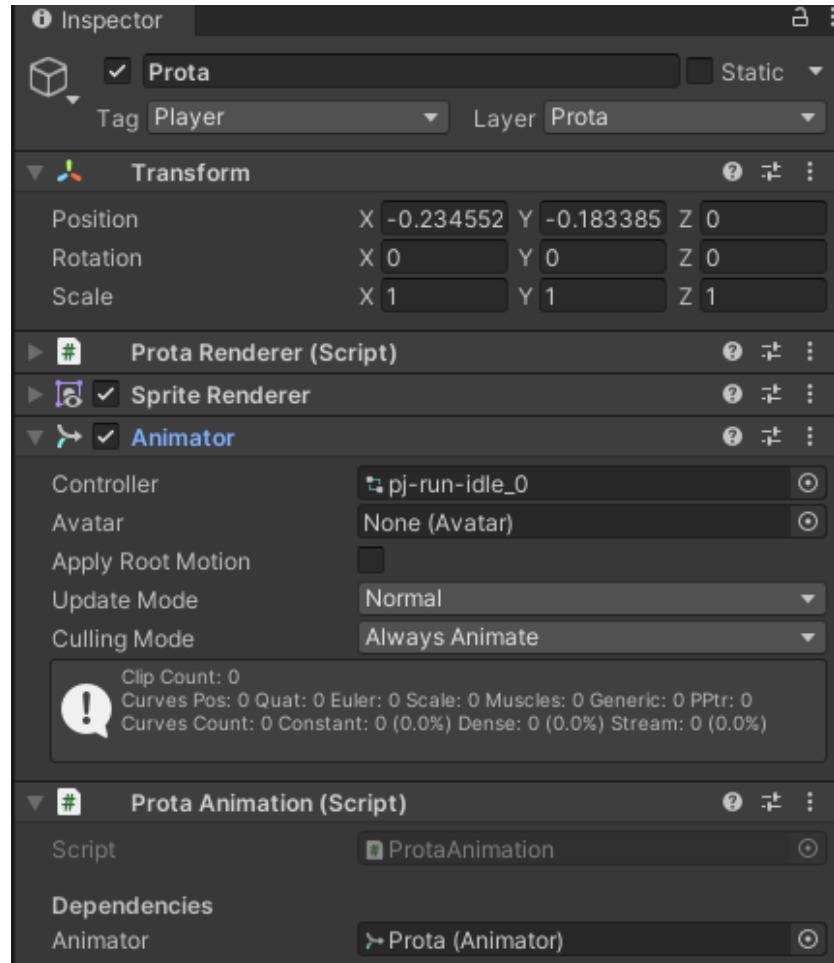


Ilustración 85 Inspector protagonista animación

Con los aspectos más básicos de nuestro protagonista listo, empecé a crear otras mecánicas un poco más completas, como pueden ser los sistemas de vida, de maná, de experiencia, etc.

Sistema de vida

La primera mecánica que creé fue la de vida. Para comenzar, creé un script para el sistema de vida general, que se le asignaría posteriormente a los enemigos y del que heredaría el script de vida del protagonista. Este script contiene variables como los puntos de vida máxima y los puntos de vida inicial, que se pueden editar para cada personaje que tenga asignado este script desde la jerarquía. Además, contiene un método que permite recibir daño, restando determinados puntos de vida a la salud actual.

```

public void RecibirPupa(float cantidad)
{
    if (cantidad <= 0f)
    {
        return;
    }
    if (Salud > 0f)
    {
        Salud -= cantidad;
        ActualizarBarraVida(Salud, saludMaxima);
        if (Salud <= 0f)
        {
            Salud = 0f;
            ActualizarBarraVida(Salud, saludMaxima);
            PersonajeMorido();
        }
    }
}

```

Ilustración 86 Script vida

El script de vida del protagonista, además de lo anterior, contiene métodos que le permite restaurar vida o que comprueba si sigue vivo. Al restaurar salud, solo lo hace si está vivo y nunca sobrepasando los puntos de salud máxima.

```

public void RestaurarSalud(float cantidad)
{
    if (Morido)
    {
        return;
    }
    if (PuedeSerCurado)
    {
        Salud += cantidad;
        if (Salud > saludMaxima)
        {
            Salud = saludMaxima;
        }
        ActualizarBarraVida( Salud, saludMaxima);
    }
}

```

Ilustración 87 Script vida protagonista

Cuando el personaje se muere (es decir, sus puntos de salud bajan de 1), se desactiva y se lanza un evento y para avisar a otros componentes de que se ha muerto y luego se revive. Al revivir al personaje, este se vuelve a activar. Además, su salud vuelve a ser su salud inicial. Más adelante, se realizarán más acciones al revivir al protagonista.

```

public void RevivirPersonaje()
{
    _boxCollider2D.enabled = true;
    Morido = false;
    Salud = saludInicial;
    ActualizarBarraVida(Salud, saludInicial);
}

2 referencias
protected override void PersonajeMorido()
{
    _boxCollider2D.enabled = false;
    Morido = true;
    EventoPersonajeMorido?.Invoke();
}

```

Ilustración 88 Script revivir protagonista

Además de devolver vida al protagonista al revivirlo, también quería devolverlo a un punto de spawn; concretamente al punto de inicio donde comienza el juego. Para poder solucionar esto, creé un manager que, al morir el protagonista, lo detectara y lo devolviera a la posición elegida, que seleccionaría más tarde desde el inspector. Creeé una carpeta dentro de scripts donde añadí este manager y donde añadiría también todos los que haría en un futuro, para poder acceder a ellos más cómodamente.

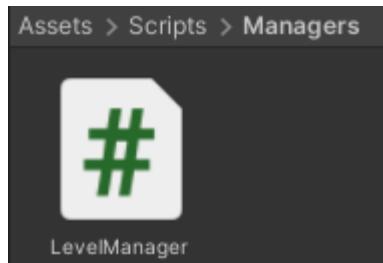


Ilustración 89 Carpeta managers

```

public class LevelManager : MonoBehaviour
{
    [SerializeField] private Transform _respawn;
    [SerializeField] private Prota _prota;

    private void Update()
    {
        if (_prota.ProtaVida.Morido)
        {
            _prota.transform.localPosition = _respawn.position;
            _prota.RevivirPersonaje();
        }
    }
}

```

Ilustración 90 Script level manager revivir protagonista

Dentro de la jerarquía, creé un objeto vacío donde añadir todos los managers de forma que siempre estén presentes en la escena y sean cómodos de acceder. Añadí un objeto con

el script del manager creado y asigné el punto de respawn. Para ver el punto de respawn en la escena, le asigné un ícono verde redondo.

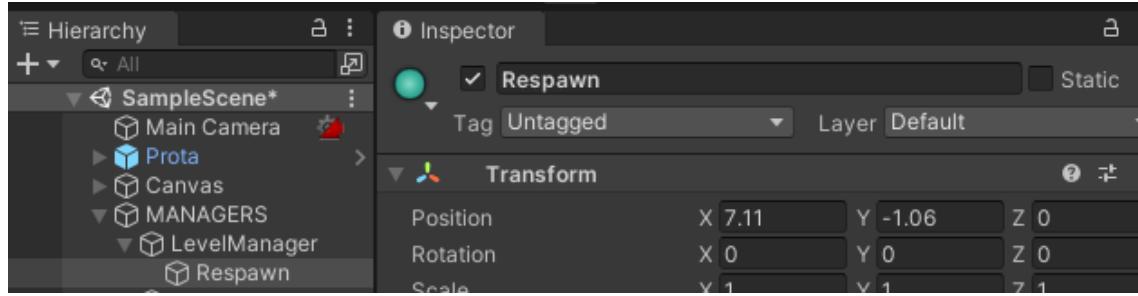


Ilustración 91 Inspector level manager

Ya tenemos creadas las mecánicas relacionadas con la vida de los personajes y la interfaz de los datos del protagonista, pero queda comunicar la interfaz creada con los datos reales de vida que correspondan al protagonista en cada momento. Para actualizar la interfaz, crearé otro manager, como el creado anteriormente para establecer el punto de respawn. Este manager se guardará también en la carpeta correspondiente.

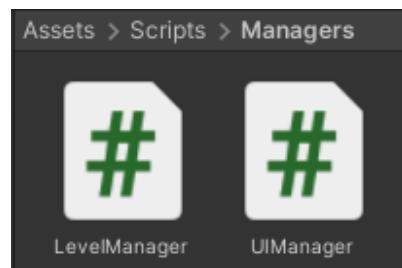


Ilustración 92 Manager UI en carpeta

También se añadirá un nuevo objeto en la jerarquía, dentro del objeto managers, al que se le asignará el script creado para el manager de la interfaz.

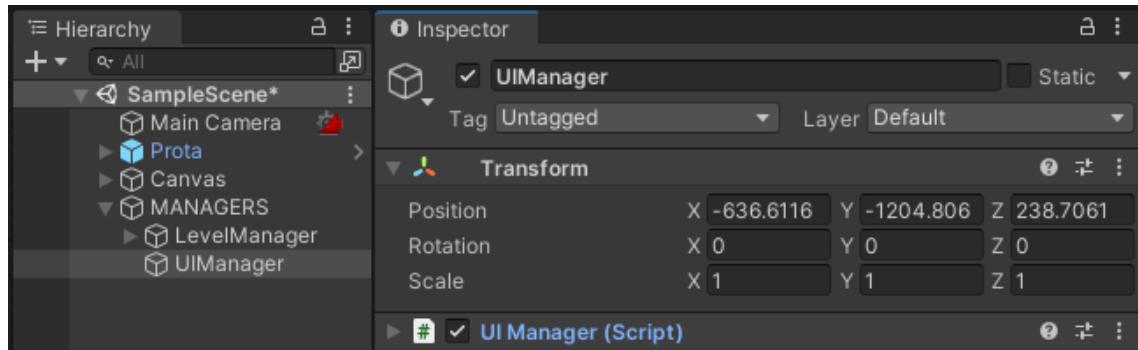


Ilustración 93 Jerarquía manager interfaz

Dentro del script del manager, creé una variable de tipo imagen, al cual le asignaría la imagen correspondiente a la barra de vida del HUD del protagonista más tarde desde el inspector.

```

public class UIManager : Singleton<UIManager>
{
    [Header("Configuration")]
    [SerializeField] private Image vidaProta;

```

Ilustración 94 Manager interfaz imagen barra vida

Teniendo acceso desde este script a la vida actual y a la vida máxima del protagonista, calculará el porcentaje de la barra de vida que tiene que llenar teniendo en cuenta los valores actualizados en cada momento del juego.

```

private void ActualizarUI()
{
    vidaProta.fillAmount = Mathf.Lerp(a:vidaProta.fillAmount,
        b:vidaActual / vidaMax, t:10f * Time.deltaTime);
}

```

Ilustración 95 Manager interfaz calcular relleno barra vida

Para asignarle a nuestro protagonista vida, abrí su prefab y le añadí el script de vida creado especialmente para él. Una vez añadido, le asigné valores de vida de prueba desde el inspector, concretamente 8 puntos de vida máxima y 8 puntos de vida inicial (para que empiece con la barra de vida rellena).

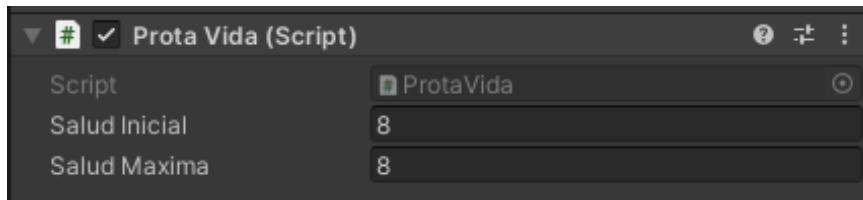


Ilustración 96 Inspector vida protagonista

Sistema de maná

La segunda mecánica que quise desarrollar en el juego es el sistema de maná, no tan necesario como el sistema de vida, pero muy típico en los juegos RPG, sobre todo en los que hay elementos mágicos. El maná es la energía mágica que gastan los personajes para lanzar hechizos o usar armas de tipo mágico. Otro elemento a tener en cuenta sobre el maná es que en casi todos los juegos RPG la barra de maná de los personajes se rellena poco a poco con el tiempo, sin necesidad de consumir pociones (aunque también puedan hacerlo). Esto lo tuve en cuenta a la hora de escribir los scripts.

Ya el sistema de maná será específico para el protagonista y no se les asignará a los enemigos, creé un solo script. Este script tiene variables como los puntos de maná máximos, los puntos de maná inicial y los puntos de recuperación automática, que se pueden configurar más tarde desde el inspector. Además, contiene métodos para poder usar el maná y restarlo al maná actual y para poder restaurarlo.

```

public void UsarMana(float cantidad)
{
    if (ManaActual >= cantidad)
    {
        ManaActual -= cantidad;
        ActualizarBarraMana();
    }
}

1 referencia
public void RestauraMana(float cantidad)
{
    if (ManaActual >= manaMaximo)
    {
        return;
    }
    ManaActual += cantidad;
    if (ManaActual > manaMaximo)
    {
        ManaActual = manaMaximo;
    }
    UIManager.Instance.ActualizarManaProta(ManaActual, manaMaximo);
}

```

Ilustración 97 Script maná gastar y restaurar

También contiene métodos para restaurar el maná a lo largo del tiempo sumando cada segundo los puntos establecidos desde el inspector, y un método para que el maná se reinicie que se llamará cuando se reviva el protagonista.

```

private void RecuperarMana()
{
    if (_protavaida.Salud > 0f && ManaActual < manaMaximo)
    {
        ManaActual += recuperacion;
        ActualizarBarraMana();
    }
}

1 referencia
public void ReiniciarMana()
{
    ManaActual = manaInicial;
    ActualizarBarraMana();
}

```

Ilustración 98 Script maná recuperar y reiniciar

Para mostrar en el HUD del personaje el maná actual en cada momento, seguí los mismos pasos anteriores para conectar los scripts de vida con el manager de la interfaz. Ahora el manager de interfaz también contiene los campos de la imagen correspondiente a la barra de maná y sus valores. Desde el inspector, asigná este campo la imagen correspondiente del HUD.

```

public class UIManager : Singleton<UIManager>
{
    [Header("Configuration")]
    [SerializeField] private Image vidaProta;
    private float vidaActual;
    private float vidaMax;
    [SerializeField] private Image manaProta;
    private float manaActual;
    private float manaMax;
}

```

Ilustración 99 Manager interfaz vida y maná

Y también añadí el método que calcula como llenar en cada momento esta barra, teniendo en cuenta el maná actual y el maná máximo.

```

private void ActualizarUI()
{
    vidaProta.fillAmount = Mathf.Lerp(a:vidaProta.fillAmount,
        b:vidaActual / vidaMax, t:10f * Time.deltaTime);
    manaProta.fillAmount = Mathf.Lerp(a: manaProta.fillAmount,
        b: manaActual / manaMax, t: 10f * Time.deltaTime);
}

```

Ilustración 100 Manager interfaz llenar barra maná

Para terminar con el sistema de maná, le añadí el script creado al prefab del protagonista, y le asigné valores de prueba para el maná inicial, el maná máximo y la recuperación. Podemos ver como los valores máximos e iniciales coinciden, para que al empezar la partida la barra esté llena. Por otro lado, la recuperación es mínima, pero más adelante crearé pociones de maná para que el personaje pueda recuperar maná más rápidamente.

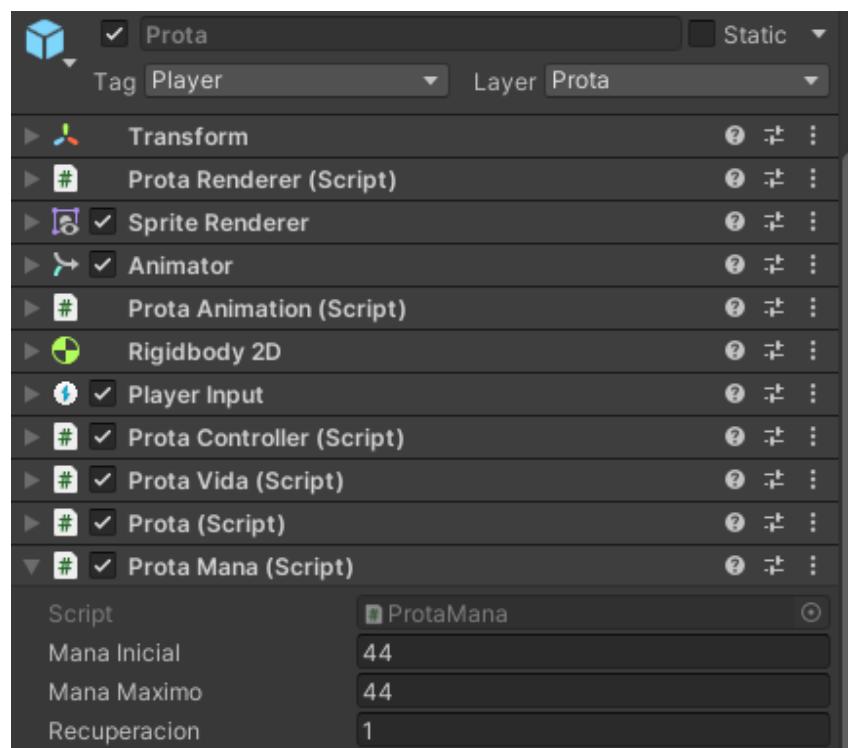


Ilustración 101 Inspector maná protagonista

Sistema de experiencia

El sistema de experiencia es otro que se usará exclusivamente para el protagonista, por lo que, de nuevo, solo creé un script para controlar este sistema. Este script contiene las variables de experiencia base, de incremento y de nivel máximo. La experiencia base hace referencia a los puntos de experiencia que el protagonista debe adquirir para subir del primer nivel (empieza en el nivel 1 siempre), mientras el incremento hace referencia a la cantidad por la que se multiplica la experiencia necesaria para subir de nivel cada vez que se alcanza uno nuevo, de forma que el incremento sea exponencial. A estas variables se les puede asignar un valor desde el inspector.

Además de las variables, este script contiene métodos para sumar experiencia a la experiencia actual del protagonista, actualizando con esto el nivel del personaje, no solo sus puntos de experiencia. También se actualizará la experiencia que se necesita para actualizar el siguiente nivel.

```
public void AñadirExp(float expObtenida)
{
    if (expObtenida > 0f)
    {
        float expRestanteSiguienteNivel = expParaSiguienteNivel - expActualTemporal;
        if (expObtenida >= expRestanteSiguienteNivel)
        {
            expObtenida -= expRestanteSiguienteNivel;
            expActual += expObtenida;
            ActualizarNivel();
            AñadirExp(expObtenida);
        }
        else
        {
            expActual += expObtenida;
            expActualTemporal += expObtenida;
            if (expActualTemporal == expParaSiguienteNivel)
            {
                ActualizarNivel();
            }
        }
    }
    stats.ExpActual = expActual;
    ActualizarBarraExp();
}
```

Ilustración 102 Script experiencia añadir experiencia

En el caso del sistema de experiencia, como también es una de las características del personaje que aparecen en el HUD, también hay que conectar los datos del script con el manager de interfaz para poder mostrar la barra de experiencia actualizada. En este caso, además de hacer el método que calcula cuánto porcentaje de la barra hay que llenar, también actualiza los números correspondientes a la experiencia actual y a la experiencia hasta el siguiente nivel que se muestran en la barra de la interfaz.

Para actualizar los campos de texto correspondientes a estos números, me ayudé de TextMesh Pro, una de las utilidades disponibles en las últimas versiones de Unity.

```

private void ActualizarUI()
{
    vidaProta.fillAmount = Mathf.Lerp(a:vidaProta.fillAmount,
        b:vidaActual / vidaMax, t:10f * Time.deltaTime);
    manaProta.fillAmount = Mathf.Lerp(a: manaProta.fillAmount,
        b: manaActual / manaMax, t: 10f * Time.deltaTime);
    expProta.fillAmount = Mathf.Lerp(a: expProta.fillAmount,
        b: expActual / expParaSiguienteNivel, t: 10f * Time.deltaTime);
    expTMP.text = $"{expActual}/{expParaSiguienteNivel}";
}

```

Ilustración 103 Manager interfaz experiencia

Más tarde, volví a editar el prefab del protagonista para añadirle también el script correspondiente al sistema de experiencia. Podemos ver como nos deja editar desde el inspector los campos de experiencia base, incremento y nivel máximo.

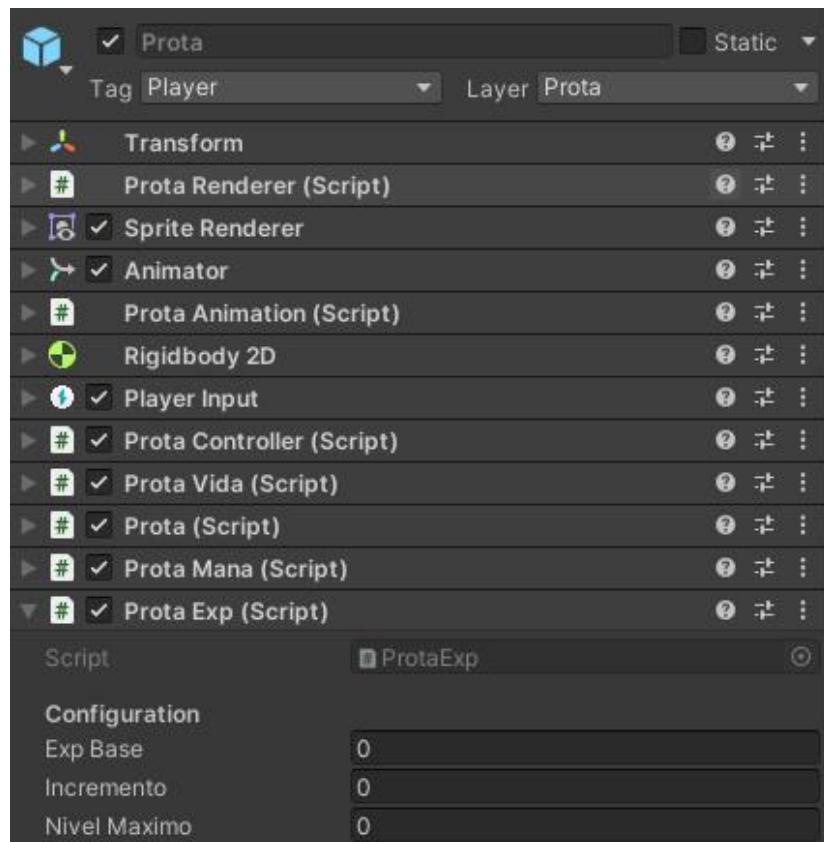


Ilustración 104 Inspector script experiencia

Sistema de estadísticas y atributos

Otro de los sistemas muy presentes en los videojuegos RPG es el de estadísticas. Es muy típico asignar varias características a los personajes, como el ataque o la defensa, y que podamos aumentarlas a lo largo de la partida, desarrollando al personaje cuando ganamos experiencia.

Creé un sistema que regulara tanto las distintas estadísticas del protagonista como los atributos que permitirían incrementar estas estadísticas. Pero, para que el jugador pudiera

acceder a las estadísticas y a los atributos, lo primero es tener un panel donde aparezcan estos datos.

Primero, desde el canvas, creé un nuevo objeto para hacer un panel con acceso a los menús principales con lo que podrá interactuar el jugador: inventario, estadísticas y misiones aceptadas con su progreso. Este panel estará presente a lo largo de todo el juego en la parte inferior izquierda de la pantalla. Para hacer el panel intuitivo, representé el botón de cada menú con elementos que suelen asociarse a ellos en los videojuegos: el inventario es representado por un zurrón, las estadísticas con músculos y las misiones con un pergamo.



Ilustración 105 Panel de acceso a menús

También desde el canvas creé un nuevo objeto para el propio menú de estadísticas. En la parte superior aparecen las distintas estadísticas asociadas al protagonista: nivel, experiencia total, daño, defensa, probabilidad de crítico, probabilidad de bloqueo, velocidad y puntos de experiencia necesarios para el siguiente nivel. En la parte inferior, los distintos atributos: fuerza, destreza e inteligencia, junto a los puntos disponibles y los botones para añadir estos puntos a cada uno de los atributos.

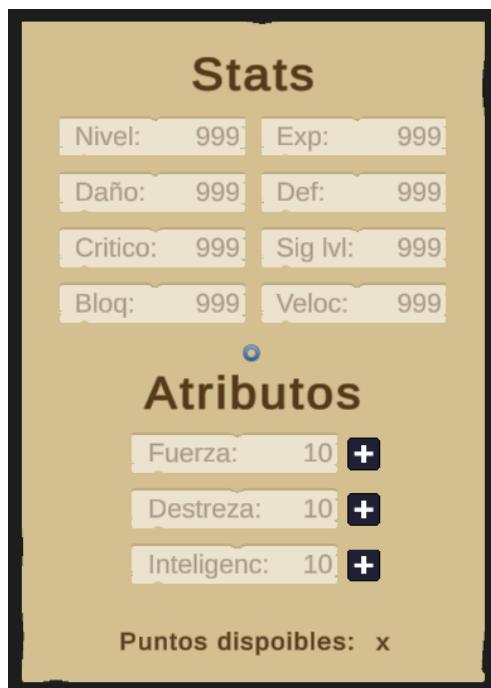


Ilustración 106 Menú estadísticas y atributos

Con las interfaces creadas, empecé a escribir los scripts necesarios. Para las estadísticas, creé un script a partir del cuál podría crear un objeto encriptable. Este objeto encriptable me ayudará a guardar unos valores predeterminados para cada estadística y volver a estos valores por defecto en otro momento. Este script contiene las variables para el nivel, la

experiencia actual, la experiencia para alcanzar el siguiente nivel, el daño, la defensa, la velocidad y los porcentajes de bloqueo y de daño crítico.

Las armas que crearé más adelante también tendrán estadísticas como las de daño o defensa, que se sumarán con las del protagonista. Por tanto el primer script creado para controlar las estadísticas contiene métodos para sumar estas estadísticas al equipar un arma al protagonista y otro para restarlas al desequiparla.

```
public void SumarStatsArmas(Arma arma)
{
    Pupa += arma.Pupa;
    PorcentajeCritico += arma.ProbabilidadCritico;
    PorcentajeBloqueo += arma.ProbabilidadBloqueo;
}

1 referencia
public void RestarStatsArmas(Arma arma)
{
    Pupa -= arma.Pupa;
    PorcentajeCritico -= arma.ProbabilidadCritico;
    PorcentajeBloqueo -= arma.ProbabilidadBloqueo;
}
```

Ilustración 107 Script estadísticas sumar y restar

Este script también cuenta con un método con los valores por defecto de cada una de las estadísticas, que nos permite restaurarlas y devolverles este valor.

```
public void ResetearStats()
{
    Nivel = 1;
    ExpActual = 0f;
    ExpParaSiguienteNivel = 0f;
    Pupa = 4f;
    Defensa = 2f;
    Velocidad = 4f;
    PorcentajeCritico = 0f;
    PorcentajeBloqueo = 0f;
}
```

Ilustración 108 Script estadísticas restaurar

El método anterior se puede ejecutar desde un botón que se mostrará en el inspector del objeto encriptable de estadísticas. Para ello, creé este método en un script aparte.

```

public override void OnInspectorGUI()
{
    base.OnInspectorGUI();
    if (GUILayout.Button(text: "Resetear stats"))
    {
        StatsTarget.ResetearStats();
    }
}

```

Ilustración 109 Script botón restaurar estadísticas

En la carpeta creada anteriormente para los objetos encriptables, creé el objeto de estadísticas. Podemos ver que, al crearlo, tiene ya asignados los valores por defecto de cada una de las distintas estadísticas.



Ilustración 110 Inspector estadísticas

Al ser un objeto encriptable, no se le asigna al protagonista de la misma forma que le he estado asignando los scripts de las mecánicas anteriores. En este caso añadí un atributo correspondiente al objeto encriptable de las estadísticas dentro del script principal del protagonista.

```

public class Prota : MonoBehaviour
{
    [SerializeField] private ProtaStatsSO stats;
}

```

Ilustración 111 Script protagonista estadísticas

Para terminar con las estadísticas, solo quedaba reflejar las estadísticas actuales en el menú correspondiente, actualizando la interfaz. Añadí un método para actualizar este menú en manager de la interfaz. Como los datos se reflejan en la interfaz en forma de números, volví a ayudarme de TextMesh Pro.

```

private void ActualizarPanelStats()
{
    if (panelStats.activeSelf == false)
    {
        return;
    }
    statNivelTMP.text = stats.Nivel.ToString();
    statExpActualTMP.text = stats.ExpActual.ToString();
    statExpParaSiguienteNivelTMP.text = stats.ExpParaSiguienteNivel.ToString();
    statDañoTMP.text = stats.Pupa.ToString();
    statDdefensaTMP.text = stats.Defensa.ToString();
    statVelocidadTMP.text = stats.Velocidad.ToString();
    statPorcentajeCriticoTMP.text = $"{stats.PorcentajeCritico}%";
    statPorcentajeBloqueoTMP.text = $"{stats.PorcentajeBloqueo}%";
}

```

Ilustración 112 Manager interfaz estadísticas

Ya tenemos las estadísticas listas, pero queda crear la forma en la que el jugador pueda ir mejorando cada una de estas estadísticas a lo largo del juego. Para ello, he creado un sistema de atributos: cada vez que el protagonista sube de nivel, consigue 3 puntos de atributo. Los puntos de atributo puede usarlos para aumentar la fuerza, la destreza o la inteligencia. Dependiendo del atributo que decida aumentar, aumentará ciertos valores de determinadas estadísticas.

Aumentar un punto de fuerza equivale a subir dos puntos el daño, un punto la defensa y 0.2 puntos la probabilidad de bloqueo. Aumentar un punto de destreza equivale a aumentar un punto el daño, otro punto la velocidad, 0.2 puntos la probabilidad de bloqueo y 0.3 puntos la probabilidad de daño crítico. Por último, aumentar un punto de inteligencia equivale a subir dos puntos de defensa, 0.3 puntos de probabilidad de bloqueo y 0.2 puntos de probabilidad de daño crítico.

Los métodos que controlan esto están escritos en el mismo script de estadísticas.

```

public void AddBonusFuerza()
{
    Pupa += 2f;
    Defensa += 1;
    PorcentajeBloqueo += 0.2f;
}

1 referencia
public void AddBonusDestreza()
{
    Pupa += 1f;
    PorcentajeBloqueo += 0.2f;
    PorcentajeCritico += 0.3f;
    Velocidad += 1f;
}

1 referencia
public void AddBonusInteligencia()
{
    Defensa += 2;
    PorcentajeBloqueo += 0.3f;
    PorcentajeCritico += 0.2f;
}

```

Ilustración 113 Script estadísticas añadir atributos

Como los atributos se suben clicando en los botones del panel de estadísticas, creé un script para poder asociarlo a estos botones y que realicen esta función. Para ello, enumeré los 3 atributos distintos y creé un método que lanzara un evento (más tarde, configuraría los botones para que al hacer click en ellos se lanzara este evento).

```

public enum TipoAtributo
{
    Fuerza,
    Destreza,
    Inteligencia
}

♂ Script de Unity (3 referencias de recurso) | 2 referencias
public class BotonAtributos : MonoBehaviour
{
    public static Action<TipoAtributo> EventoAddAtributo;
    [SerializeField] private TipoAtributo tipo;

    0 referencias
    public void AddAtributo()
    {
        EventoAddAtributo?.Invoke(tipo);
    }
}

```

Ilustración 114 Script evento botón atributo

Desde el script del protagonista, creé los métodos para suscribirse y desuscribirse al evento creado anteriormente y responder a él.

```

private void OnEnable()
{
    BotonAtributos.EventoAddAtributo += RespuestaAtributo;
}

⊕ Mensaje de Unity | 0 referencias
private void OnDisable()
{
    BotonAtributos.EventoAddAtributo -= RespuestaAtributo;
}

```

Ilustración 115 Script suscripción evento botón atributo

La respuesta a este evento está desarrollada en un método aparte. Como anteriormente creé una enumeración para los 3 tipos de atributos distintos, en la respuesta utilizo un switch con esta enumeración. Dependiendo del tipo de atributo seleccionado, llamará al método que aumenta las estadísticas correspondientes a ese atributo. Tras aumentar las estadísticas, se resta un punto a los puntos de atributos disponibles.

```

private void RespuestaAtributo(TipoAtributo tipo)
{
    if (stats.PuntosDisponibles <= 0)
    {
        return;
    }
    switch (tipo)
    {
        case TipoAtributo.Fuerza:
            stats.Fuerza++;
            stats.AddBonusFuerza();
            break;
        case TipoAtributo.Destreza:
            stats.Destreza++;
            stats.AddBonusDestreza();
            break;
        case TipoAtributo.Inteligencia:
            stats.Inteligencia++;
            stats.AddBonusInteligencia();
            break;
    }
    stats.PuntosDisponibles -= 1;
}

```

Ilustración 116 Respuesta evento atributos

Al crear el objeto del menú de estadísticas y atributos en el canvas, seleccioné el tipo de objeto botón en los cuadraditos al lado de cada atributo, para poder hacer que, al clicarlos, respondan a un evento. Para ello, le añadí el script creado para los botones, seleccionando el tipo de entre los que creé en la enumeración, y luego seleccioné el método deseado dentro del campo “On Click”. De esta forma ya funcionan los botones.

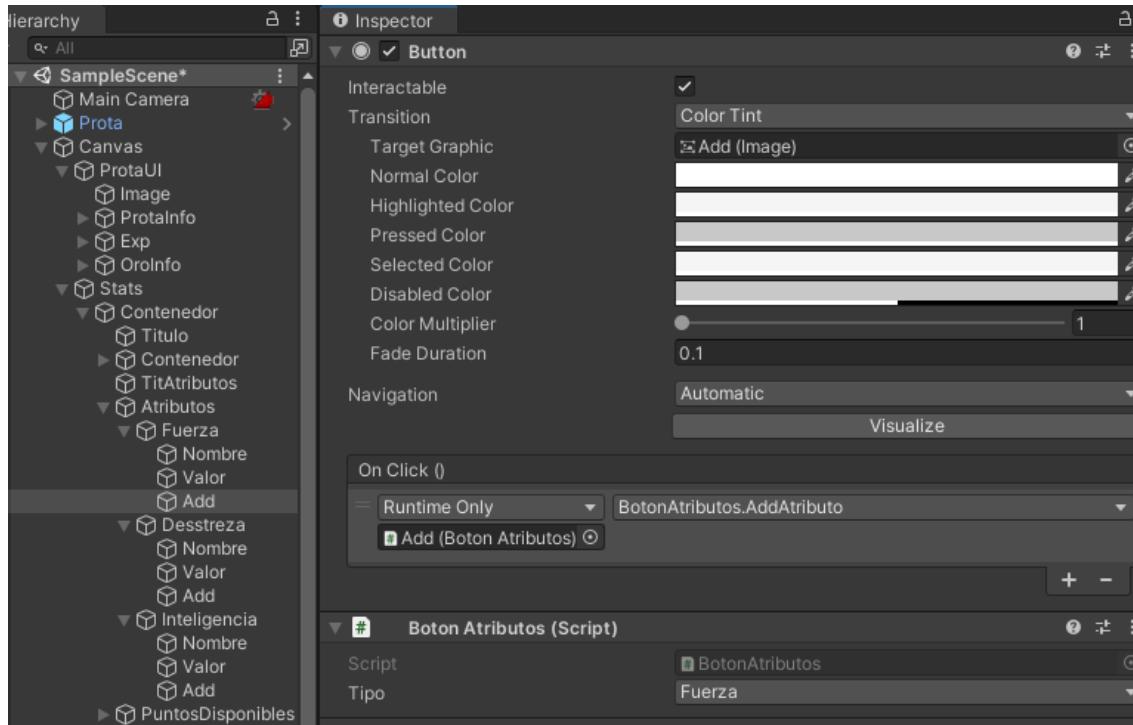


Ilustración 117 Inspector botones atributos

Para poder ver reflejados en la interfaz los cambios que haga el jugador en los atributos, aumenté el método creado en el manager de la interfaz para las estadísticas, volviendo a hacer uso de TextMesh Pro.

```
private void ActualizarPanelStats()
{
    if (panelStats.activeSelf == false)
    {
        return;
    }
    statNivelTMP.text = stats.Nivel.ToString();
    statExpActualTMP.text = stats.ExpActual.ToString();
    statExpParaSiguienteNivelTMP.text = stats.ExpParaSiguienteNivel.ToString();
    statDañoTMP.text = stats.Pupa.ToString();
    statDdefensaTMP.text = stats.Defensa.ToString();
    statVelocidadTMP.text = stats.Velocidad.ToString();
    statPorcentajeCriticoTMP.text = $"{stats.PorcentajeCritico}%";
    statPorcentajeBloqueoTMP.text = $"{stats.PorcentajeBloqueo}%";
    fuerzaTMP.text = stats.Fuerza.ToString();
    destrezaTMP.text = stats.Destreza.ToString();
    inteligenciaTMP.text = stats.Inteligencia.ToString();
    puntosDisponiblesTMP.text = stats.PuntosDisponibles.ToString();
}
```

Ilustración 118 Manager interfaz estadísticas y atributos

También edité el método de restaurar las estadísticas en el script de estadísticas, añadiendo valores para los atributos y los puntos disponibles.

```

public void ResetearStats()
{
    Nivel = 1;
    ExpActual = 0f;
    ExpParaSiguienteNivel = 0f;
    Pupa = 4f;
    Defensa = 2f;
    Velocidad = 4f;
    PorcentajeCritico = 0f;
    PorcentajeBloqueo = 0f;

    Fuerza = 0;
    Destreza = 0;
    Inteligencia = 0;
    PuntosDisponibles = 0;
}

```

Ilustración 119 Script restaurar estadísticas y atributos

Podemos comprobar desde el inspector que el objeto encriptable de estadísticas ahora contiene también los campos de atributos.

Cuando el juego esté en marcha, desde el objeto encriptable podemos ver como varían los valores a tiempo real al ser aumentados. Al usar el botón de restaurar, volverán al valor inicial.

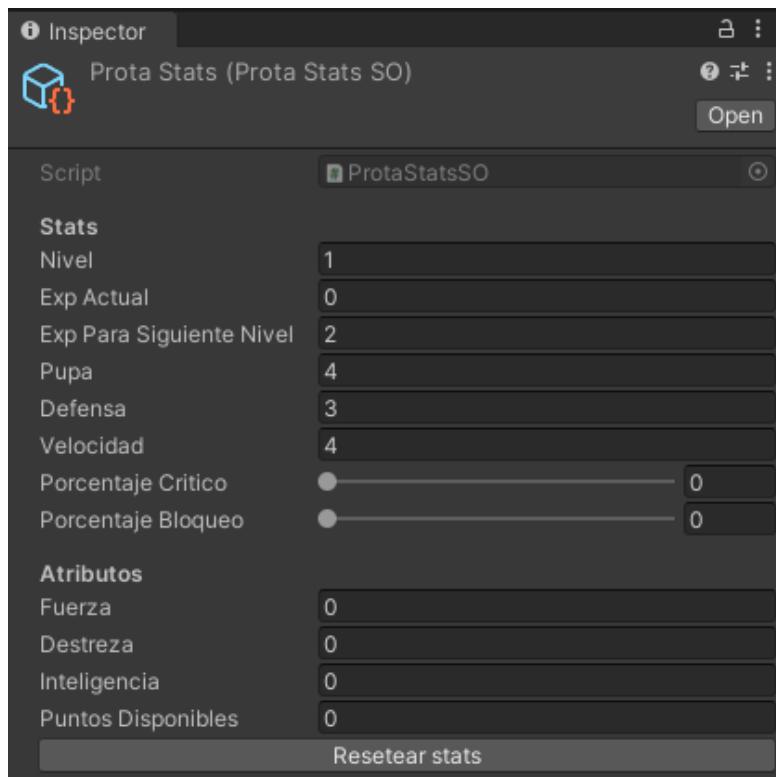


Ilustración 120 Inspector estadísticas y atributos

Mapa

Ahora que nuestro protagonista ya cuenta con algunos sistemas típicos de los videojuegos RPG, es hora de empezar con el mapa donde vivirá su aventura.

Primero creé un nuevo objeto de tipo grid, de rejilla cuadrada, donde irían las distintas capas en las que se “pintarían” cada parte del mapa. Estas capas, también basadas en rejillas cuadradas, están organizadas en base a la profundidad: la capa más al fondo es donde pintaría el suelo, la siguiente donde pintaría los límites, luego la capa en la que pintaría los detalles sobre el suelo y tras esta la capa donde irían los objetos que se verían por encima del protagonista. Además, creé otras dos capas por encima del protagonista para los árboles y las casas.

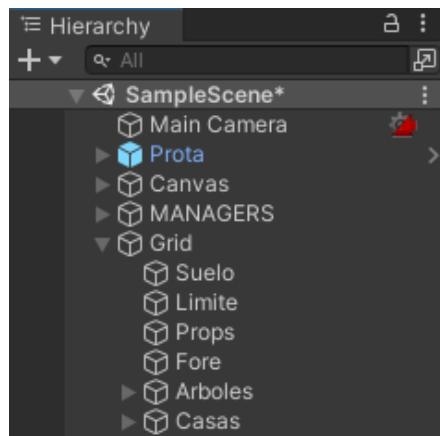


Ilustración 121 Jerarquía capas del mapa

Además de las capas creadas en el grid donde va el mapa, también podemos crear capas que definirán como actúan entre si los elementos de la escena. Algunas de estas capas ya vienen por defecto, pero podemos editarlos y añadir capas nuevas.

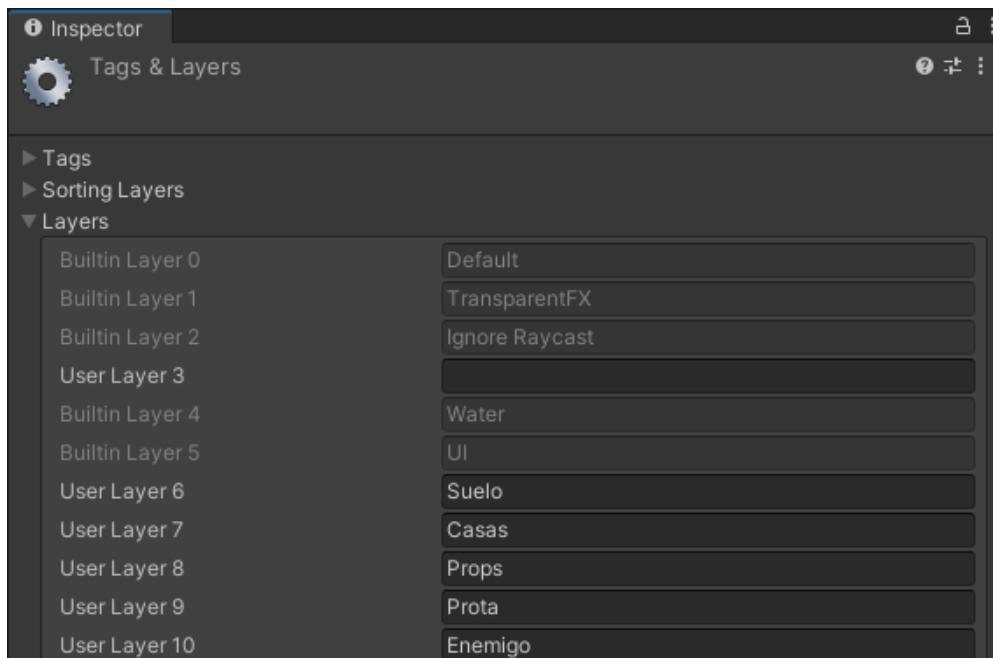


Ilustración 122 Inspector capas

Me aseguré de que cada objeto del grid estuviera en su capa correspondiente. Podemos comprobarlo y editarlos en la parte superior derecha del inspector.

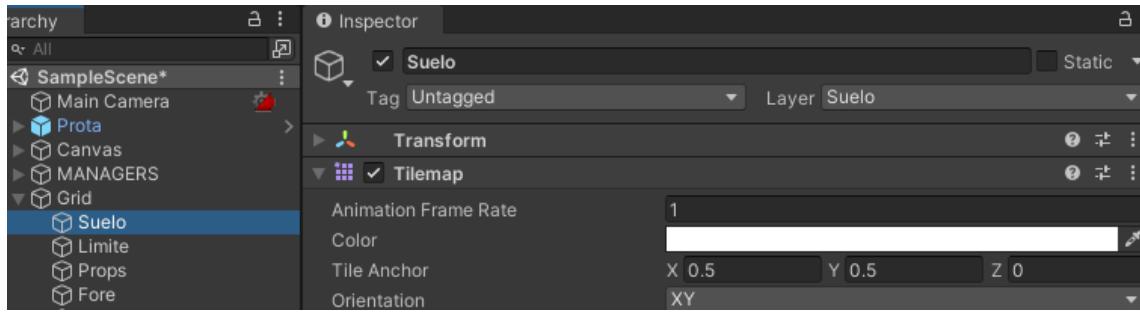


Ilustración 123 Inspector grid

El siguiente paso, previo a crear el mapa, es crear las paletas de tiles. Es una forma de ordenar los “pinceles” con los que vamos a pintar el mapa. Creé una paleta para los elementos del exterior y otra para los elementos del interior. Para ello, le di al botón de crear una nueva paleta desde la ventana Tile Palette y luego seleccioné el spritesheet deseado.

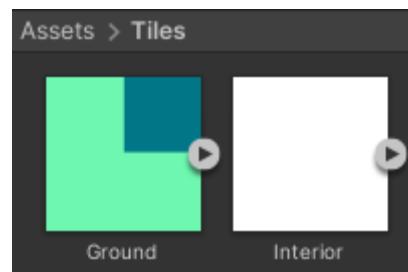


Ilustración 124 Paletas

Desde la ventana Tile Palette podemos elegir un cuadrado o más de la paleta seleccionada y pintar con ellos en la escena. También os ofrece herramientas para borrar, llenar o mover estos cuadrados de la escena.

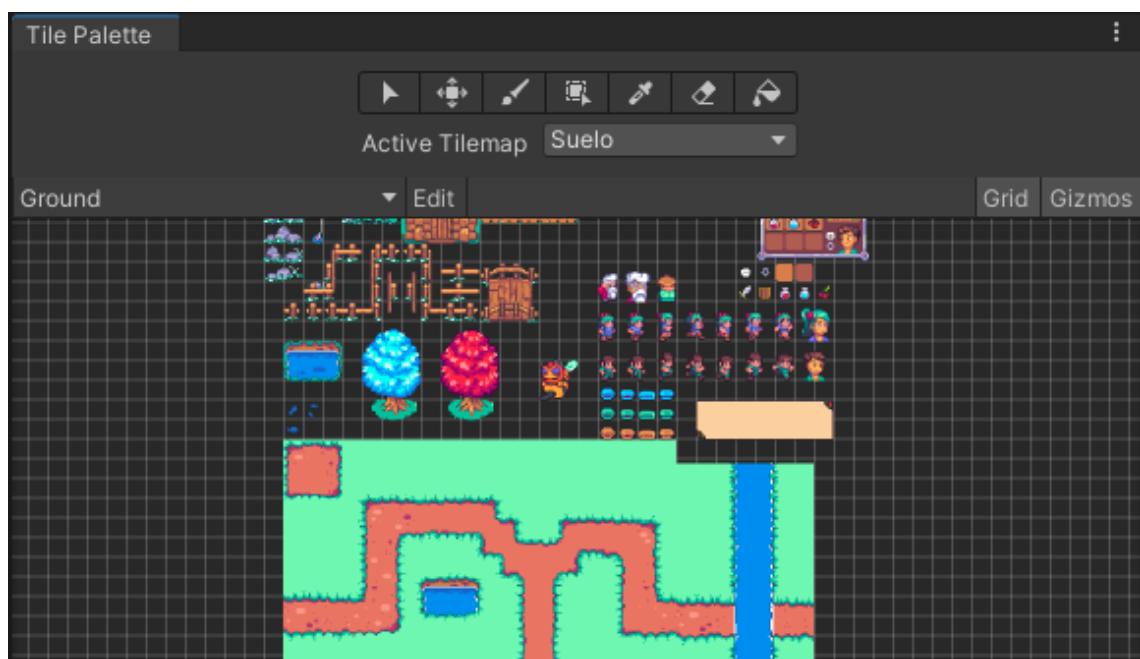


Ilustración 125 Ventana Tile Palette

Ya tenía todo listo para crear el mapa. Estuve haciendo varios bocetos previamente, aunque lo que más me importaba era que fuera funcional: cada aldea debía tener una tienda, distintos NPCs y sitios para craftear y aceptar misiones. Todas estas mecánicas las programaré más adelante.

Para crear las aldeas, primero empecé pintando el suelo: el césped, los caminos, el agua y los puentes. Luego, en una capa superior, pinté algunos detalles como flores y hierbas. En otra capa más adelante pinté carteles y vallas.

En las capas superiores de árboles y casas, no usé la paleta para pintar directamente en la escena: los pinté aparte y los convertí en prefab. De esta forma, era más fácil duplicarlos y moverlos por el mapa a mi antojo. Además, en los prefab incluí los collider para que el protagonista se chocara con ellos.

En cuanto a los collider del resto del mapa, los dibujé en la capa creada para el límite. Tras dibujar en las zonas en las que quería que el protagonista se chocara: vallas, agua, carteles, etc; añadí collider a esta capa y luego la puse en invisible, para que hiciera de límite, pero no estropeara el mapa que había pintado ya debajo.



Ilustración 126 Mapa primera aldea

Los mapas también contienen bosques, que estarán a las afueras de las aldeas y las comunicarán entre ellas. En estos bosques no hay casas ni NPCs, están más desiertos y solo contienen enemigos.



Ilustración 127 Mapa primer bosque

Ya podemos añadir el prefab de nuestro protagonista al mapa creado y hacer que se mueva por él. Sin embargo, hace falta configurar la cámara para que persiga al protagonista por el mapa de la forma deseada. Hice uso de Cinemachine, una utilidad que ya incluyen las últimas versiones de Unity. Simplemente hay que añadir esta cámara a la jerarquía y elegir el elemento al que queremos que siga (en mi caso al protagonista) y el zoom.

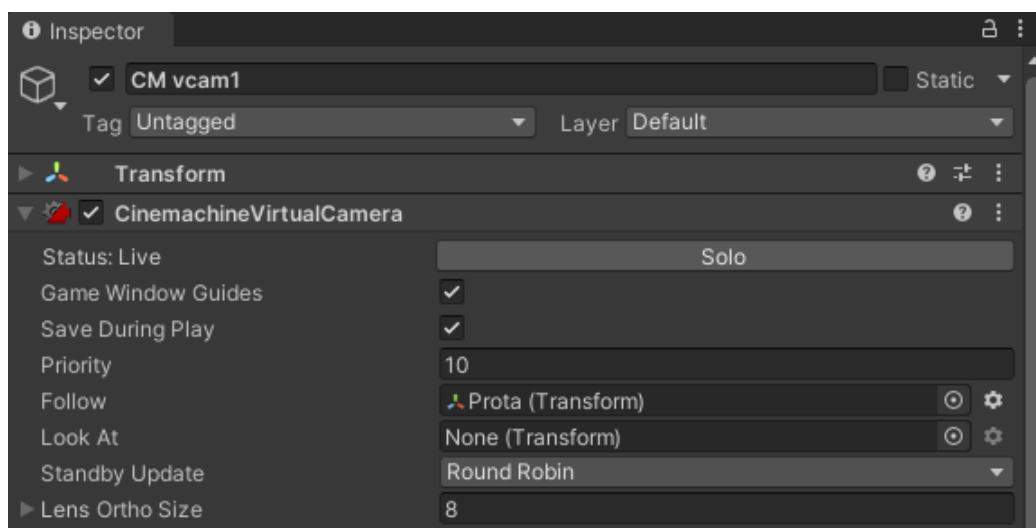


Ilustración 128 Inspector cámara

En lugar de añadir la cámara directamente a la jerarquía, añadí una cámara a cada una de las partes del mapa (configuradas de la misma forma) para realizar transiciones entre ellas cada vez que el protagonista se mueva a una zona distinta.

Para hacer las transiciones, añadí a cada parte del mapa un collider del tipo trigger, para poder comprobar si el protagonista sale o entra en cada uno de ellos. Creé un script para controlar que cuando el protagonista sale de un collider y entra en otro, se desactiva la cámara del primer sitio y se activa la cámara del segundo.

```
[SerializeField] private CinemachineVirtualCamera camara;

private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Player"))
    {
        camara.gameObject.SetActive(true);
    }
}

private void OnTriggerExit2D(Collider2D other)
{
    if (other.CompareTag("Player"))
    {
        camara.gameObject.SetActive(false);
    }
}
```

Ilustración 129 Script transición de cámaras

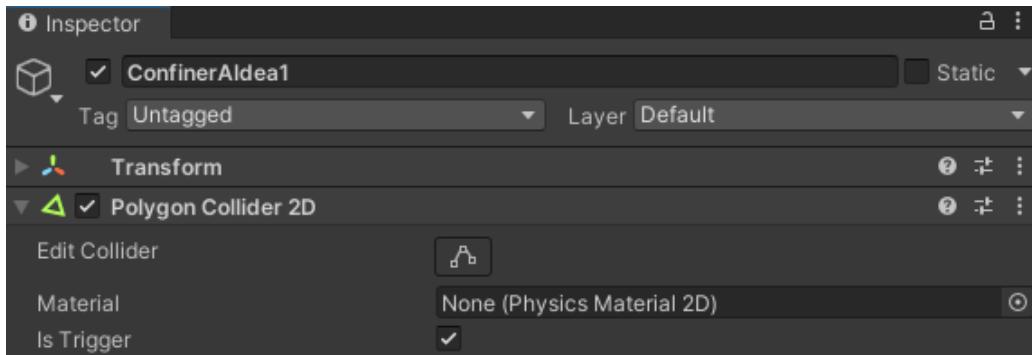


Ilustración 130 Inspector collider aldea

Estos límites establecidos en las zonas mediante collider, además de para realizar las transiciones de cámaras, también sirven para que la cámara no salga de este límite y se vea fuera del mapa.

A continuación, podemos ver la aldea con el límite pintado con líneas verdes y la cámara pintada con líneas rojas.



Ilustración 131 Aldea límite y cámara

También modifiqué algunos aspectos de la cámara para que siguiera al personaje a mi gusto: establecí el espacio en el que el personaje podía moverse libremente sin que se moviera la cámara, el espacio en el que la cámara seguiría al personaje al mismo ritmo al que este se moviera y el espacio en el que se redirigiría la cámara al centro del personaje. No solo edité estos espacios, sino también la velocidad con la que se movería la cámara en cada uno de ellos. Estos parámetros se editan desde el inspector de Cinemachine.

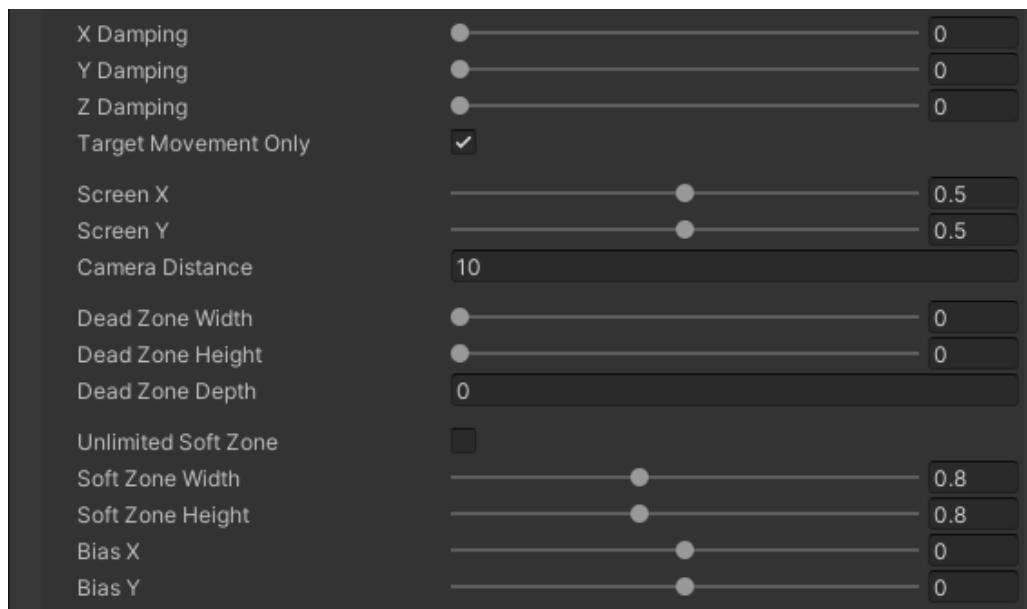


Ilustración 132 Parámetros inspector Cinemachine



Ilustración 133 Escena parámetros Cinemachine

Sprint 2

Comencé el segundo Sprint con un punto que quedó pendiente sobre el mapa, aunque ahora mismo no es muy importante: la zona de respawn. De momento, no tenía enemigos que puedan atacar y, por tanto, matar al protagonista, pero lo dejé preparado para un futuro.

Para controlar el punto de respawn, creé un nuevo manager, que guardé junto al manager de interfaz. Este manager tiene asociado un objeto y unas coordenadas. Cuando el protagonista muere, mueve el objeto al punto de coordenadas seleccionado.

```
private void Update()
{
    if (_prota.ProtaVida.Morido)
    {
        _prota.transform.localPosition = _respawn.position;
        _prota.RevivirPersonaje();
    }
}
```

Ilustración 134 Script manager de nivel

Al añadir este script al objeto de la jerarquía, referencie al protagonista como objeto y el punto en que deseaba que hiciera el respawn en las coordenadas. De esta forma, al revivir el personaje aparecería en este punto de coordenadas.

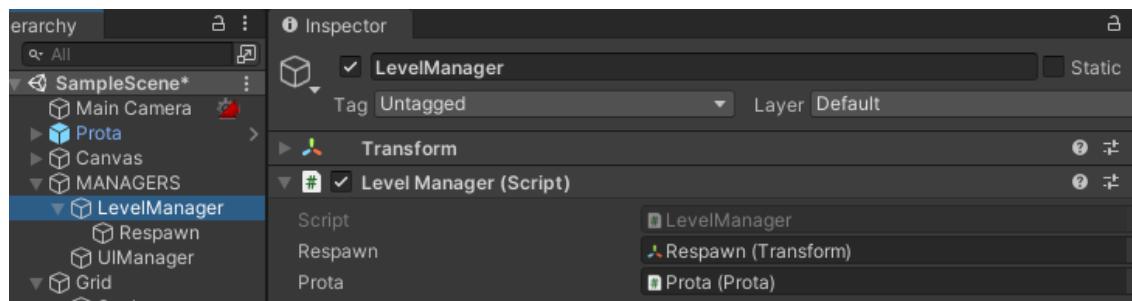


Ilustración 135 Inspector manager de nivel

Para poder ver en la escena el punto de respawn seleccionado, le asigné un ícono de un círculo verde a este punto de coordenadas.

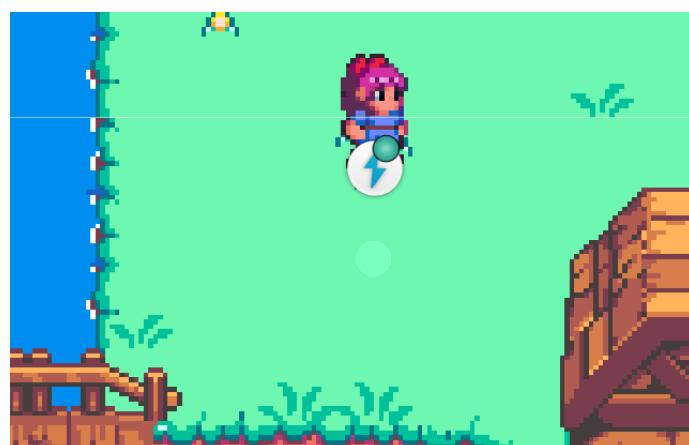


Ilustración 136 Escena punto de respawn

Objetos

Un elemento importante en cualquier videojuego de rol son los objetos. Creé una clase base que permitiría usar estos objetos y de la que heredarían más tarde los objetos más específicos como las armas o las pociones. La clase base de objetos nos permitirá crear objetos encriptables para ayudarnos a guardar los objetos que creemos y más adelante arrastrarlos a la escena. Esta clase contiene los parámetros como el id de los objetos, el nombre, la imagen o la descripción, junto a un enumerado de los tipos de objetos que se pueden crear. Además, contiene también otra información relativa a los objetos como si son consumibles, si se pueden acumular en el inventario y cuántos podemos acumular en el inventario, en el caso de que se acumulable. También contiene los métodos para usarlos, equiparlos, desequiparlos o acceder a su descripción.

```
public virtual bool UsaItem()
{
    return true;
}

2 referencias
public virtual bool EquiparItem()
{
    return true;
}

2 referencias
public virtual bool BorraItem()
{
    return true;
}

3 referencias
public virtual string DescripcionItemCraft()
{
    return "";
}
```

Ilustración 137 Script objetos métodos de uso

Añadí una cabecera que me permitía crear los objetos encriptables a partir de este script desde el menú de assets de Unity.

```
[CreateAssetMenu(menuName = "Items/Item")]
```

Ilustración 138 Script crear objeto encriptable objeto

Para comprobar que todo esto funciona, primero creé dos objetos básicos: una poción de vida y otra de maná. El primer paso fue crear un script específico para la poción de vida, que heredara del script general de objetos, y que además contuviera variables específicas de este tipo de objeto: cuántos puntos de vida restaura. También le asigné una cabecera para crear este objeto específico desde el menú de assets. Sobrescribí los métodos generales de uso y de descripción.

```

public override bool UsaItem()
{
    if (Inventario.Instance.Prota.ProtaVida.PuedeSerCurado)
    {
        Inventario.Instance.Prota.ProtaVida.RestaurarSalud(PVRestaurar);
        return true;
    }
    return false;
}

2 referencias
public override string DescripcionItemCraft()
{
    string descripcion = $"Restaura {PVRestaurar} de salud";
    return descripcion;
}

```

Ilustración 139 Script objeto poción vida

Repetí las mismas acciones en un script específico para las pociones de maná, indicando que restaura maná en lugar de vida.

```

public override bool UsaItem()
{
    if (Inventario.Instance.Prota.ProtaManá.PuedeRestaurarse)
    {
        Inventario.Instance.Prota.ProtaManá.RestauraManá(PMRestaurar);
        return true;
    }
    return false;
}

```

Ilustración 140 Script objeto poción maná

Desde el menú de assets, en la carpeta correspondiente a estos objetos dentro de objetos encriptables, creé los objetos encriptables de la poción de vida y de la poción de maná.



Ilustración 141 Carpeta para pociones

A ambos les asigné el tipo de objeto poción, indiqué que fueran consumibles, acumulables, y que se pudieran acumular hasta 60 en el mismo espacio del inventario. También indiqué que ambas pociones restauraban 20 puntos de su correspondiente característica. Por último, a cada una le asigné un id, un nombre, una imagen y una descripción.

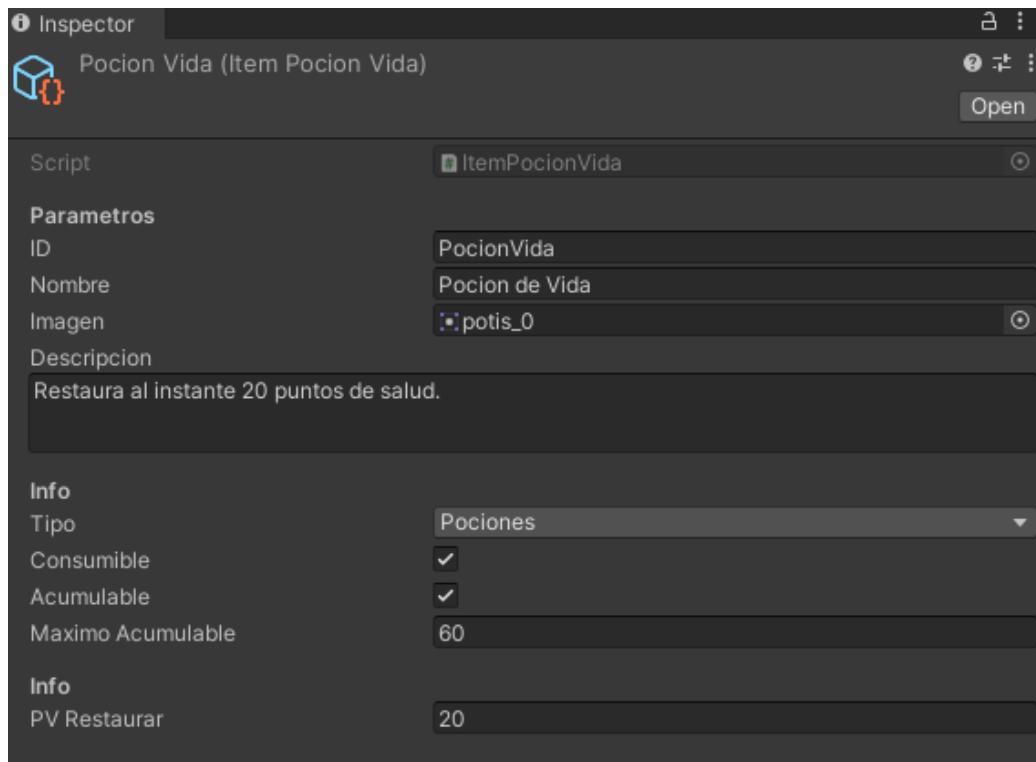


Ilustración 142 Inspector objeto encriptable poción vida

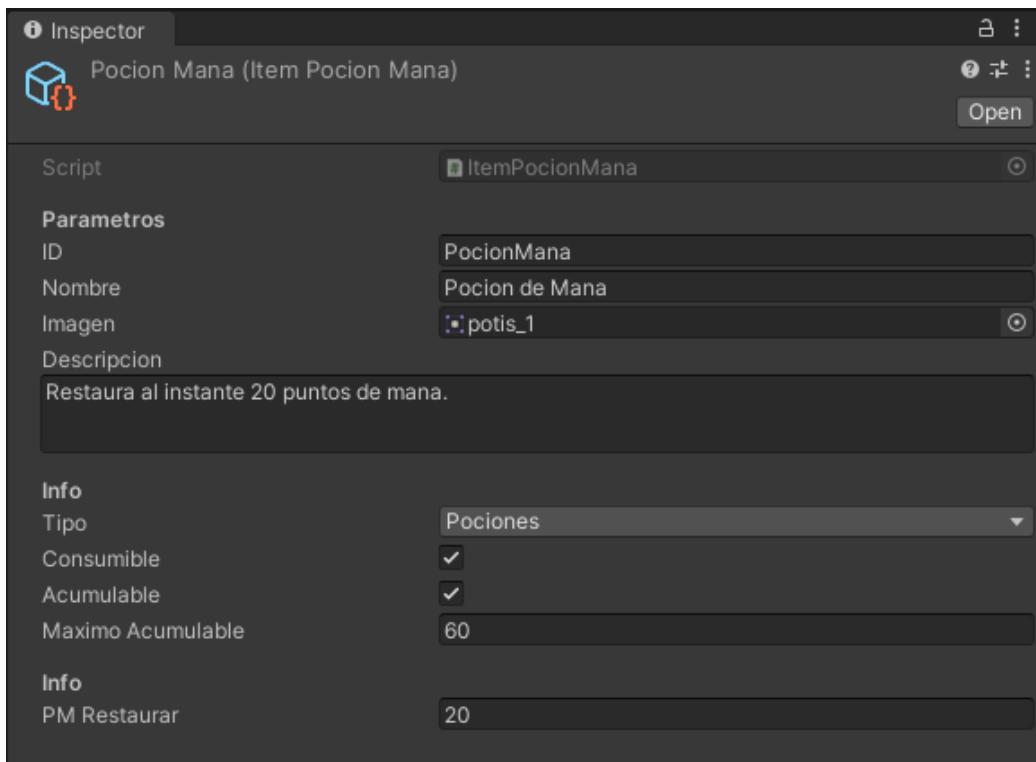


Ilustración 143 Inspector objeto encriptable poción maná

Tras comprobar que estos objetos estaban bien creados y funcionaban correctamente, creé algunos objetos más, siguiendo los mismos pasos. Estos otros objetos incluyen pociones de maná o de vida que restauran más puntos y serán de mayor utilidad en la parte final de la partida, y también ingredientes que se podrán conseguir a lo largo de la historia y

permitirán al personaje craftear nuevos objetos más útiles. Añado el resultado de solo algunos de estos objetos, puesto a que todos se crean de forma muy similar.

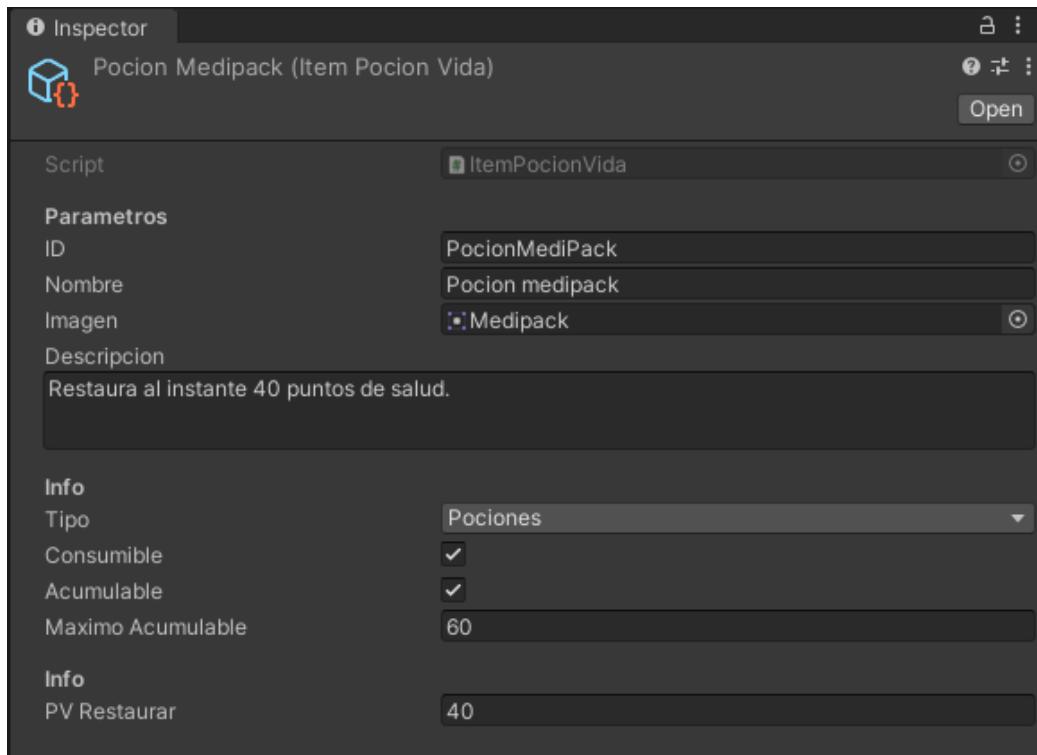


Ilustración 144 Inspector objeto encriptable superpoción vida

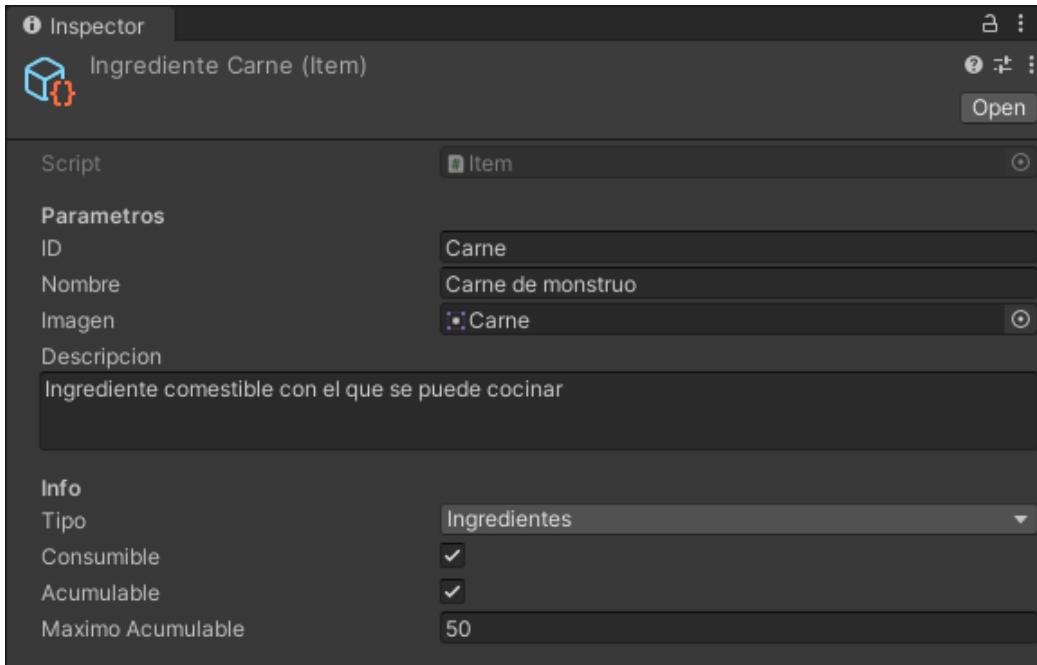


Ilustración 145 Inspector objeto encriptable ingrediente carne

Sistema de inventario

Con los objetos creados, necesitamos un sistema para poder almacenarlos y desde el que poder acceder a ellos y usarlos. Por este motivo, creé el sistema de inventario. Para ello, el primer paso fue crear la interfaz del inventario desde el canvas.

La interfaz del inventario cuenta con un panel principal donde hay 24 huecos o slots para almacenar los objetos, posicionados con un grupo de grid dentro del panel. Cada uno de estos huecos muestran el objeto almacenado (si contiene uno) y la cantidad del mismo. Debajo del panel principal, hay 3 botones para usar, equipar y desechar el objeto seleccionado.

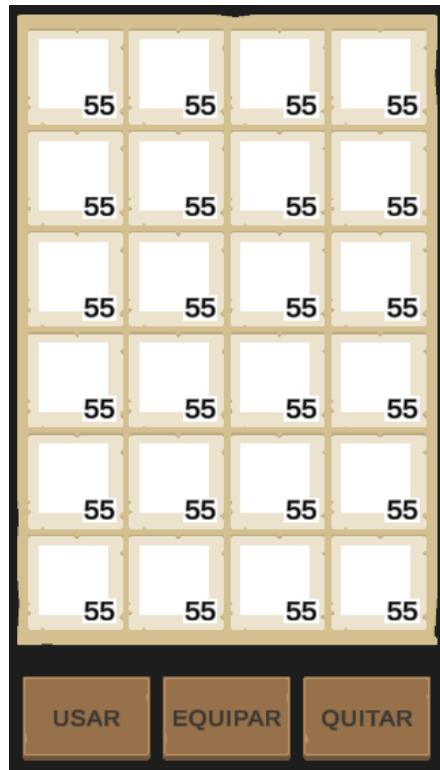


Ilustración 146 Interfaz inventario

Cuando se selecciona un objeto, el fondo del slot cambia de color y se abre un panel a la derecha con información del objeto seleccionado: imagen, nombre y descripción. Esto se controla con el componente botón del slot.

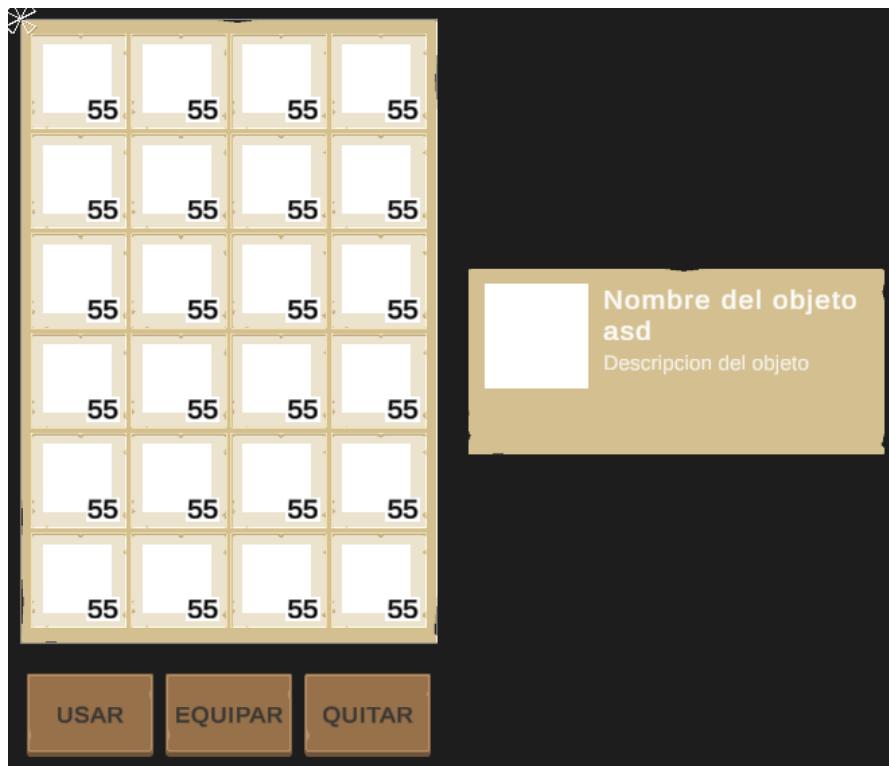


Ilustración 147 Interfaz inventario abierto

Para instanciar los slots en el inventario, creé un script donde hacía referencia al propio objeto slot y al panel donde estarían contenido, y también a cada uno de los campos de los objetos que pueden contener: nombre, cantidad, etc.

Creé un script para los slots del inventario, para poder referenciar en cada uno de ello la imagen y la cantidad del objeto que contiene. Si ese slot no contiene ningún objeto, se desactiva la imagen y el número de la cantidad, para que se vea vacío.

```
public void ActualizaSlot(InventoryItem item, int cantidad)
{
    icono.sprite = item.Imagen;
    cantidadTMP.text = cantidad.ToString();
}
```

Ilustración 148 Script slot actualizar imagen y cantidad

Mas tarde creé un script general para el inventario, con referencia al protagonista, y con una lista de objetos que se irá llenando conforme los añadamos. La longitud de esta lista se define por una variable que indica el número de slots, editable desde el inspector. En mi caso, fijé esta cantidad en 24. Además, este script cuenta con métodos para añadir los objetos al inventario. Para añadirlos, comprueba si el objeto ya existe en el inventario, si es acumulable y la cantidad de ese objeto que teníamos previamente en el inventario. Dependiendo de estos factores, añade o no el objeto al inventario, y lo hace en un nuevo slot o en uno que ya contenía objetos de ese tipo.

```

public void AddItem(InventoryItem item, int cantidad)
{
    if (item == null)
    {
        return;
    }

    List<int> index = Existencias(item.ID);
    if (item.Acumulable)
    {
        if (index.Count > 0)
        {
            for (int i = 0; i < index.Count; i++)
            {
                if (itemsInventory[index[i]].Cantidad < item.MaximoAcumulable)
                {
                    itemsInventory[index[i]].Cantidad += cantidad;
                    if (itemsInventory[index[i]].Cantidad > item.MaximoAcumulable)
                    {
                        int diferencia = itemsInventory[index[i]].Cantidad - item.MaximoAcumulable;
                        itemsInventory[index[i]].Cantidad = item.MaximoAcumulable;
                        AddItem(item, diferencia);
                    }
                    return;
                }
            }
        }
    }
}

```

Ilustración 149 Script inventario añadir objeto

Podemos llamar a este método de añadir desde la tienda o desde el menú de crafteo o de loot, pero también podemos hacerlo cuando el protagonista colisione con objetos que están situados en la escena. En todos estos casos, hay que quitar el objeto del mundo o del manú correspondiente tras añadirlo al inventario del protagonista.

Para poder interactuar con los slots y los objetos que contienen dentro del inventario, creé un evento que se lanzará al clicar en cada uno de los slots y al clicar en los botones de usar, equipar y desequipar teniendo un slot seleccionado.

```

public void ClickSlot()
{
    EventoSlotInteraccion?.Invoke(TiposInteraccion.Click, Index);
}

```

Ilustración 150 Script slot lanzar evento clicar

```

public void SlotUsaItem()
{
    if (Inventario.Instance.ItemsInventario[Index] != null)
    {
        EventoSlotInteraccion?.Invoke(TiposInteraccion.Usar, Index);
    }
}

1 referencia
public void SlotEquiparItem()
{
    if (Inventario.Instance.ItemsInventario[Index] != null)
    {
        EventoSlotInteraccion?.Invoke(TiposInteraccion.Equipar, Index);
    }
}

1 referencia
public void SlotQuitarItem()
{
    if (Inventario.Instance.ItemsInventario[Index] != null)
    {
        EventoSlotInteraccion?.Invoke(TiposInteraccion.Borrar, Index);
    }
}

```

Ilustración 151 Script slot lanzar evento usar, equipar y quitar

Dentro del script creado para el inventario, creé los métodos para suscribirse y desuscribirse del evento anterior.

```

private void OnEnable()
{
    InventarioSlot.EventoSlotInteraccion += RespuestaSlotInteraccion;
}

⊗ Mensaje de Unity | 0 referencias
private void OnDisable()
{
    InventarioSlot.EventoSlotInteraccion -= RespuestaSlotInteraccion;
}

```

Ilustración 152 Script inventario escuchar evento interactuar

La respuesta a este evento será distinta dependiendo del botón que seleccionemos una vez el slot esté seleccionado. Dependiendo de si usamos el botón usar, equipar o borrar, se ejecutará el método correspondiente. Esto se controla con un switch con los distintos tipo de interacción.

```

private void RespuestaSlotInteraccion(TiposInteraccion tipo, int index)
{
    switch (tipo)
    {
        case TiposInteraccion.Uso:
            UsaItem(index);
            break;
        case TiposInteraccion.Equipar:
            EquiparItem(index);
            break;
        case TiposInteraccion.Borrar:
            QuitarItem(index);
            break;
    }
}

```

Ilustración 153 Script inventario respuesta evento interactuar

Desde el switch llamo a cada uno de los métodos independientes de respuesta. El método de usar objeto, además de darnos los atributos correspondientes a este objeto, quita una unidad del objeto de nuestro inventario. También se eliminan objetos del inventario al ser usados para craftear otros objetos.

Es necesario reflejar en la interfaz del inventario todo lo creado para que el jugador pueda interactuar con él. Comencé con el método para activar los slots dentro del inventario y que muestren la imagen y cantidad correspondiente de los objetos.

```

public void ActivaUISlot(bool estado)
{
    icono.gameObject.SetActive(estado);
    fondoCantidad.SetActive(estado);
}

```

Ilustración 154 Script interfaz activar slot

A continuación, instancié la imagen, el nombre y la descripción de cada objeto que se mostraría en el panel de la derecha al clicar en un slot que contenga un objeto. También me ayudé en este caso de TextMesh Pro.

```

private void ActualizaDescripcion(int index)
{
    if (Inventario.Instance.ItemsInventario[index] != null)
    {
        icono.sprite = Inventario.Instance.ItemsInventario[index].Imagen;
        nombre.text = Inventario.Instance.ItemsInventario[index].Nombre;
        descripcion.text = Inventario.Instance.ItemsInventario[index].Descripcion;
        inventarioDescripcion.SetActive(true);
    }
    else
    {
        inventarioDescripcion.SetActive(false);
    }
}

```

Ilustración 155 Script interfaz descripción objetos

Desde el inspector del canvas, añadí a los botones de usar, equipar y quitar los eventos correspondientes en el campo On Click.

Por último, desde el manager de la interfaz creé los métodos necesarios para que el menú de inventario se abra y se cierre desde el menú de accesos creado anteriormente situado en la parte inferior izquierda de la pantalla; concretamente, desde el botón con el dibujo de un zurrón.

NPCs

Voy a añadir más personajes a mi juego, a parte del protagonista. Para ello, empezaré por crear las animaciones para cada uno de ellos con los spritesheets guardados en su carpeta correspondiente de personajes, dentro de la carpeta de sprites en assets. El proceso de animar ha sido el mismo para todos los NPCs, por lo que solo pondré el resultado de uno de ellos.



Ilustración 156 Spritesheet NPC

Como todos los sprites estaban previamente tratados y recortados, pude usarlos directamente en la ventana Animation. Creé dos animaciones nuevas, como las creadas para el protagonista: idle y correr. Para la primera, arrastré los cuatro sprites de la parte inferior del spritesheet y puse cada uno de ellos en el tiempo correspondiente. Para la segunda animación, repetí lo mismo con los sprites de la parte superior.

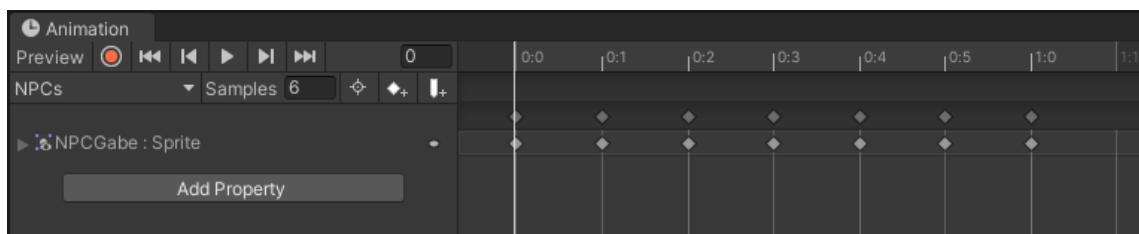


Ilustración 157 Animation NPC

Con el NPC arrastrado a la escena, le añadí la componente Animator y seleccioné dentro de él una de las animaciones creadas anteriormente para este NPC en concreto.

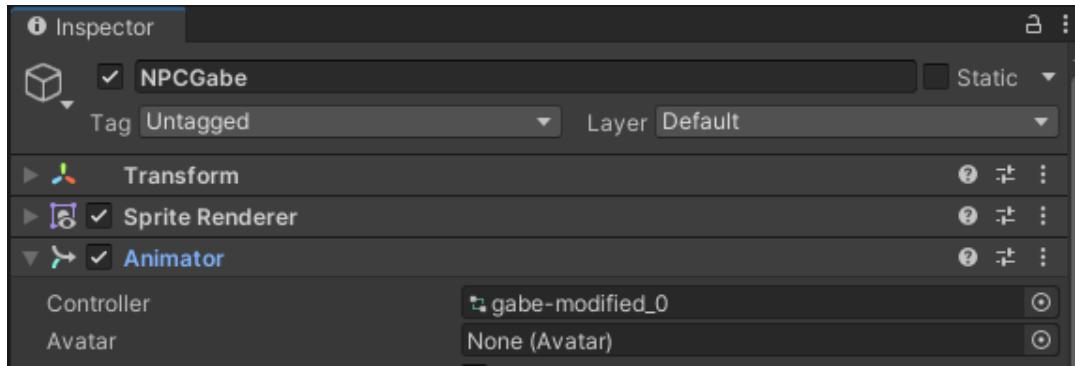


Ilustración 158 Inspector Animator NPC

Para los NPC no creé arboles de animaciones, ya que cada uno de ellos tendría asignado solo un tipo de animación: o estarían todo el tiempo en idle, o todo el tiempo corriendo. No era necesario crear transiciones.

Como quiero que algunos NPCs se estén moviendo por el mapa, creé un script para trazar caminos por los que andarán. Este script cuenta con una lista de vectores, de forma que, desde el inspector, al añadir el script a un NPC, podemos añadir puntos a la lista trazando vectores de movimiento entre los distintos puntos seleccionados y de forma ordenada. Para poder añadir los puntos de una forma más visual, me he ayudado de los gizmos para pintar estos puntos del recorrido sobre la escena.

```
private void OnDrawGizmos()
{
    if (juegoIniciado == false && transform.hasChanged)
    {
        PosicionActual = transform.position;
    }
    if (puntos == null || puntos.Length <= 0)
    {
        return;
    }
    for (int i = 0; i < puntos.Length; i++)
    {
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(puntos[i] + PosicionActual, radius:0.5f);
        if (i < puntos.Length - 1)
        {
            Gizmos.color = Color.gray;
            Gizmos.DrawLine(puntos[i] + PosicionActual, puntos[i + 1] + PosicionActual);
        }
    }
}
```

Ilustración 159 Script movimiento gizmos

Luego creé otro script para indicar la velocidad a la que el personaje se movería por este recorrido.

```

private void MoverPersonaje()
{
    transform.position = Vector3.MoveTowards(current: transform.position,
                                              target: PuntoSiguiente, maxDistanceDelta: velocidad * Time.deltaTime);
}

```

Ilustración 160 Script movimiento velocidad

Para crear la lógica con la que el personaje sigue el recorrido de puntos asignados, creé un método con acceso al recorrido y a la posición del personaje. Este método comprueba la distancia entre el personaje y el punto al que se dirige para comprobar si lo ha alcanzado. En el caso de que sí haya alcanzado este punto, el punto objetivo se cambia por el siguiente de la lista. Cuanto el personaje ha alcanzado el último punto de la lista, vuelve hasta el punto inicial. Así sigue el movimiento en forma de bucle.

```

private bool IndexAlcanzado()
{
    float distanciaPunto = (transform.position - PuntoSiguiente).magnitude;
    if (distanciaPunto < 0.1f)
    {
        _ultimaPosicion = transform.position;
        return true;
    }
    return false;
}

1 referencia
private void ActualizaIndex()
{
    if (indexPuntoActual == _waypoint.Puntos.Length - 1)
    {
        indexPuntoActual = 0;
    }
    else
    {
        if (indexPuntoActual < _waypoint.Puntos.Length - 1)
        {
            indexPuntoActual++;
        }
    }
}

```

Ilustración 161 Script movimiento actualizar punto objetivo

Además, en este script creé una enumeración para seleccionar si el movimiento es horizontal o vertical. En el caso de que el movimiento del personaje sea de forma horizontal, un método gira su sprite para que el personaje siempre mire hacia el frente, dependiendo del lado al que esté moviéndose en ese momento.

```

private void GirarPersonaje()
{
    if (direccion != Direccion.Horizontal)
    {
        return;
    }
    if (PuntoSiguiente.x > _ultimaPosicion.x)
    {
        transform.localScale = new Vector3(x: 1, y: 1, z: 1);
    }
    else
    {
        transform.localScale = new Vector3(x: -1, y: 1, z: 1);
    }
}

```

Ilustración 162 Script movimiento girar personaje

Podemos ver como desde el inspector, una vez añadido estos scripts de movimientos al NPC, podemos configurar si su movimiento es vertical u horizontal, su velocidad, y crear la lista de puntos por los que se moverá.

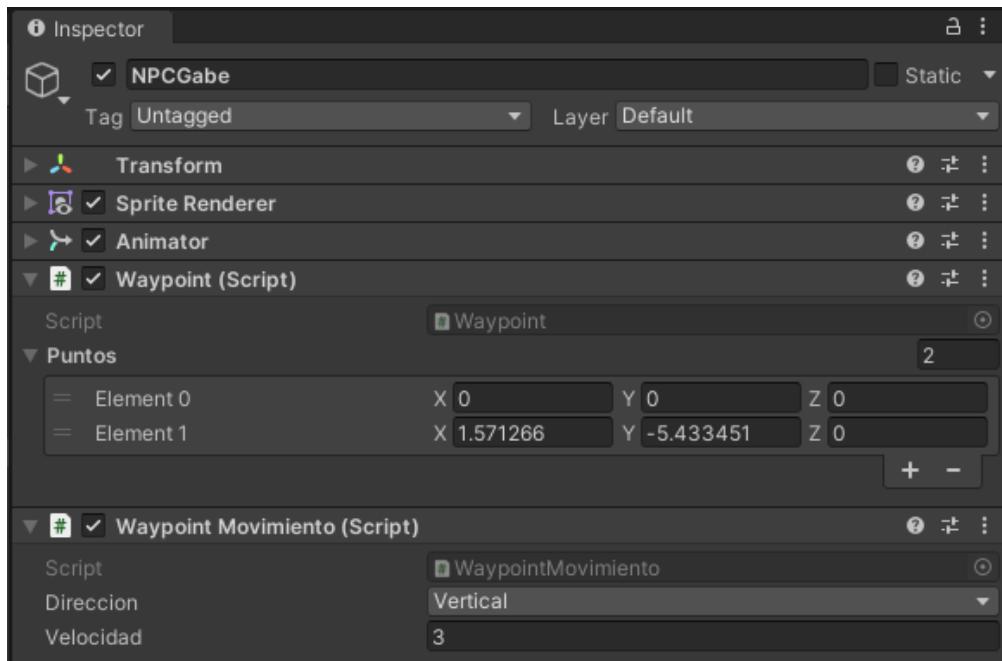


Ilustración 163 Inspector movimiento NPC

También podemos ver en la escena el recorrido creado y mover los puntos del recorrido de forma más visual.



Ilustración 164 Escena movimiento NPC con gizmos

Añadí mas NPC a la escena, concretamente a las zonas del mapa correspondientes a las aldeas. Algunos de ellos los añadí de forma estática y a otros les añadí un recorrido y sus animaciones correspondientes.



Ilustración 165 Escena creación de NPCs

Para tener estos elementos ordenados, creé un objeto vacío en la jerarquía donde añadí todos los NPCs.

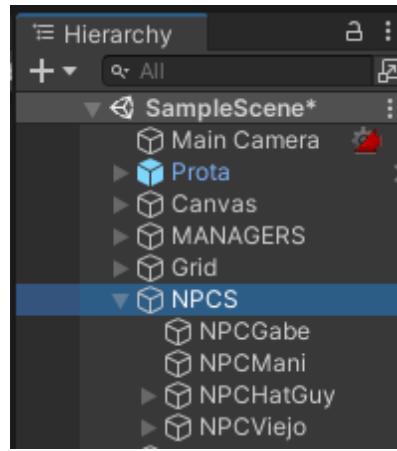


Ilustración 166 Jerarquía NPCs

A cada uno de los NPCs les añadí un collider de su tamaño para que interactuaran físicamente con el protagonista y otros elementos.

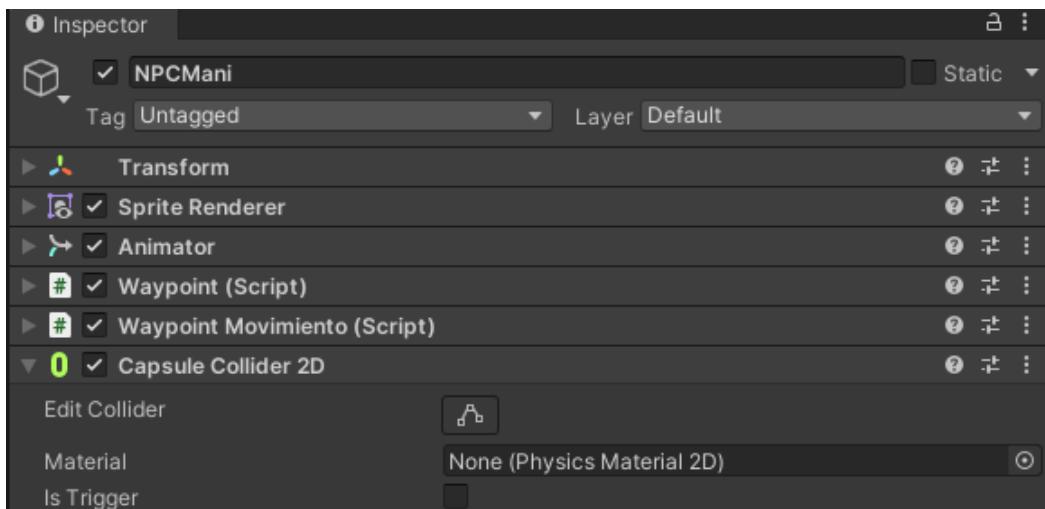


Ilustración 167 Inspector NPCs collider

Sistema de diálogos

Para poder tener más interacción con los NPC en el juego, creé un sistema de diálogo que permitiera a nuestro protagonista tener una conversación con ellos. Primero creé en el canvas el panel que contendría los diálogos. Este panel consta de una imagen del NPC con el que hablamos, su nombre, texto de la conversación y un botón de siguiente.



Ilustración 168 Interfaz diálogo

También creé un botón que surgiría de la cabeza de los NPCs con conversaciones disponibles al acercarnos a ellos, y desde el que se accede a la conversación. Creé dos versiones, una con una “?” para las conversaciones con misiones y otra con un “!” para el resto de las conversaciones. Es típico de los juegos RPG indicar con una “?” que un personaje tiene misiones.



Ilustración 169 Interfaz botón diálogo

Para poder interactuar con los NPC y que salga el botón de diálogo al acercarnos, les añadí otro collider, esta vez cuadrado y con un tamaño mayor, de tipo trigger.

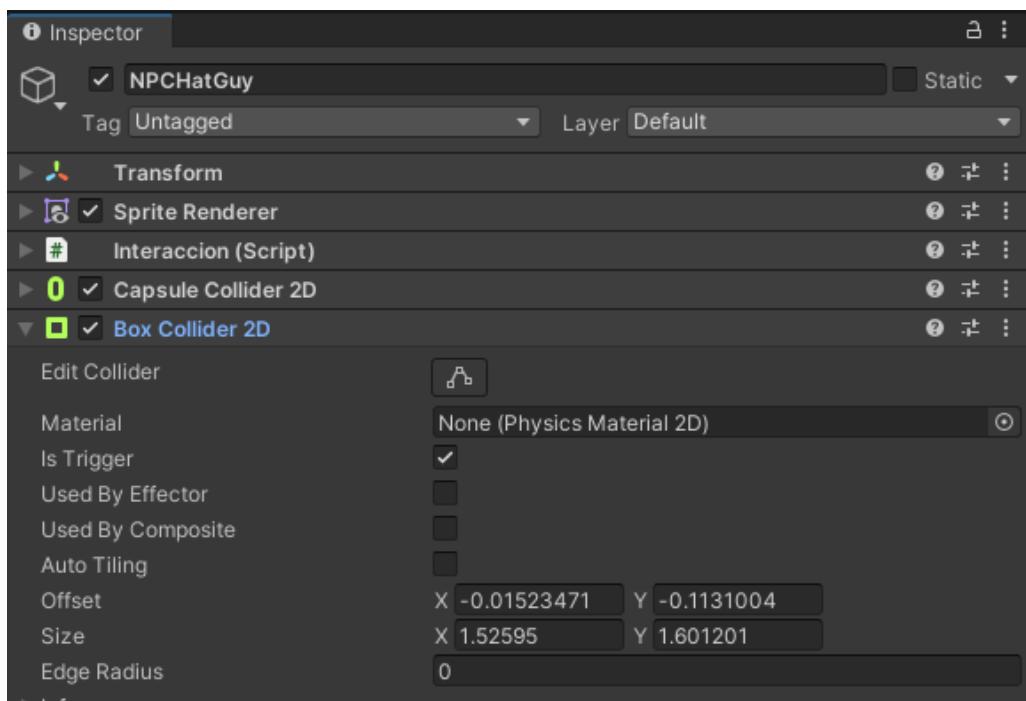


Ilustración 170 Inspector NPC collider trigger

Creé un nuevo script que usara este collider para comprobar si el protagonista ha entrado no a la zona que rodea a los personajes. Para detectar al protagonista, usé el tag “Player” asignado a su prefab.

```

private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Player"))
    {
        DialogoManager.Instance.NPCInteractable = this;
        botonInteraccion.SetActive(true);
    }
}

⊗ Mensaje de Unity | 0 referencias
private void OnTriggerExit2D(Collider2D other)
{
    if (other.CompareTag("Player"))
    {
        DialogoManager.Instance.NPCInteractable = null;
        botonInteraccion.SetActive(false);
    }
}

```

Ilustración 171 Script detectar protagonista diálogos

En este punto creé el script para crear los diálogos, en forma de objetos encriptables. Este script contiene las variables de la imagen del personaje asociado, su nombre y si tienen alguna interacción extra, junto al tipo de esta interacción. Para distinguir las interacciones extras, creé un enumerado con los tres tipos: tienda, misiones y craftear. Estas mecánicas las crearé más adelante. Además de estos parámetros, el objeto diálogo contiene un saludo, una despedida y una lista de frases que formarán la conversación.

```

public class TextoDialogo
{
    [TextArea] public string frase;
}

```

Ilustración 172 Script diálogo

Al crear los objetos encriptables de los diálogos, dentro de la carpeta correspondiente en los assets, podemos llenar estos campos. También podemos decidir la longitud de la lista de textos que forman la conversación y escribir cada uno de ellos por separado y de forma ordenada.

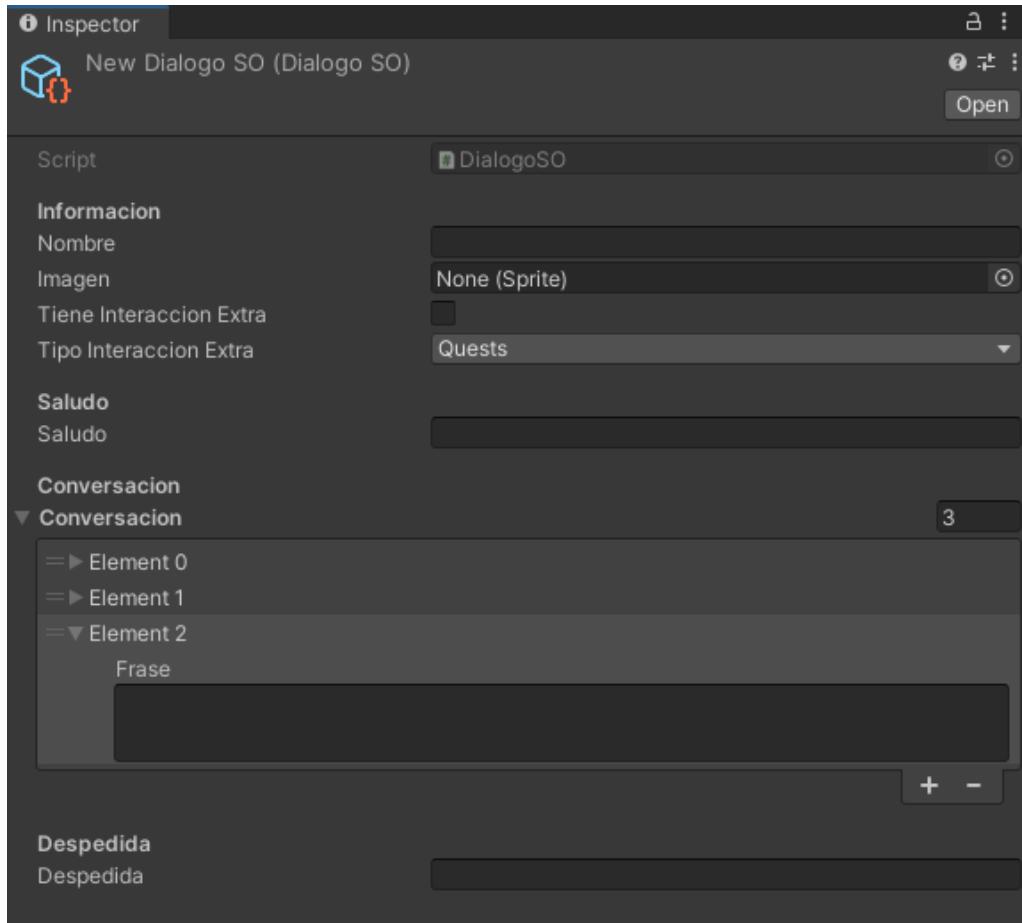


Ilustración 173 Inspector objeto encriptable diálogo

Más adelante creé un manager para gestionar el sistema de diálogo y actualizar la interfaz con los datos correspondientes. En este manager creé los métodos para abrir y cerrar el panel al empezar o terminar un diálogo y para usar el botón de siguiente. Al abrir la conversación, el primer texto que aparece es el saludo. Al pulsar siguiente, aparece el primer texto de la lista, y va avanzando por los siguientes a medida que seguimos usando el botón. Cuando ya han salido todos los textos de la lista, el botón siguiente muestra el mensaje de despedida. Tras esto, el botón siguiente cierra el diálogo.

```
public void BotonSiguiente()
{
    if (despedido)
    {
        AbrirCerrarDialogo(estado: false);
        despedido = false;
        return;
    }
    if (NPCInteractable.Dialogo.TieneInteraccionExtra)
    {
        UIManager.Instance.AbrirPanelIteraccion(NPCInteractable.Dialogo.TipoInteraccionExtra);
        AbrirCerrarDialogo(estado: false);
        return;
    }
    ContinuarDialogo();
}
```

Ilustración 174 Script manager diálogo botón siguiente

```

private void ContinuarDialogo()
{
    if (NPCInteractable == null)
    {
        return;
    }
    if (despedido)
    {
        return;
    }
    if (secuenciaDialogo.Count == 0)
    {
        string despedida = NPCInteractable.Dialogo.Despedida;
        MostrarTexetoAnimado(despedida);
        despedido = true;
        return;
    }
    string siguienteFrase = secuenciaDialogo.Dequeue();
    MostrarTexetoAnimado(siguienteFrase);
}

```

Ilustración 175 Script manager diálogo continuar diálogo

El manager de diálogo también contiene el método que actualiza la interfaz con las imágenes y textos correspondientes a cada uno de los diálogos y de los NPC a los que pertenecen. Aquí volví a hacer uso de TextMeshPro para actualizar los textos.

```

private void ConfigurarPanel(DialogoSO dialogo)
{
    AbrirCerrarDialogo(estado: true);
    CargarSecuenciaDialogo(dialogo);
    icono.sprite = dialogo.Imagen;
    nombreTMP.text = $"{dialogo.Nombre}:";
    MostrarTexetoAnimado(dialogo.Saludo);
}

```

Ilustración 176 Script manager diálogo actualizar datos panel

Por último, para darle un toque más dinámico a las conversaciones, creé un pequeño método para que mostrara una por una las letras de cada texto que conforma la conversación, en lugar de mostrar todo el texto de una vez. A efectos prácticos no tiene ninguna utilidad, pero estoy acostumbrada a ver los textos de esta forma en los videojuegos y se me hacía raro no incluirlo.

```

private IEnumerator AnimaTexto(string frase)
{
    dialogoAnimado = false;
    conversacionTMP.text = "";
    char[] letras = frase.ToCharArray();
    for (int i = 0; i < letras.Length; i++)
    {
        conversacionTMP.text += letras[i];
        yield return new WaitForSeconds(0.001f);
    }
    dialogoAnimado = true;
}

```

Ilustración 177 Script manager diálogo animación texto

Cuanto todo el sistema de diálogos estaba completo, a falta de programar las mecánicas que conllevan cada una de las interacciones extra, creé todas las conversaciones del juego en la carpeta correspondiente de objetos encriptables, en assets, y se los asigné a los NPC correspondiente. A continuación, un ejemplo del diálogo asignado al NPC que nos ofrecerá misiones en las distintas aldeas.

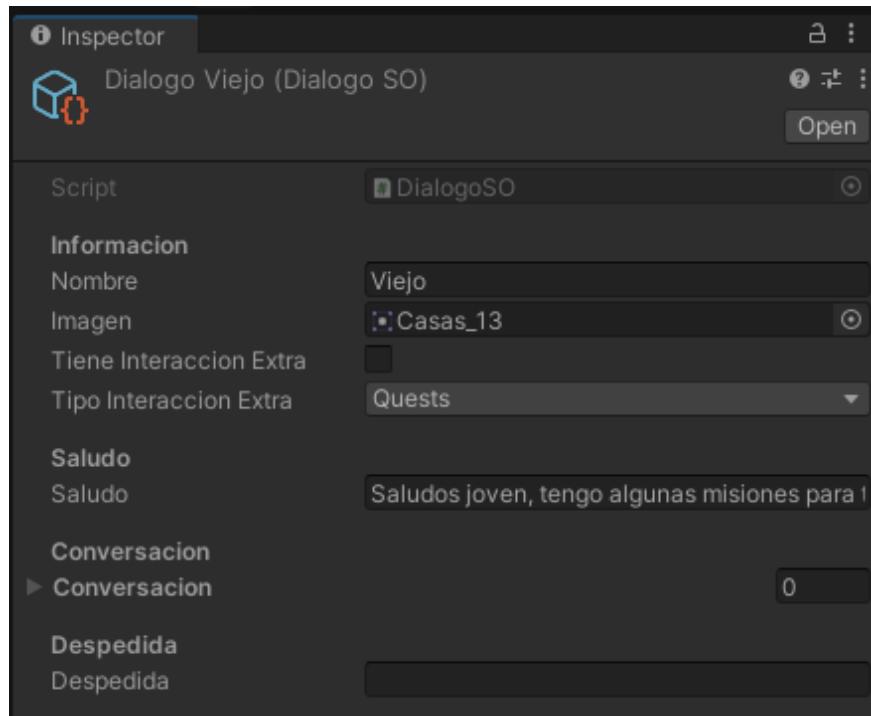


Ilustración 178 Inspector objeto encriptable diálogo misiones

Así queda la interfaz de los diálogos actualizada con los datos del diálogo creado:



Ilustración 179 Interfaz diálogo actualizada

Sistema de dinero

El dinero es otra de las mecánicas presentes en casi todos los videojuegos RPG. Lo necesitamos para comprar objetos que nos ayudarán a completar la historia. Para gestionar esta mecánica en mi juego, he creado un manager. Creé un script donde estarían todas las mecánicas relacionadas al dinero y asigné este script a un objeto vacío dentro del objeto que contiene a todos los managers de la escena.

El script creado contiene la variable en la que indicar desde el inspector la cantidad de dinero con la que empezará la partida, y los métodos para añadir y restar dinero.

```
private void CargarOro()
{
    OroTotal = PlayerPrefs.GetInt(KEY_ORO, oroTest);
}

1 referencia
public void AddOro(int cantidad)
{
    OroTotal += cantidad;
    PlayerPrefs.SetInt(KEY_ORO, OroTotal);
    PlayerPrefs.Save();
}

1 referencia
public void QuitarOro(int cantidad)
{
    if (cantidad > OroTotal)
    {
        return;
    }
    OroTotal -= cantidad;
    PlayerPrefs.SetInt(KEY_ORO, OroTotal);
    PlayerPrefs.Save();
}
```

Ilustración 180 Script manager dinero

Para comenzar la partida, y a modo de prueba, seleccioné que la cantidad de dinero disponible fuera de 10 monedas de oro.

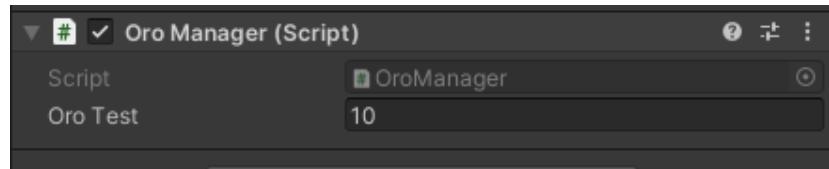


Ilustración 181 Inspector manager dinero

Para actualizar la cantidad de dinero disponible que se muestra en el HUD creé un método en el manager de la interfaz, que instancia la cantidad que tenemos actualmente en el texto que se muestra.

```
oroTMP.text = OroManager.Instance.OroTotal.ToString();
```

Ilustración 182 Script manager interfaz actualizar dinero

Enemigos

A lo largo del juego nos encontraremos con enemigos que intentarán matarnos y a los que nos tendremos que enfrentar para poder avanzar.

Para construir estos enemigos, primero creé las animaciones. Para estas animaciones seguí el mismo proceso que para animar a los NPCs, ya que estarán todo el tiempo en movimiento y no necesitan un árbol de animaciones para transicionar entre una animación y otra.

En la ventana Animation creé una animación para cada uno de los enemigos, usando los spritesheet correspondientes para cada uno de ellos: slimes verdes, azules y naranjas; y llamas rojas y blancas. Estas animaciones simulan que están andando.



Ilustración 183 Spritesheet enemigo

Tras arrastrar los spritesheets recortados a la ventana Animation, seleccioné el tiempo en el que quería que cambiara cada sprite.

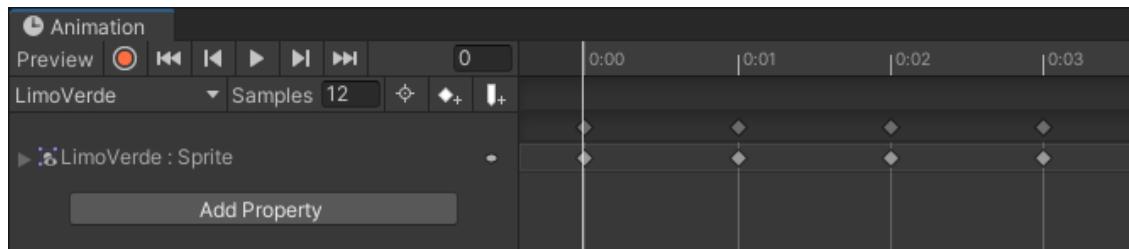


Ilustración 184 Animation enemigo

Para asignar a cada enemigo su animación correspondiente, les añadí desde el inspector el componente Animator, seleccionando desde él la animación.

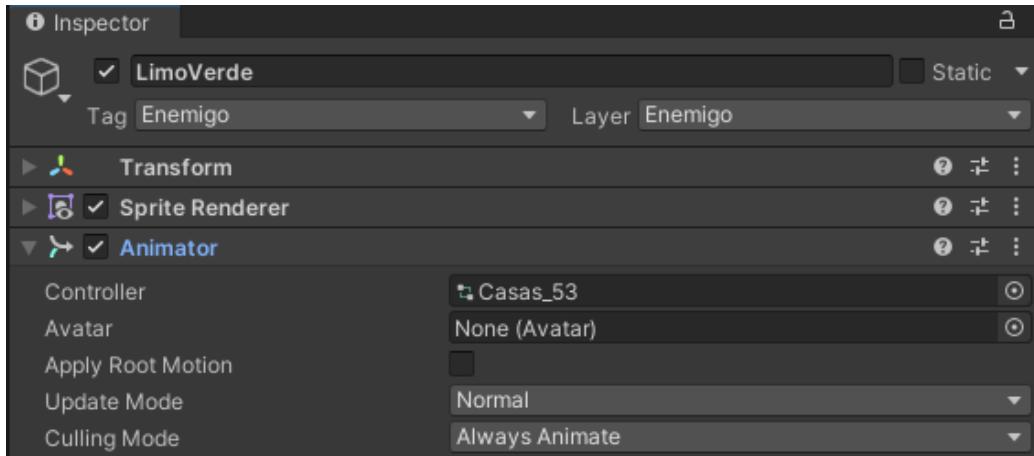


Ilustración 185 Inspector animator enemigo

Para el movimiento de los enemigos utilicé los scripts creados anteriormente para el movimiento de los NPCs, que nos permitía asignarles una ruta, una velocidad y decidir si el movimiento es horizontal o vertical para girar los sprites o no.

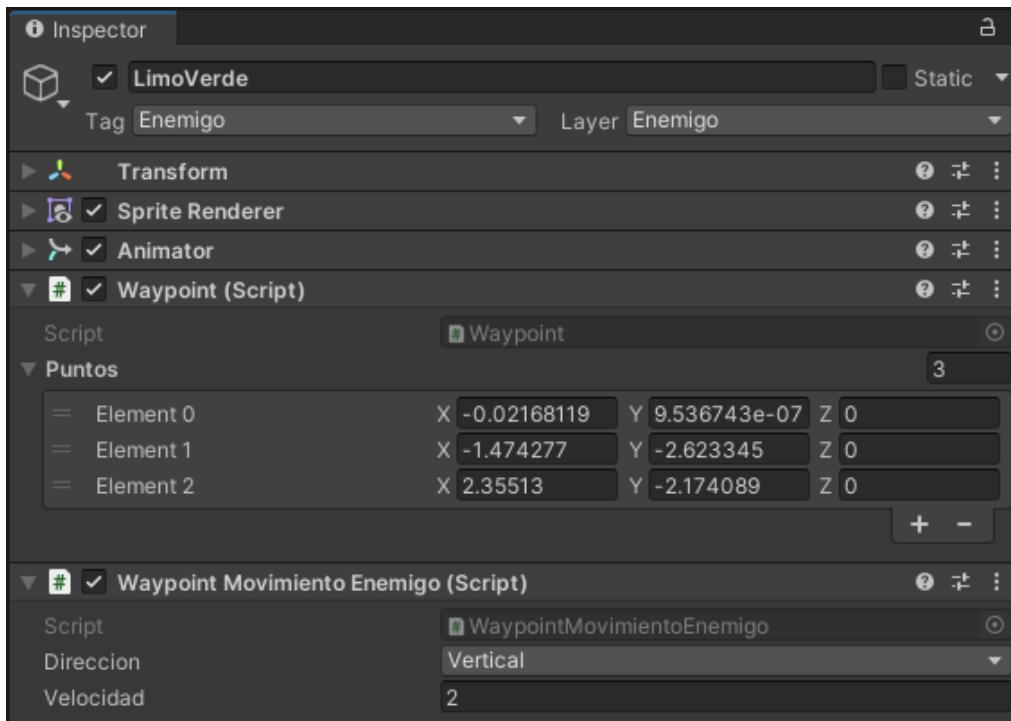


Ilustración 186 Inspector movimiento enemigo

También podemos seleccionar desde la escena los puntos que queremos que formen la ruta que seguirán los enemigos.

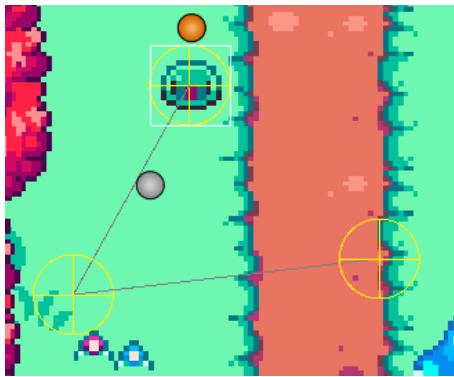


Ilustración 187 Escena movimiento enemigo

Hasta ahora el desarrollo de los enemigos ha sido muy parecido al de los NPCs, pero ahora es cuando empieza a ser más difícil: creé un sistema de IA para los enemigos.

El sistema de IA permite establecer un rango a partir del cual el enemigo perseguirá al protagonista en lugar de seguir su ruta indicada. Si el protagonista sale de este rango, el enemigo volverá a retomar su camino. Además, también permite establecer el rango a partir del cual el enemigo podrá atacar al protagonista. De la misma forma, el enemigo dejará de atacar si el protagonista sale de este rango.

Para crear el sistema de IA he usado estados, los cuales transicionan en base a acciones y decisiones. Creé distintos scripts para los estados, las decisiones, las acciones y las transiciones, y un script controlador para con acceso a los estados y con las variables para indicar cada uno de los rangos necesarios en cada enemigo.



Ilustración 188 Scripts IA enemigos

Los primeros scripts que creé fueron los de acciones y decisiones, ambos como objetos encriptables y con un único método para ejecutar cada una de las acciones y las decisiones haciendo referencia al script controlador. También creé el script de estado, como objeto encriptable y con una cabecera que me permitiera crear estos objetos encriptables desde assets. En el caso de los estados, tiene como parámetro una lista de acciones y de transiciones, y unos métodos para ejecutar las acciones y las transiciones. Para ejecutar las transiciones, se comprueba si el valor de las decisiones es verdadero falso.

```

private void EjecutarAcciones(IAController controlador)
{
    if (Acciones == null || Acciones.Length <= 0)
    {
        return;
    }
    for (int i = 0; i < Acciones.Length; i++)
    {
        Acciones[i].Ejecutar(controlador);
    }
}

```

Ilustración 189 Script IA ejecutar acciones

Luego creé el script para las transiciones, que tiene como parámetros referencias a una decisión y a dos estados: uno verdadero y otro falso.

Para crear el script de controlador, añadí como parámetro un estado inicial y un método que permite cambiar de un estado a otro

En el estado de los enemigos se puede seleccionar que, al comenzar el juego, su acción por defecto sea seguir la ruta indicada. Para crear las transiciones entre estados, podemos usar las decisiones. Las decisiones devuelven verdadero o falso, dependiendo de si se cumple la condición indicada. Al cumplirse o no la decisión, se ejecutará el estado asignado a la decisión verdadera o el asignado a la decisión falsa. Normalmente, uno de estos dos estados será permanecer en el estado actual. Además, a la hora de crear las transiciones, un estado puede usar más de una decisión distinta.

Podemos ver un ejemplo del objeto encriptable del estado de un enemigo en el que está andando por su camino por defecto y persigue al protagonista cuando entra en su rango de ataque. Para que esto funcione, primero creé un método que detecta al protagonista cuando la distancia entre él y el enemigo es menor a la establecida como rango de detección del enemigo.

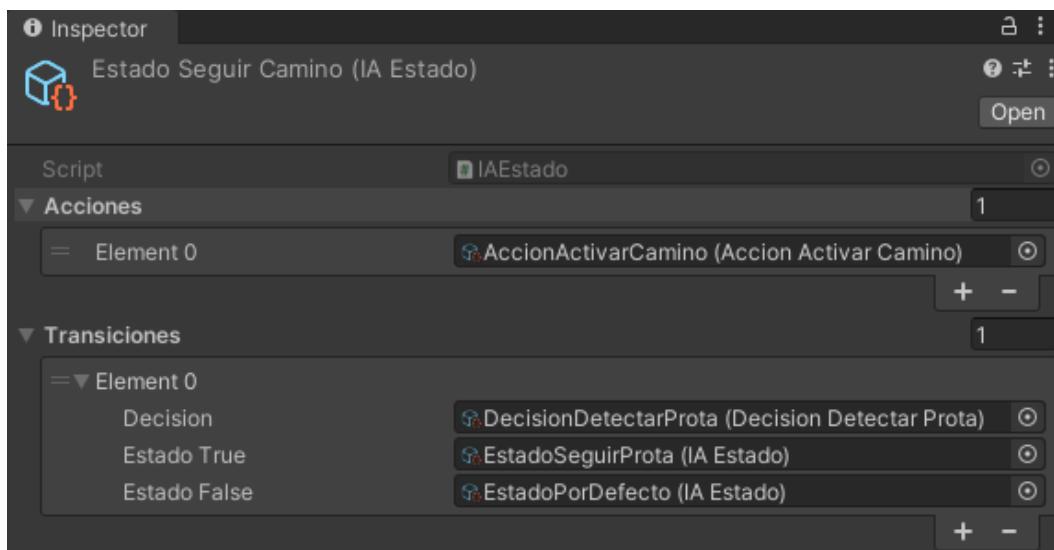


Ilustración 190 Inspector objeto encriptable estado enemigo

Para que los enemigos sigan esta lógica, es necesario añadirles desde el inspector el script controlador del sistema de IA creado. En este mismo script podemos indicar el estado inicial del enemigo y el rango a partir del cuál detectará al protagonista.

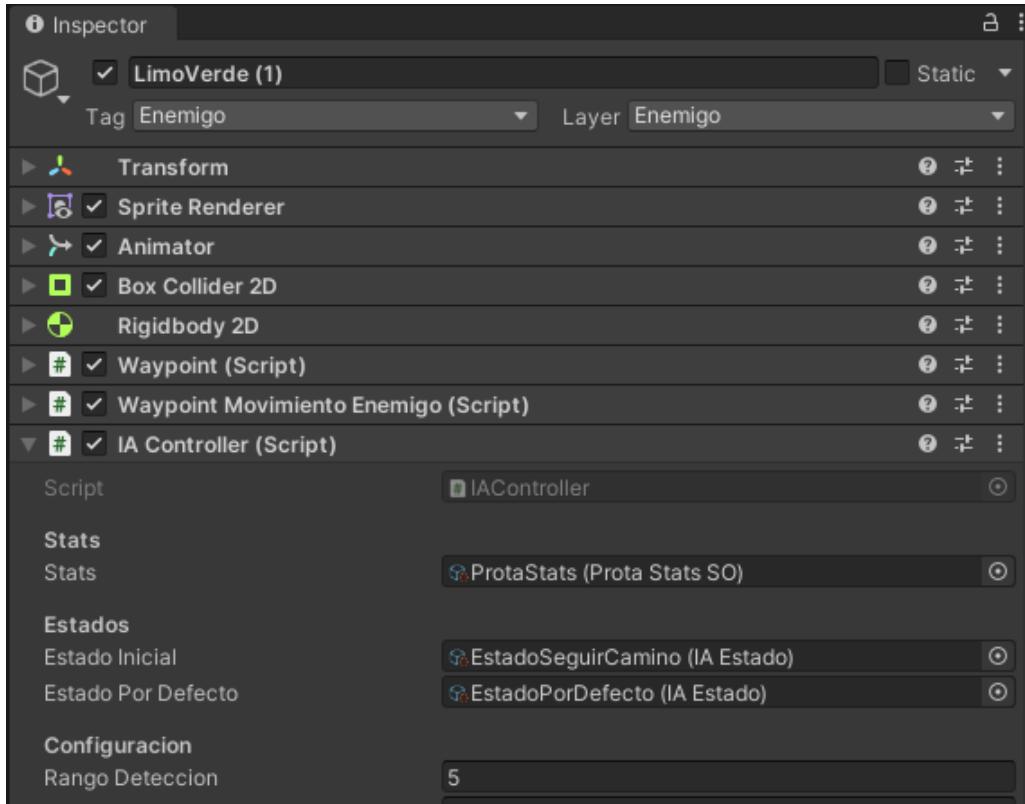


Ilustración 191 Inspector controlador IA enemigo

Además de perseguir al protagonista, el enemigo debe atacarlo. Para esto, añadí un parámetro para establecer el rango de ataque del enemigo y un método para comprobar si el protagonista estaba dentro de este rango.

```
public bool EnAreaAtaque(float rango)
{
    float distancia = (ProtaRef.position - transform.position).sqrMagnitude;
    if (distancia < Mathf.Pow(f:rango, p:2))
    {
        return true;
    }
    return false;
}
```

Ilustración 192 Script detectar protagonista en area de ataque

Ambos rangos, el de perseguir y el de atacar, pueden ser controlados desde la propia escena. He vuelto a ayudarme de los gizmos para poder pintar estos rangos y hacernos una idea del tamaño con respecto a las proporciones de nuestro juego.

```

public void OnDrawGizmos()
{
    if (mostrarRangoAtaque)
    {
        Gizmos.color = Color.magenta;
        Gizmos.DrawWireSphere(transform.position, rangoAtaque);
    }
    if (mostrarRangoAtaqueCargado)
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, rangoAtaqueCargado);
    }
    if (mostrarRangoDeteccion)
    {
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(transform.position, rangoDeteccion);
    }
}

```

Ilustración 193 Script rangos gizmos

En la escena los rangos se representan en forma de círculos. El círculo rojo o magenta es el rango de ataque, y el amarillo es el rango para perseguir al protagonista.

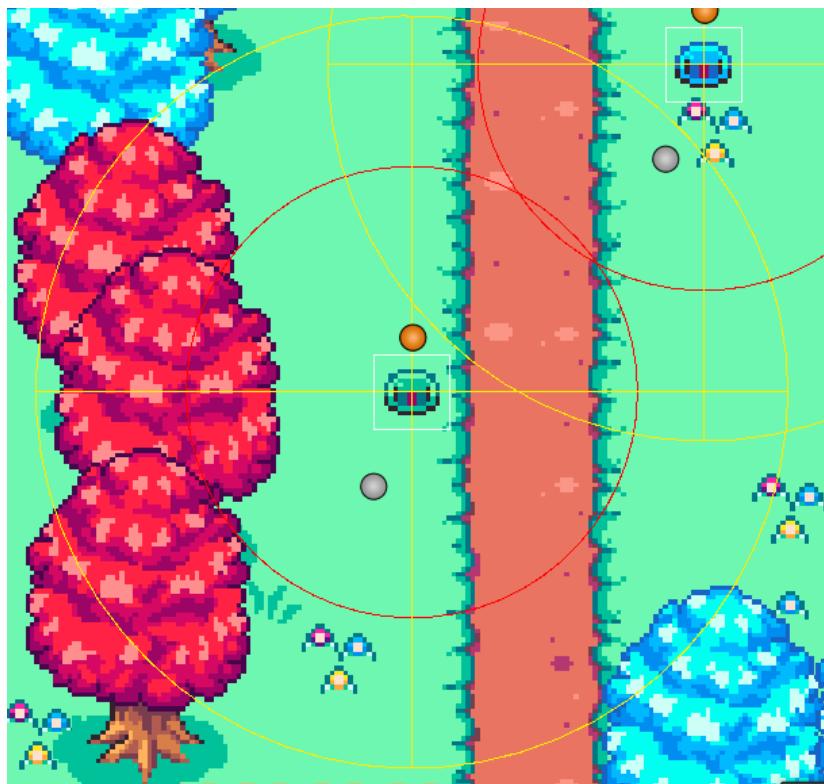


Ilustración 194 Escena rangos gizmos

Para controlar la forma en la que el enemigo persigue al protagonista, creé métodos con vectores que indican la dirección en la que se debe mover el enemigo para perseguirlo y a la distancia que debe quedarse de él, para pegarle desde cierta distancia sin colisionar entre ellos. Para que estos métodos funcionen pasé como parámetros las coordenadas de ambos. De la misma forma, controlé cómo se movería el enemigo para realizar ataques cargados al protagonista.

```

private IEnumerator IECargoado(float cantidad)
{
    Vector3 protaPosicion = ProtaRef.position;
    Vector3 posicionInicial = transform.position;
    Vector3 posicionDestino = (protaPosicion - posicionInicial).normalized;
    Vector3 posicionAtaque = protaPosicion - posicionDestino * 0.5f;
    _boxCollider2D.enabled = false;
    float transicionAtaque = 0f;
    while (transicionAtaque <= 1f)
    {
        transicionAtaque += Time.deltaTime * velocidad;
        float interpolacion = (-Mathf.Pow(f: transicionAtaque, p: 2) + transicionAtaque) * 4f;
        transform.position = Vector3.Lerp(a: posicionInicial, b: posicionAtaque, interpolacion);
        yield return null;
    }
    if (ProtaRef != null)
    {
        HacerPupa(cantidad);
    }
    _boxCollider2D.enabled = true;
}

```

Ilustración 195 Script movimiento para ataque cargado

Ahora, desde el script controlador en los enemigos, podemos señalar también los rangos de los distintos ataques para cada uno de ellos.

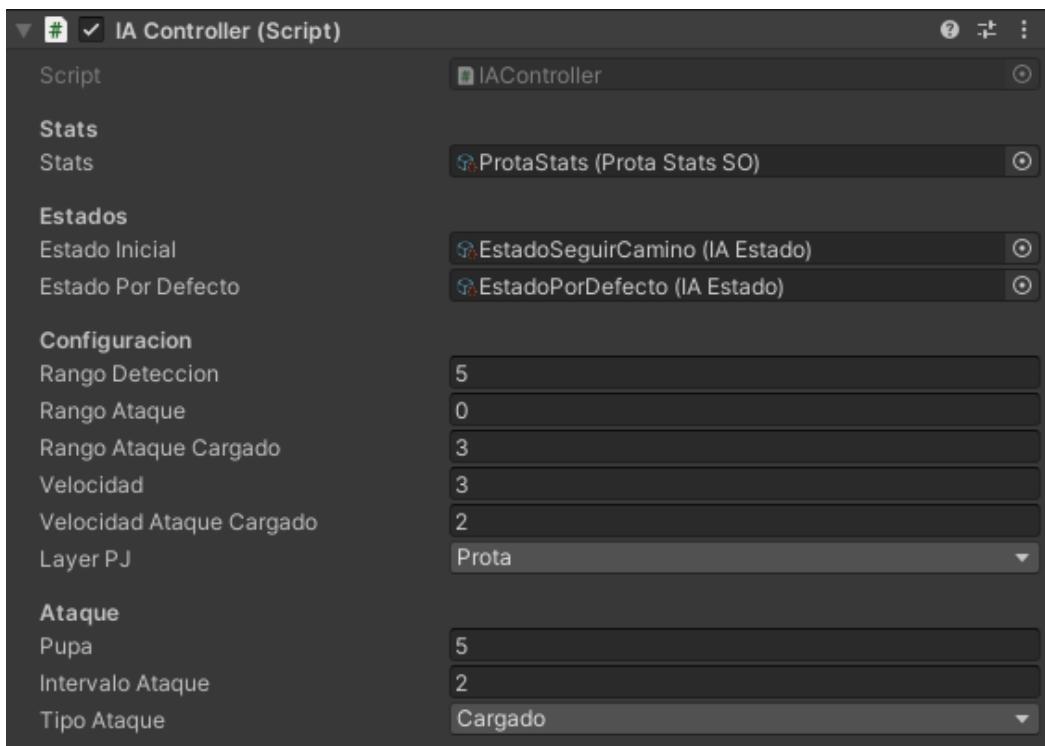


Ilustración 196 Inspector controlador ataques

Con las mecánicas de los enemigos listas, creé todos los enemigos necesarios para el juego y los coloqué en la escena en sus correspondientes sitios (en los bosques). Para tener a los enemigos ordenados, creé un objeto vacío en la jerarquía dentro del cual metí todos los enemigos.

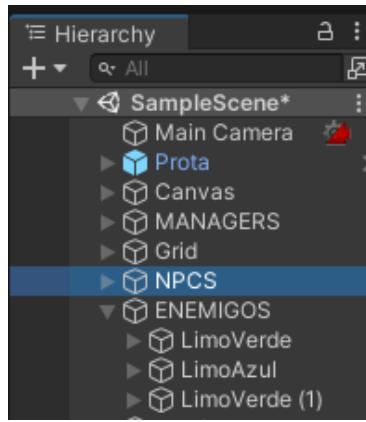


Ilustración 197 Jerarquía enemigos

A continuación, un ejemplo de como quedan los enemigos colocados en el primer bosque.



Ilustración 198 Escena enemigos

Añadí a todos los enemigos, desde el inspector, el script base de vida creado en el Sprint anterior, del que hereda la vida del protagonista. Le asigné a los enemigos del principio menos puntos de vida (a partir de 24) y más a los del final (sobre 100). A todos los enemigos les di el mismo valor de vida inicial y máxima. También les asigné una barra sobre la cabeza con el valor de la vida actual en comparación con la vida máxima, para poder ver el porcentaje de vida que les queda.

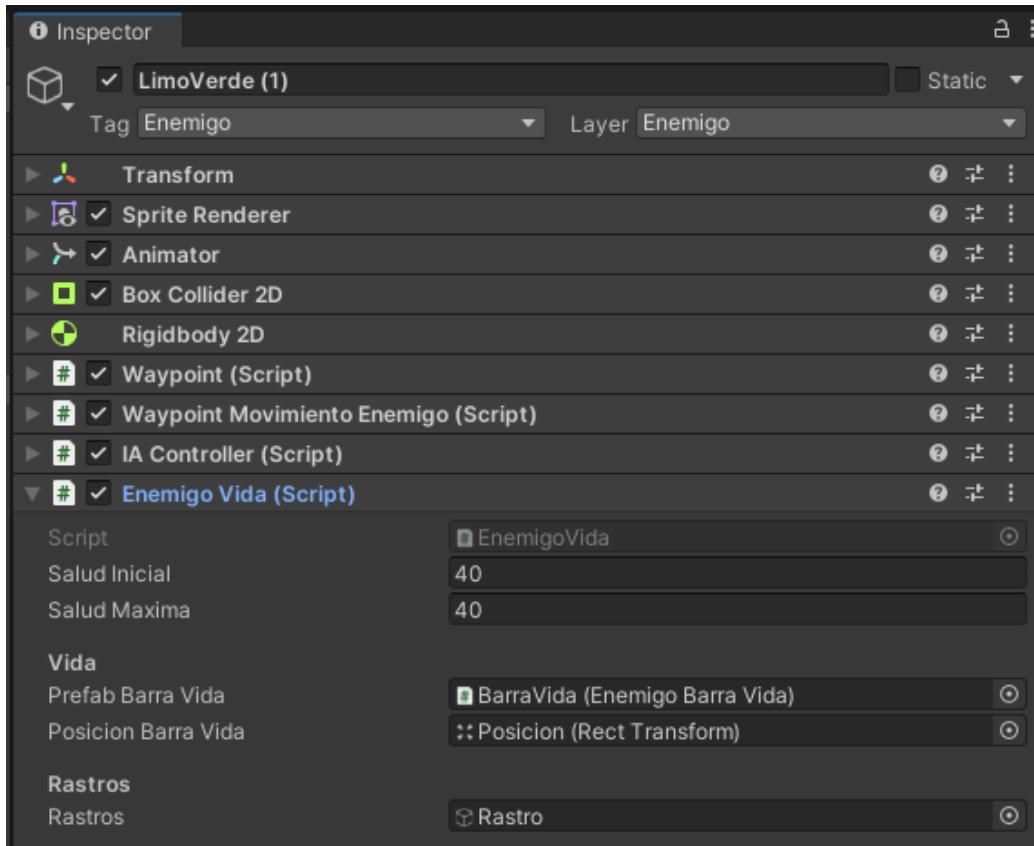


Ilustración 199 Inspector enemigos vida

Para que el protagonista pudiera interactuar con los enemigos, aun quedaba por añadirles dos componentes: el rigidbody y el collider.

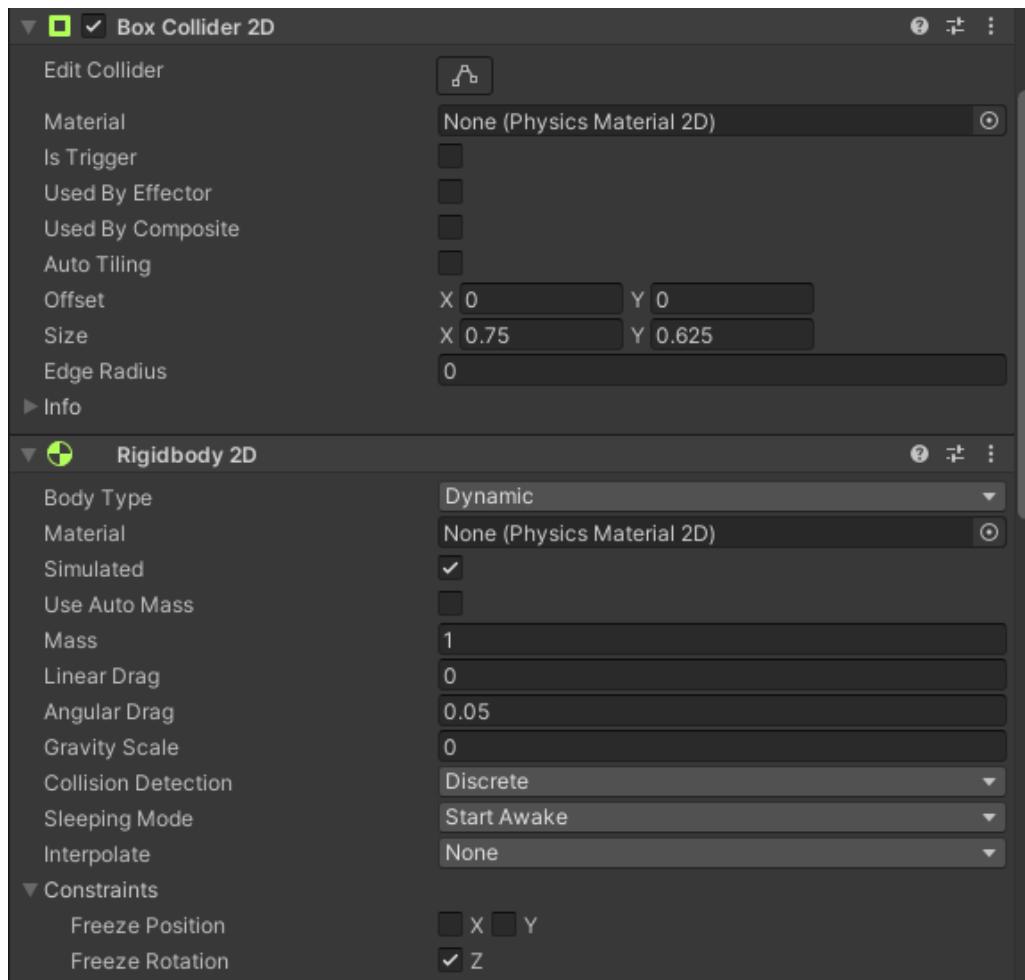


Ilustración 200 Inspector enemigo rigidbody y collider

Armas

Para poder combatir con los enemigos, el protagonista necesitar hacer uso de armas. Las armas no son más que un tipo de objeto que nos ofrece características extras como ataque, probabilidad de crítico o probabilidad de bloqueo. Este tipo de objeto se puede equipar.

Para controlar si el protagonista tiene un arma equipada, creé un pequeño panel que estará siempre presente durante la partida en la parte superior derecha de la pantalla. Este panel está vacío si no hay ningún arma equipada. Si el protagonista se equipa un arma, el arma equipada sale dentro del panel.

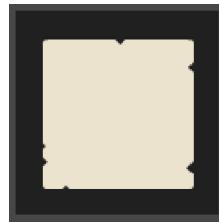


Ilustración 201 Interfaz arma equipada

Para crear las armas, creé un script de tipo objeto encriptable y con una cabecera que me permitiera crear objetos encriptables de este tipo desde la ventana de assets. Este script cuenta con una enumeración para distinguir si el arma es de tipo físico o mágico: si un

arma es mágica, necesitará usar maná para hacer daño. El script cuenta con variables para el ícono del arma, el efecto que hace en caso de ser mágica, el daño, el maná que gasta en caso de ser mágica y las estadísticas que nos aumentan de bloqueo y de daño crítico.

Podemos editar estos campos desde el inspector del objeto encriptable.

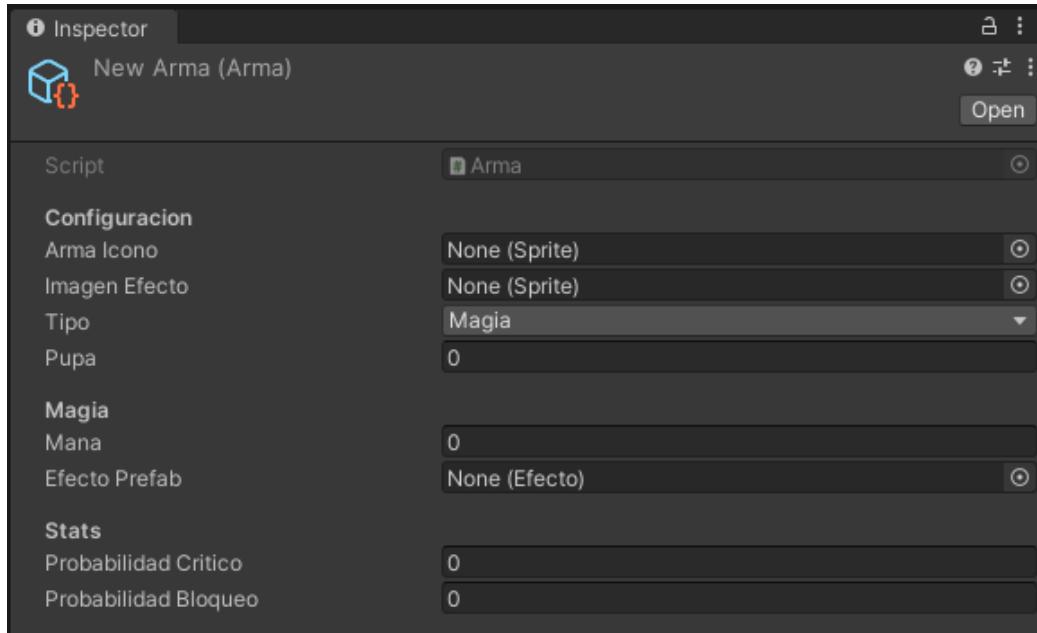


Ilustración 202 Inspector objeto encriptable arma

Creé 8 armas para mi juego: 4 de ellas mágicas y otras 4 físicas. Cada una de las armas tiene distintas características, de forma que las armas que otorgan más daño serán mejores para completar la historia, pero solo se podrán conseguir una vez avanzada la misma.

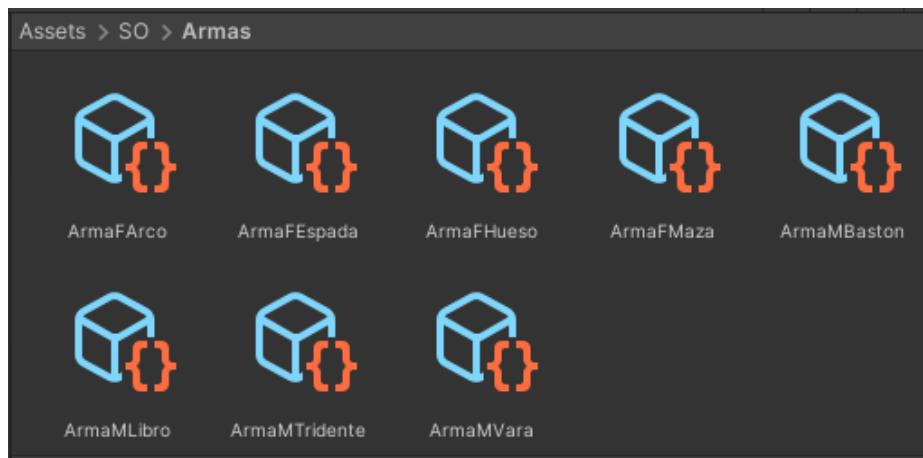


Ilustración 203 Carpeta objetos encriptables armas

A continuación muestro cada una de las armas creadas como objetos encriptables y los parámetros establecidos para cada una de ellas desde el inspector.

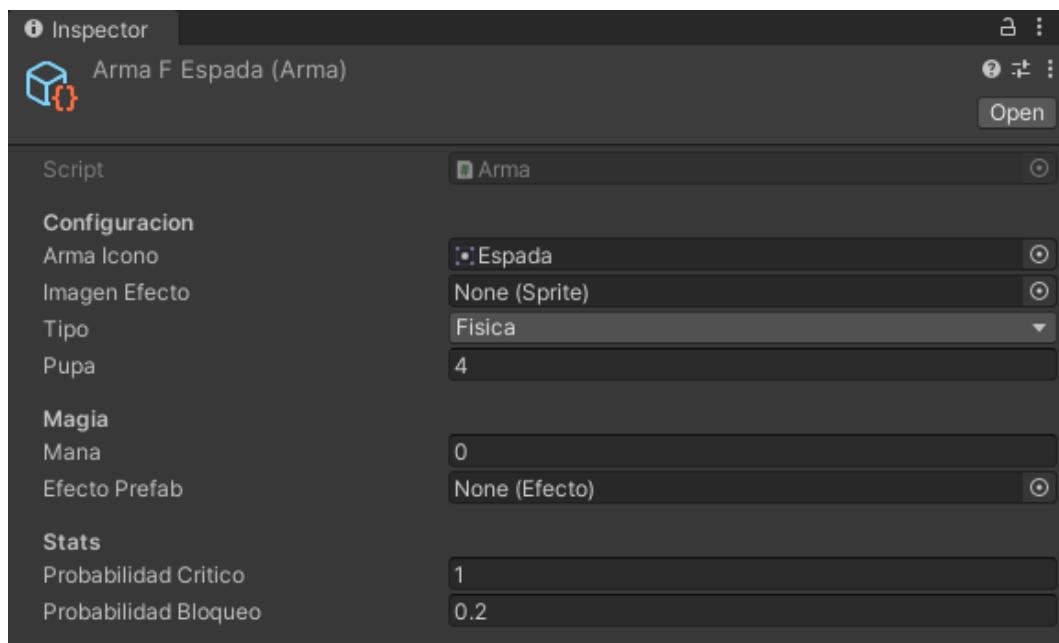


Ilustración 204 Inspector arma física 1

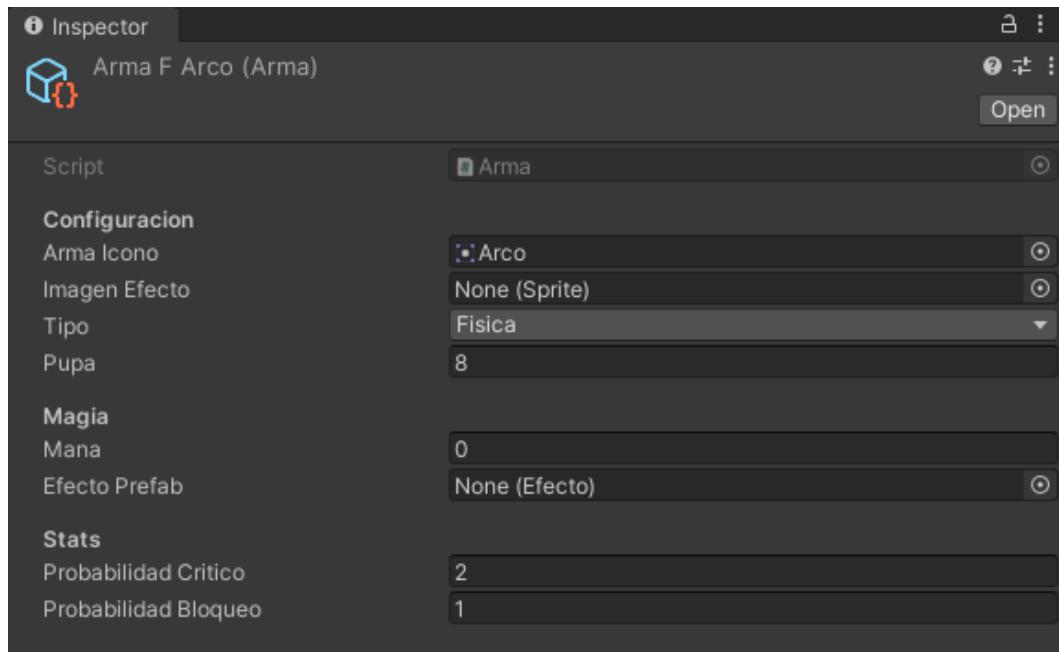


Ilustración 205 Inspector arma física 2

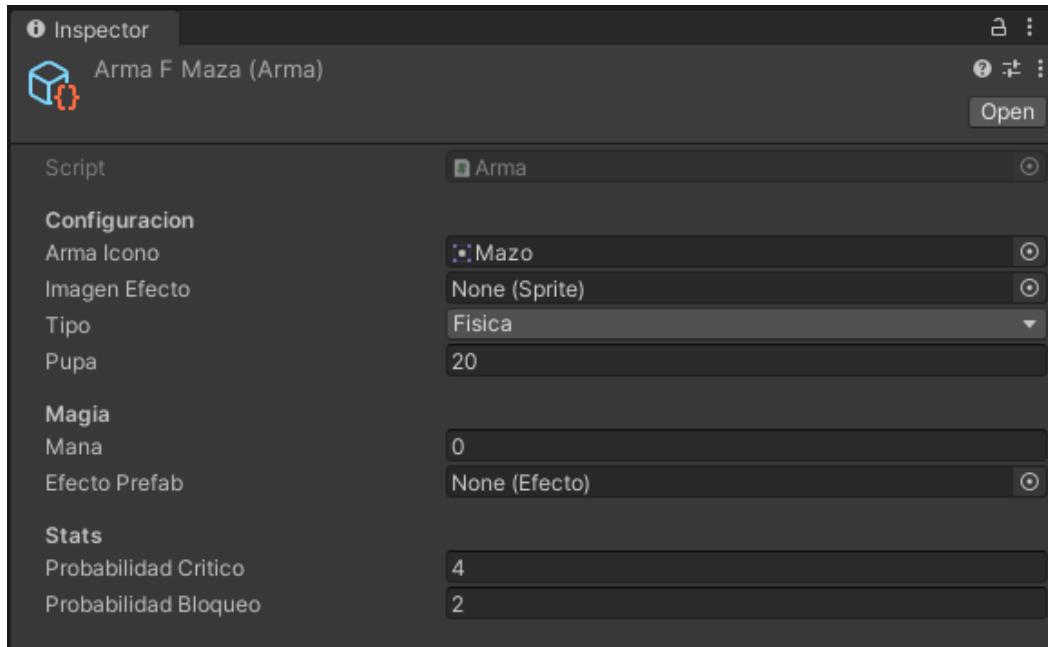


Ilustración 206 Inspector arma física 3

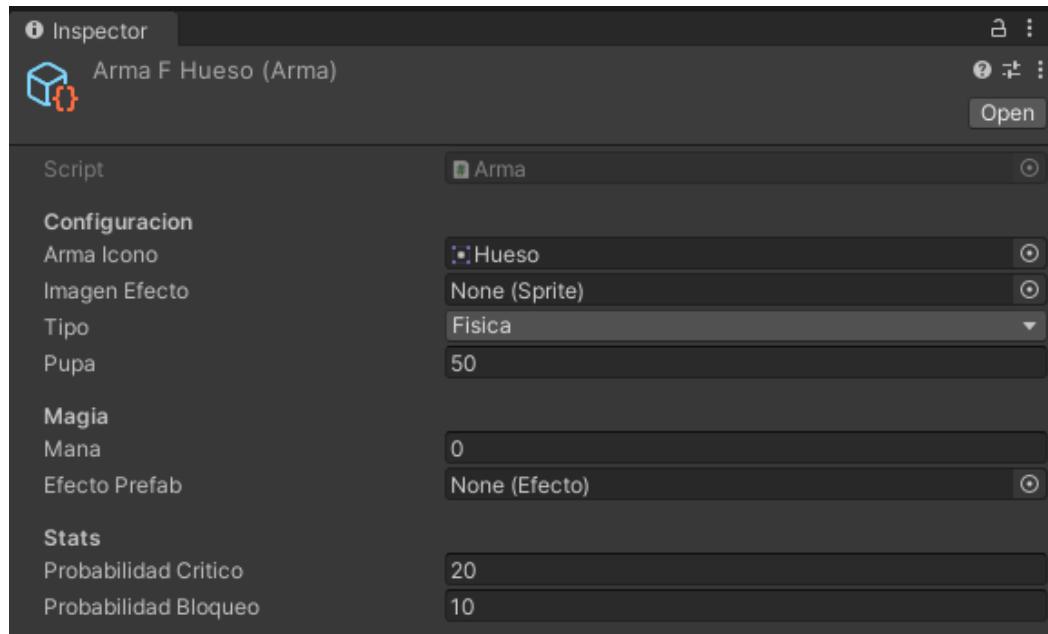


Ilustración 207 Inspector arma física 4

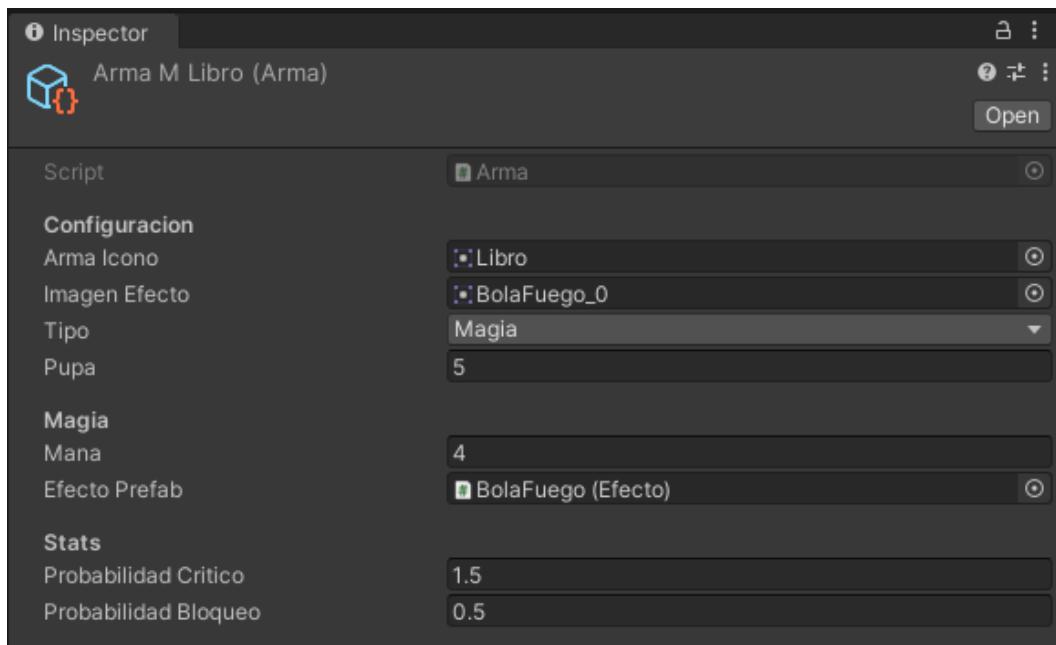


Ilustración 208 Inspector arma mágica 1

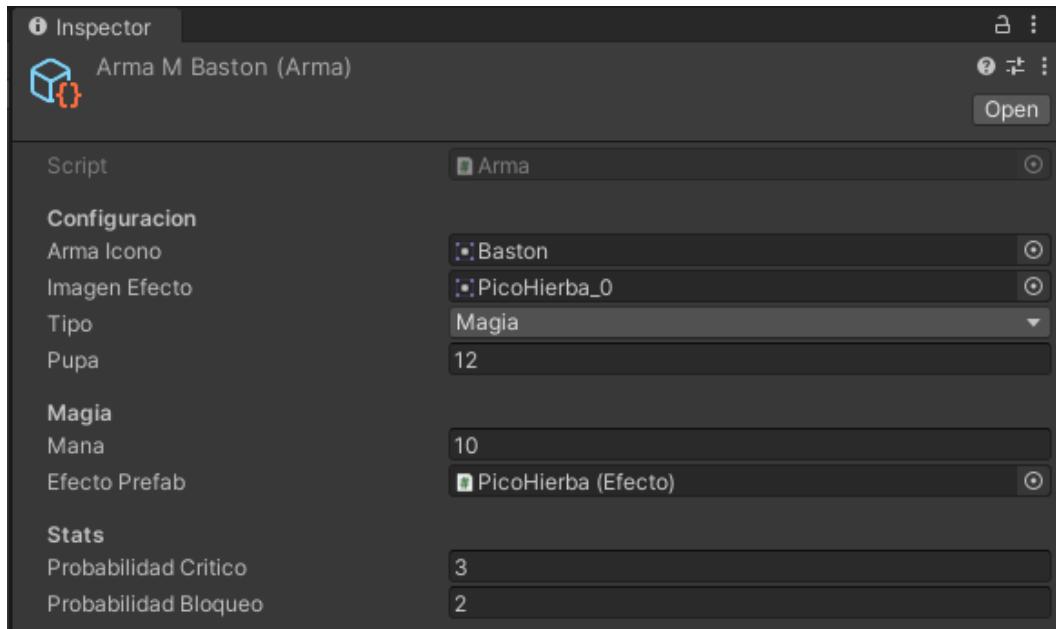


Ilustración 209 Inspector arma mágica 2

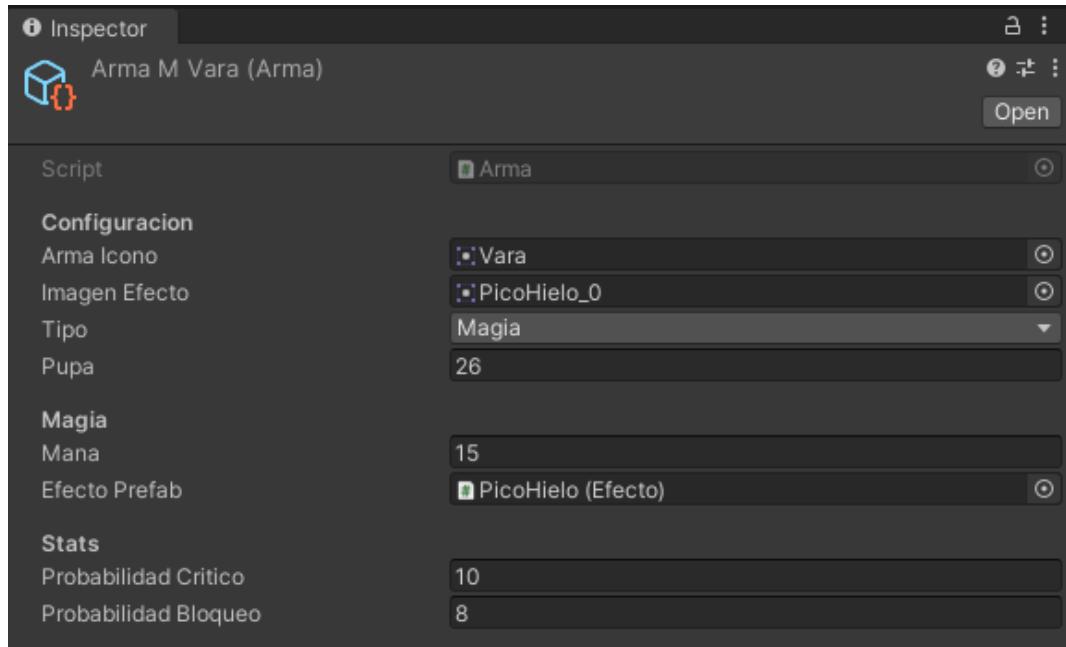


Ilustración 210 Inspector arma mágica 3

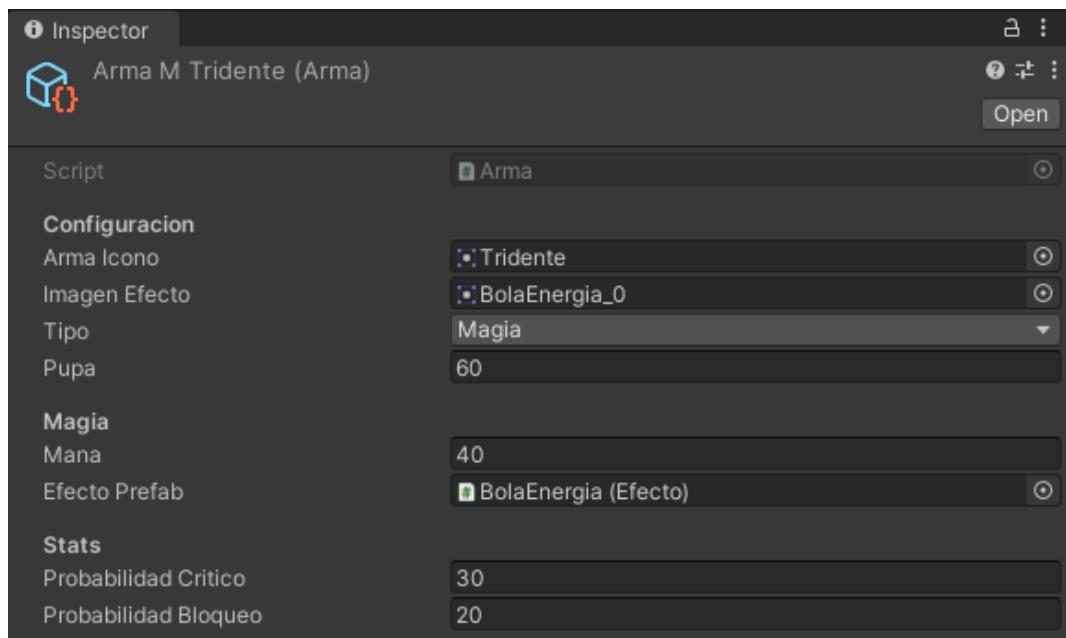


Ilustración 211 Inspector arma mágica 4

Para poder equipar un arma al protagonista, sobrescribí el método que respondía al evento de pulsar el botón equipar sobre un objeto. Esto comprueba que el objeto que intentamos equiparnos sea un arma y que no tengamos un arma equipada anteriormente. Si se cumplen estas condiciones, el arma se equipa; incrementando nuestras estadísticas con las indicadas en el arma y permitiéndonos atacar con ellas. Sobrescribí también el método de desequipar arma.

Por último, conecté los datos del arma equipada con la interfaz, que en este caso era el panel de la parte superior derecha de la pantalla. En este panel se muestra el ícono del arma equipada y se quita este ícono al desequipar el arma.

```

public void EquivarArma(ItemArma itemArma)
{
    ArmaEquipada = itemArma;
    armaIcono.sprite = itemArma.Arma.ArmaIcono;
    armaIcono.gameObject.SetActive(true);
    Inventario.Instance.Prota.ProtaAtaque.EquivarArma(itemArma);
}

1 referencia
public void DesequiparArma()
{
    armaIcono.gameObject.SetActive(false);
    ArmaEquipada = null;
    Inventario.Instance.Prota.ProtaAtaque.QuitarArma();
}

```

Ilustración 212 Script interfaz arma equipada

Sistema de combate

Para terminar el segundo Sprint, creé el sistema de combate mediante el cual podrían interactuar el protagonista y los enemigos, dañándose entre ellos e incluso matándose.

Para que el protagonista pueda seleccionar a los enemigos antes de atacarlos, he creado un script que sirva de manager y gestione este tipo de interacción. Como ya indiqué previamente, los enemigos se seleccionarían haciendo click en ellos (o entrando en el rango de ataque si llevamos equipada un arma física). Por tanto, utilicé raycast para comprobar si hay enemigos en el sitio clicado y, en el caso de que si lo haya, seleccionarlo. Al seleccionar un enemigo, se lanza un evento. Al clicar en otro lado y deseleccionar el enemigo, se lanza otro evento. Esta selección funciona mientras el enemigo siga vivo. Una vez muerto, al clicar este enemigo veremos su loot (no lo seleccionaremos).

```

if (Mouse.current.leftButton.wasPressedThisFrame)
{
    RaycastHit2D hit = Physics2D.Raycast(camara.ScreenToWorldPoint(Mouse.current.position.ReadValue()),
        direction:Vector2.zero, distance:Mathf.Infinity, LayerMask.GetMask("Enemigo"));
    if (hit.collider != null)
    {
        EnemigoTarget = hit.collider.GetComponent<EnemigoInteraccion>();
        EnemigoVida enemigoVida = EnemigoTarget.GetComponent<EnemigoVida>();
        if (enemigoVida.Salud > 0f)
        {
            EventoEnemigoSeleccionado?.Invoke(EnemigoTarget);
        }
        else
        {
            EnemigoLoot loot = EnemigoTarget.GetComponent<EnemigoLoot>();
            LootManager.Instance.MostrarLoot(loot);
        }
    }
    else
    {
        EventoNoEnemigoSeleccionado?.Invoke();
    }
}

```

Ilustración 213 Script manager seleccionar enemigo arma mágica

En cuanto a la selección por entrar en el campo de ataque usando un arma física, lo gestioné con un collider de tipo trigger alrededor del protagonista y métodos que

comparan si hay enemigos en este collider. En este tipo de selección también se lanzan eventos cuando el enemigo entra en el campo y se selecciona y cuando sale de él y se deselecciona.

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Enemigo"))
    {
        EnemigoDetectado = other.GetComponent<EnemigoInteraccion>();
        if (EnemigoDetectado.GetComponent<EnemigoVida>().Salud > 0)
        {
            EventoEnemigoDetectado?.Invoke(EnemigoDetectado);
        }
    }
}

⌚ Mensaje de Unity | 0 referencias
private void OnTriggerExit2D(Collider2D other)
{
    if (other.CompareTag("Enemigo"))
    {
        EventoNoEnemigoDetectado?.Invoke();
    }
}
```

Ilustración 214 Script manager selección enemigo arma física

Dependiendo de por cuál de estos dos métodos se haya seleccionado el enemigo, se mostrará un pequeño círculo rojo (en el caso de ser seleccionado con un arma física) o blanco (si es seleccionado de lejos con un arma mágica) bajo el enemigo, para poder identificar fácilmente cuál es el enemigo seleccionado.

```
public void MostrarEnemigoSeleccionado(bool estado, TipoDeteccion tipo)
{
    if (tipo == TipoDeteccion.Magia)
    {
        seleccionMagia.SetActive(estado);
    } else
    {
        seleccionFisica.SetActive(estado);
    }
}

1 referencia
public void DesactivarSpritesSeleccion()
{
    seleccionMagia.SetActive(false);
    seleccionFisica.SetActive(false);
}
```

Ilustración 215 Script selección mágica o física

Por ejemplo, este es un ejemplo de un enemigo de tipo slime siendo seleccionado con un arma física:



Ilustración 216 Escena enemigo seleccionado con arma física

Cuando el protagonista ha seleccionado un enemigo, puede atacarle. Para esto, creé un script de ataque que asigné al protagonista. Este script tiene acceso a las estadísticas del protagonista. También contiene las variables que se podrán editar desde el inspector del tiempo que tenemos que esperar entre cada ataque. Además, este script contiene los métodos para responder a los eventos de enemigos seleccionados y permitirnos atacarlos en ese momento. Para atacarlos, también creé el método para poder usar el arma equipada, comprobando antes si es de tipo físico o mágico. Al usar el arma sobre el enemigo, lanzamos un evento de dañar al enemigo.

```
private void UsarArma()
{
    if (ArmaEquipada.Tipo == TipoArma.Magia)
    {
        if (_protaMana.ManaActual < ArmaEquipada.Mana)
        {
            return;
        }
        _protaMana.UsarMana(ArmaEquipada.Mana);
    }
    else
    {
        float pupa = ObtenerPupa();
        EnemigoVida enemigoVida = EnemigoTarget.GetComponent<EnemigoVida>();
        enemigoVida.RecibirPupa(pupa);
        EventoEnemigoPupa?.Invoke(pupa, enemigoVida);
    }
}
```

Ilustración 217 Script ataque protagonista usar arma

La cantidad de daño que se hace en cada ataque se calcula con las estadísticas de daño del protagonista y las del arma, y además se genera un valor que, comparándolo con las estadísticas de daño crítico, permite o no hacer el doble de daño en ese golpe.

```
public float ObtenerPupa()
{
    float cantidad = stats.Pupa;
    if (Random.value < stats.PorcentajeCritico / 100)
    {
        cantidad *= 2;
    }
    return cantidad;
}
```

Ilustración 218 Script ataque protagonista calcular daño

El script tiene también como parámetro un pooler que configuré para que soltara números por pantalla indicando el daño que ha recibido el protagonista en cada golpe.

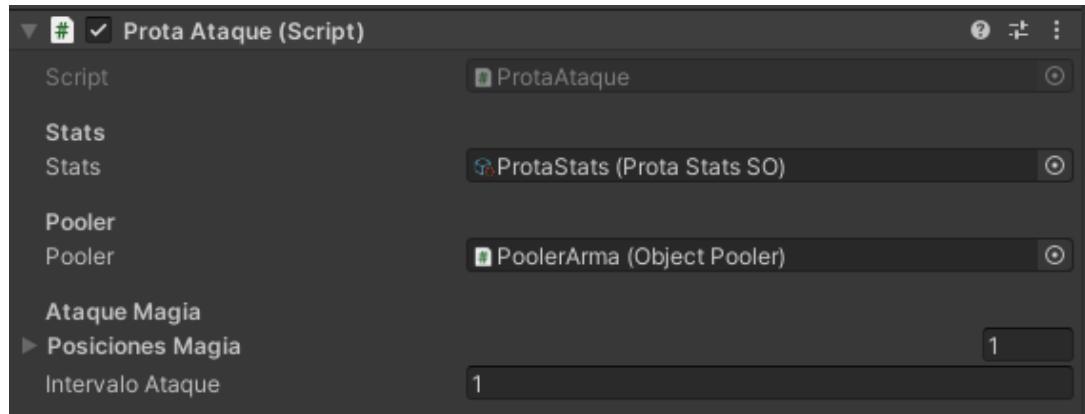


Ilustración 219 Inspector ataque protagonista

Este pooler lo configuré en un script aparte y lo conecté con la interfaz gracias a TextMesh Pro.

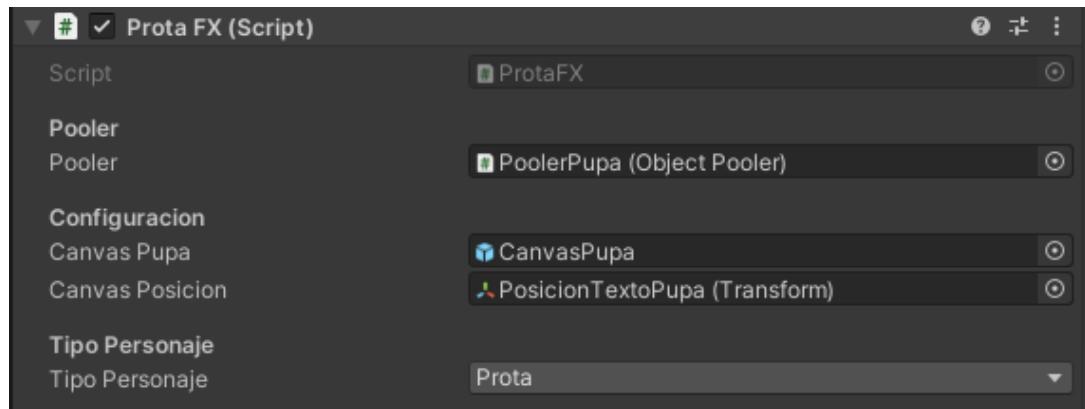


Ilustración 220 Inspector pooler daño protagonista

Para indicar la zona en la que debían salir los números del daño recibido, usé coordenadas del prefab del protagonista y usé un ícono de un rombo rosa para poder ver bien estas coordenadas.

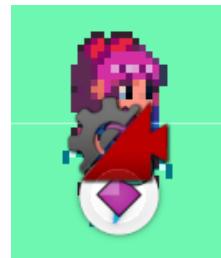


Ilustración 221 Escena coordenadas pooler daño protagonista

Este sistema de pooler para poder ver el daño realizado lo utilicé también sobre los enemigos. De esta forma, además de poder ver el porcentaje de vida que le queda a cada enemigo en la barra sobre su cabeza, se puede ver cuántos puntos de daño reciben por parte del protagonista en cada golpe. Les añadí el script desde el inspector y los configuré de forma parecida, pero con las coordenadas de cada uno de los enemigos.

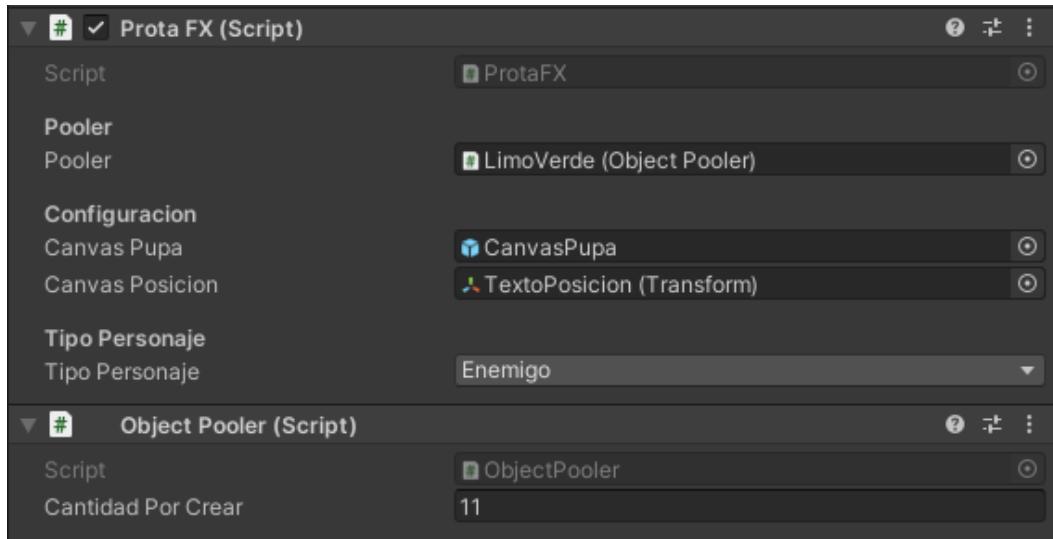


Ilustración 222 Inspector pooler daño enemigos

También usé iconos para mostrar en la interfaz las coordenadas del pooler. En este caso usé círculos naranjas.

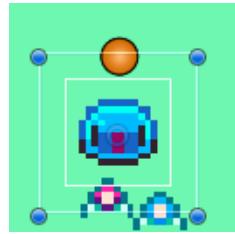


Ilustración 223 Escena pooler daño enemigos

Sprint 3

Sistema de looteo

Normalmente, en los juegos de rol, al derrotar enemigos podemos conseguir objetos. Para conseguir esta dinámica en mi juego creé un sistema de looteo. A grandes rasgos, creé los menús y scripts necesarios para asignar a los enemigos una lista de objetos junto a un porcentaje con la probabilidad de que, al morir, el enemigo nos de los objetos asociados a él. Para poder acceder a estos objetos, creé un menú que se puede abrir clicando el enemigo muerto donde se ven estos objetos y nos permite recogerlos y añadirlos a nuestro inventario.

El primer paso fue crear en el canvas la interfaz del menú donde se muestran los objetos que podemos lootear de los enemigos. Este menú es un rectángulo al que accedemos clicando un enemigo muerto y que cerramos a la X de la esquina superior derecha. Dentro de él aparecerán los objetos que nos de cada enemigo.

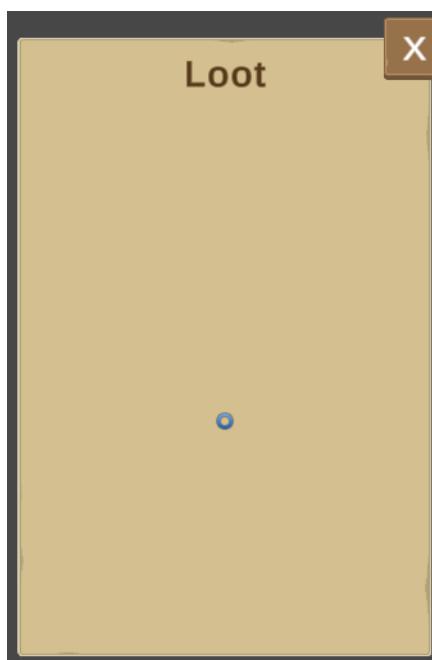


Ilustración 224 Interfaz looteo

Antes de programar los scripts para permitirnos lootear, edité el script de vida de los enemigos de forma que, cuando se quedan sin vida y mueren, se desactivan su movimiento, el ataque y las imágenes y animaciones. Al desactivar estas características del enemigo, se activa una imagen de una sombra en el lugar donde mueren que nos permite interactuar a través de un clic.

También creé un método que lanza el evento que otorga experiencia al personaje cuando el enemigo es desactivado. La experiencia que da cada enemigo se editaría desde el script del loot del propio enemigo.

```

private void DesactivarEnemigo()
{
    _spriteRenderer.enabled = false;
    _iaController.enabled = false;
    _boxCollider2D.isTrigger = true;
    _enemigoInteraccion.DesactivarSpritesSeleccion();
    _enemigoMovimiento.enabled = false;
    _enemigoBarraVida.gameObject.SetActive(false);
    rastros.SetActive(true);
}

2 referencias
protected override void PersonajeMorido()
{
    DesactivarEnemigo();
    EventoEnemigoMorido?.Invoke(_enemigoLoot.Exp);
}

```

Ilustración 225 Script desactivar enemigo

Así es como se ve el enemigo cuando se desactiva y podemos lootearlo:



Ilustración 226 Enemigo muerto

Para empezar a programar el looteo, primero hice un script para crear cada uno de los objetos que pueden soltar los enemigos y que más tarde se asignarán a ellos a través de una lista. Estos objetos looteables hacen referencia a un objeto ya existente del videojuego, un nombre y la cantidad que podemos obtener de él cada vez. Además de los datos del propio objeto, contienen una variable de un porcentaje que indica la probabilidad de que un enemigo nos suelte ese objeto.

```

public class DropItem
{
    [Header("Configuracion")]
    public string Nombre;
    public InventarioItem Item;
    public int Cantidad;

    [Header("Drop")]
    [Range(0, 100)]public float PorcentajeDrop;

    2 referencias
    public bool ItemRecogido { get; set; }
}

```

Ilustración 227 Script objeto looteable

Luego creé otro script assignable a los enemigos con una lista de objetos looteables editable desde el inspector. Este script también contiene como variables la experiencia que dará el enemigo al protagonista. Además, contiene un método que genera un número

del 0 al 100 para cada objeto de la lista y compara este número con el porcentaje asignado al objeto para comprobar si este objeto se generará al matarlo o no.

```
private void SeleccionarLoot()
{
    foreach (DropItem item in lootDisponible)
    {
        float probabilidad = Random.Range(0, 100);
        if (probabilidad <= item.PorcentajeDrop)
        {
            lootSeleccionado.Add(item);
        }
    }
}
```

Ilustración 228 Script loot enemigo

Podemos ver que, al asignar este script a un enemigo, podemos editar la experiencia y los objetos que podemos recibir de él. Añadí el script a todos los enemigos del videojuego dando más punto de experiencia a los enemigos más fuertes y que aparecen en la parte más tardía del videojuego. Además, a los enemigos más fuertes le añadí más objetos y mejores a su lista de loot.

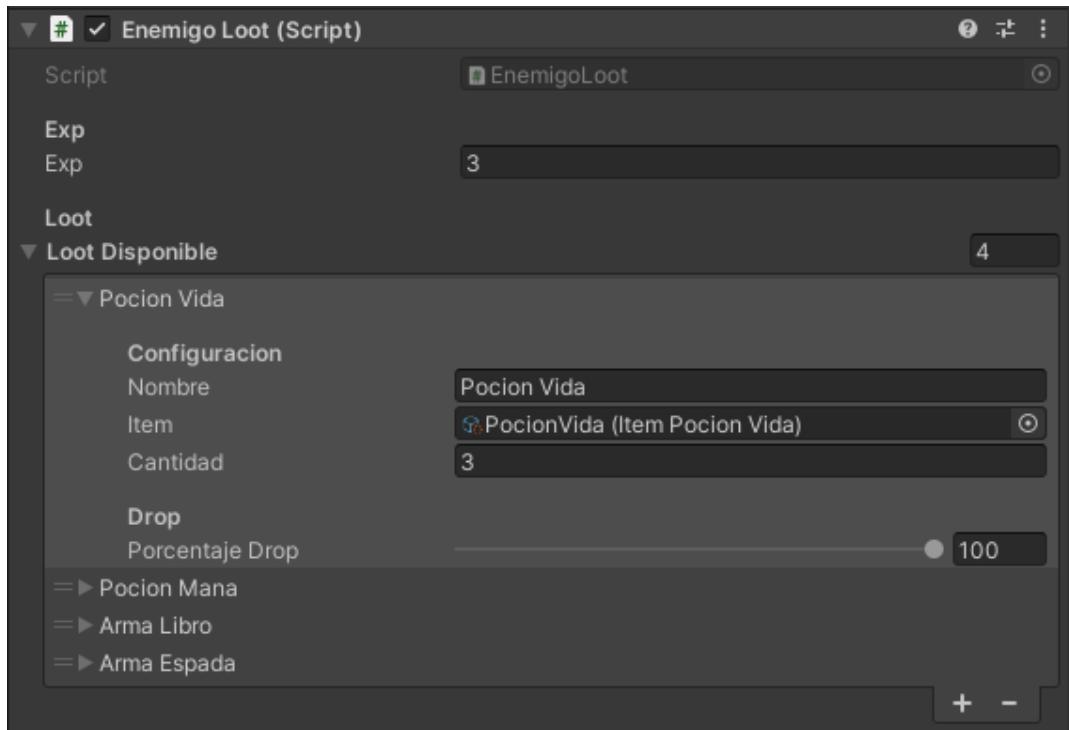


Ilustración 229 Inspector loot enemigo

Más tarde creé un manager con métodos para cargar los objetos al panel de looteo y actualizar la interfaz donde se muestran.

Para mostrar en el panel de loot los objetos que suelta un enemigo, se instancia cada uno de los objetos en un contenedor en el panel al matar al enemigo y mientras no se hayan recogido esos objetos.

```

private void CargarLootAlMenu(DropItem item)
{
    if (item.ItemRecogido)
    {
        return;
    }
    LootBoton loot = Instantiate(prefabLootBoton, contenedor);
    loot.ConfigurarLootItem(item);
    loot.transform.SetParent(contenedor);
}

```

Ilustración 230 Manager looteo cargar en el panel

Configuré el panel para que se abra al clicar la sombra de un enemigo muerto y para que se cierre al clicar sobre la X de la parte superior derecha del mismo.

```

public void CerrarLoot()
{
    panelLoot.SetActive(false);
}

```

Ilustración 231 Manager looteo cerrar panel

Así es como se ve el panel de looteo tras clicar la sombra de un enemigo que nos ha soltado 3 pociones de vida y 5 pociones de maná. Si hay muchos objetos y no caben en el panel, se puede hacer scroll con el ratón para ver los objetos de la parte de abajo.



Ilustración 232 Interfaz looteo actualizada

Hasta ahora, podemos asignar looteo a los enemigos y podemos ver los objetos que nos sueltan; pero, aún faltan algunos scripts que permitan al protagonista recoger esos objetos y obtener la experiencia asignada a los enemigos.

Programé un script para que, mientras no haya recogido el objeto del panel de loot, este objeto se pueda clicar haciendo que desaparezca del panel y se instancie en el inventario del protagonista.

```
public void RecogerObjeto()
{
    if (ItemPorRecoger == null)
    {
        return;
    }
    Inventario.Instance.AddItem(ItemPorRecoger.Item, ItemPorRecoger.Cantidad);
    ItemPorRecoger.ItemRecogido = true;
    Destroy(gameObject);
}
```

Ilustración 233 Script recoger objeto loot

Para recoger la experiencia, creé una respuesta al evento que se lanza cuando un enemigo muere. Esta respuesta llama al método de añadir experiencia al protagonista pasando como parámetro la experiencia que tiene asignada el enemigo a su loot.

```
private void OnEnable()
{
    EnemigoVida.EventoEnemigoMorido += RespuestaEnemigoMorido;
}

♀ Mensaje de Unity | 0 referencias
private void OnDisable()
{
    EnemigoVida.EventoEnemigoMorido -= RespuestaEnemigoMorido;
}

2 referencias
private void RespuestaEnemigoMorido(float exp)
{
    AñadirExp(exp);
}
```

Ilustración 234 Script añadir experiencia loot

El método de añadir experiencia lo creé en el script de experiencia del protagonista. Este método usa como parámetro un número que hace referencia a una cantidad de experiencia y actualiza la experiencia actual del protagonista y la experiencia que le falta para llegar al siguiente nivel. Además, si ya se ha alcanzado la experiencia necesaria para llegar al siguiente nivel, este método también actualiza el nivel del protagonista.

```

public void AñadirExp(float expObtenida)
{
    if (expObtenida > 0f)
    {
        float expRestanteSiguienteNivel = expParaSiguienteNivel - expActualTemporal;
        if (expObtenida >= expRestanteSiguienteNivel)
        {
            expObtenida -= expRestanteSiguienteNivel;
            expActual += expObtenida;
            ActualizarNivel();
            AñadirExp(expObtenida);
        }
        else
        {
            expActual += expObtenida;
            expActualTemporal += expObtenida;
            if (expActualTemporal == expParaSiguienteNivel)
            {
                ActualizarNivel();
            }
        }
    }
    stats.ExpActual = expActual;
    ActualizarBarraExp();
}

```

Ilustración 235 Script añadir experiencia

Ya está listo el sistema de looteo que nos permite obtener objetos y experiencia tras matar enemigos para hacernos más fuertes y poder avanzar en el juego.

Sistema de misiones

También quise incorporar a mi videojuego un sistema de misiones, ya que es una parte muy entretenida y útil en muchos videojuegos RPG. Este sistema de misiones permite al protagonista obtener experiencia, dinero y objetos aceptando y realizando encargos a algunos NPCs; además de la experiencia y los objetos obtenidos por matar enemigos.

En el caso de mi juego, todas las misiones se basarán en matar un cierto número de enemigos.

Para comenzar con este sistema, creé en el canvas un panel donde se mostrarán las misiones disponibles que el protagonista puede aceptar. Este panel sigue la estética del resto de la interfaz del juego y se abre tras hablar con el NPC correspondiente. Podemos distinguir los NPC que ofrecen misiones porque, al acercarnos, tienen un “?” en la cabeza. El panel se cierra en la “X” de la esquina superior derecha.

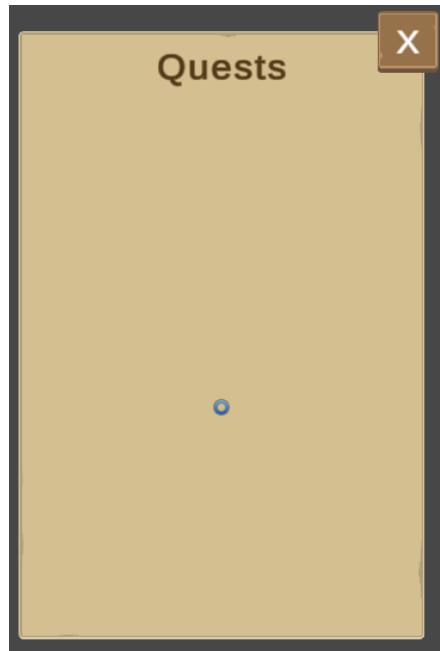


Ilustración 236 Interfaz misiones disponibles

Creé otro panel parecido, también en el canvas, para las misiones activas (las que ya ha aceptado el protagonista). Este panel se abre y se cierra desde el menú de la parte inferior izquierda de la pantalla, haciendo clic en el dibujo del pergamo.

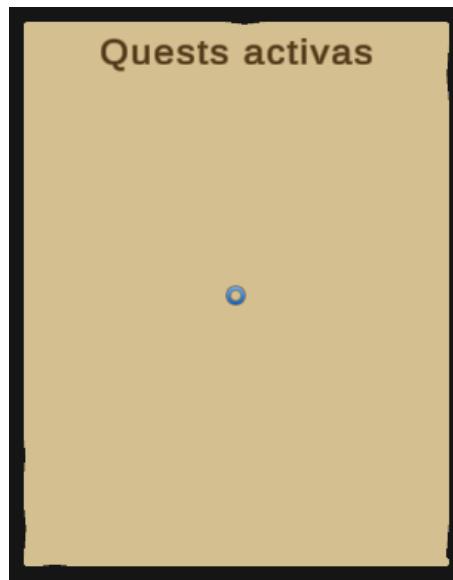


Ilustración 237 Interfaz misiones activas

Por último, en cuanto a la interfaz, creé un pequeño panel que se activará al completar una misión y mostrará el nombre de la misión completada y los objetos, experiencia y dinero que hemos conseguido. Se cierra usando el botón para reclamar la recompensa.

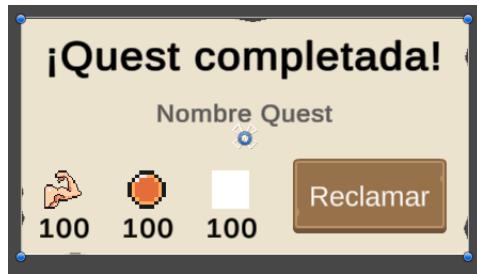


Ilustración 238 Interfaz misión completada

Para empezar a programar el sistema, hice un script de tipo objeto encriptable para las misiones, de forma que fuera sencillo crear estos objetos misiones más adelante desde el proyecto y editarlas a mi gusto. Estos objetos misiones tienen información de un nombre, un identificador, una cantidad (de enemigos que hay que matar para completarla), una descripción, y la recompensa. La recompensa cuenta con variables para el dinero, la experiencia y el objeto que recibimos al terminarla. Además, estos objetos tienen una variable con la cantidad actual de enemigos que hemos matado y un boolean que comprueba si se ha completado o no.

Además de la información de cada misión, el script de misiones cuenta con métodos para actualizar la cantidad actual de enemigos matados, comprobar si se ha terminado y lanzar un evento cuando se ha completado para permitirnos obtener la recompensa.

```
public void AddProgreso(int cantidad)
{
    CantidadActual += cantidad;
    VerificarCompletada();
}

1 referencia
private void VerificarCompletada()
{
    if (CantidadActual >= Cantidad)
    {
        CantidadActual = Cantidad;
        QuestCompletada();
    }
}

1 referencia
private void QuestCompletada()
{
    if (Completada)
    {
        return;
    }
    Completada = true;
    EventoQuestCompletada?.Invoke(obj: this);
}
```

Ilustración 239 Script misiones

Así se ve desde el inspector el objeto encriptable de misión en el que podemos editar sus campos:

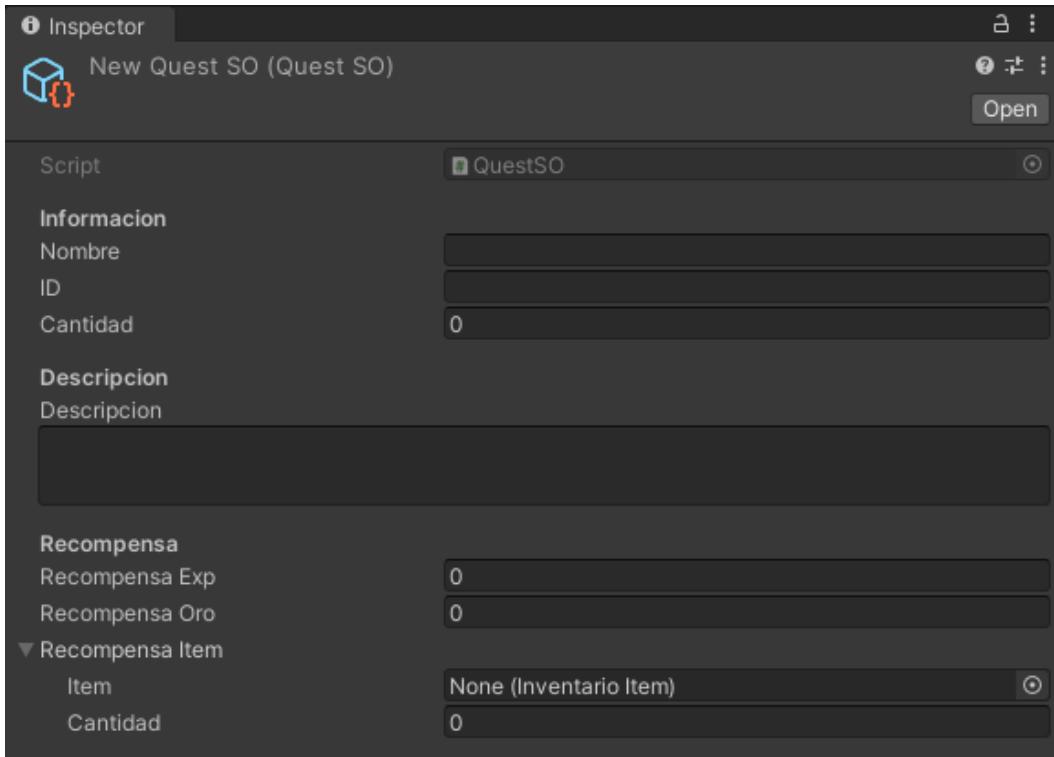


Ilustración 240 Inspector misión

Creé un manager para gestionar las misiones con los métodos para que sus datos se muestren correctamente en la interfaz y estén siempre actualizados.

Este manager nos permite cargar las misiones creadas al panel de misiones, instanciándolas dentro de contenedores en este panel.

```
private void CargarQuest()
{
    for (int i = 0; i < questDisponibles.Length; i++)
    {
        MenuQuestDescripcion nuevoQuest = Instantiate(questPrefab, questContenedor);
        nuevoQuest.ConfigurarQuestUI(questDisponibles[i]);
    }
}
```

Ilustración 241 Script manager cargar misiones

Otro método se encarga de que todos los datos de las misiones que se muestran en la interfaz son los correspondientes a la misión que se muestra.

```
public virtual void ConfigurarQuestUI(QuestSO quest)
{
    QuestCargada = quest;
    questNombre.text = quest.Nombre;
    questDescripcion.text = quest.Descripcion;
}
```

Ilustración 242 Script interfaz misiones

Y otro método se encarga de formatear los datos de las misiones que se muestran, de forma que la recompensa aparezca en forma de lista.

```

public class MenuQuestDescripcion : QuestDescripcion
{
    [SerializeField] private TextMeshProUGUI questRecompensa;

    4 referencias
    public override void ConfigurarQuestUI(QuestSO quest)
    {
        base.ConfigurarQuestUI(quest);
        questRecompensa.text = $"- {quest.RecompensaExp} exp"
            + $"\\n- {quest.RecompensaOro} oro"
            + $"\\n- {quest.RecompensaItem.Item.Nombre} x{quest.RecompensaItem.Cantidad}";
    }
}

```

Ilustración 243 Script descripción misiones

Creé en la jerarquía un objeto vacío con el nombre de QuestManager y le añadí el script del manager creado para gestionar las misiones. Este script hace referencia al personaje que mata a los enemigos para completar las misiones y recibe la recompensa y también a las misiones creadas en forma de objetos encriptables que el protagonista puede completar.

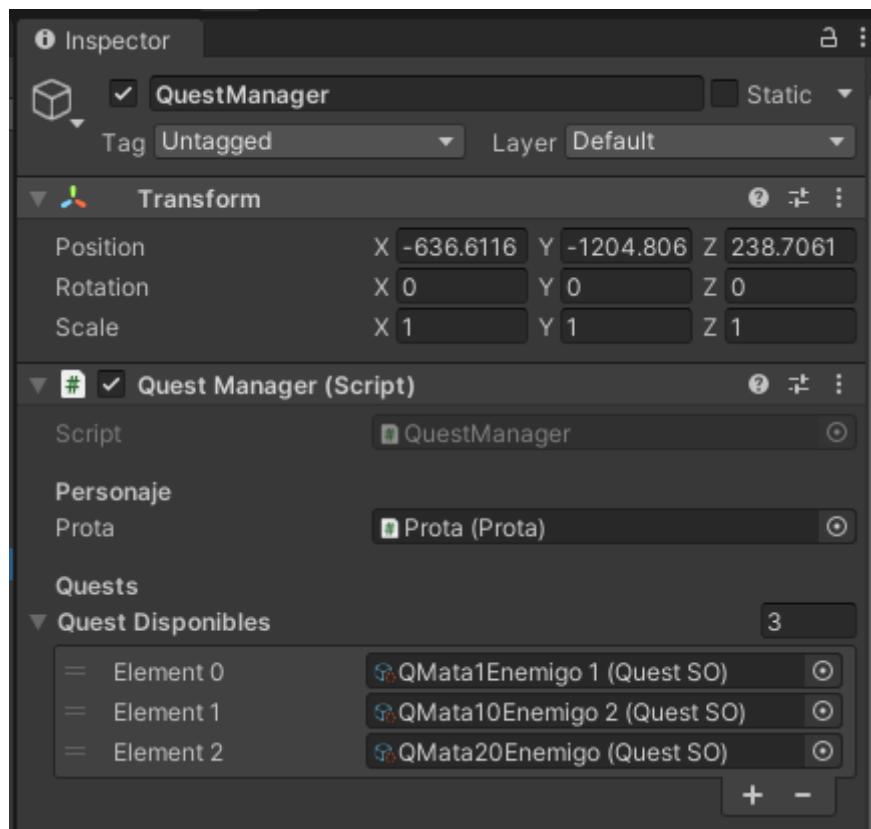


Ilustración 244 Inspector manager misiones

Así es como se muestra en el juego el panel de misiones cuando hay misiones disponibles:



Ilustración 245 Interfaz misiones actualizada

Para poder aceptar las misiones, podemos pulsar el botón aceptar en cada una de ellas. El botón aceptar lanza un método que quita las misiones del panel de misiones disponibles y las instancia en las misiones activas (aceptadas). Es entonces cuando el objeto misión se activa y podemos completarla.

```
public void AceptarQuest()
{
    if (QuestCargda == null)
    {
        return;
    }
    QuestManager.Instance.AddQuest(QuestCargda);
    gameObject.SetActive(false);
}
```

Ilustración 246 Script aceptar misiones

También necesité un script para actualizar la interfaz del panel de misiones activas, mostrando tanto las recompensas como el desarrollo actual de cada una de las misiones.

```

public override void ConfigurarQuestUI(QuestSO quest)
{
    base.ConfigurarQuestUI(quest);
    recompensaExp.text = quest.RecompensaExp.ToString();
    recompensaOro.text = quest.RecompensaOro.ToString();
    objetivo.text = $"{quest.CantidadActual}/{quest.Cantidad}";
    recompensaItemIcono.sprite = quest.RecompensaItem.Item.Imagen;
    recompensaItemCantidad.text = quest.RecompensaItem.Cantidad.ToString();
}

```

Ilustración 247 Script interfaz misiones activas

Así se ve en el juego la interfaz de las misiones activas cuando ya hemos aceptado algunas misiones:



Ilustración 248 Interfaz misiones activas actualizada

Como vimos anteriormente, en el objeto encriptable de misión se comprueba si hemos completado la misión y se lanza un evento cuando se ha completado. En el manager de misiones creé el método de respuesta a este evento para mostrar en pantalla el panel que avisa de que se ha completado la misión y permite reclamar las recompensas.

```

private void RespuestaQuestCompletada(QuestSO quest)
{
    QuestPorReclamar = Ref(quest.ID);
    if (quest != null)
    {
        MostrarQuestCompletada(QuestPorReclamar);
    }
}

↳ Mensaje de Unity | 0 referencias
private void OnEnable()
{
    QuestSO.EventoQuestCompletada += RespuestaQuestCompletada;
}

↳ Mensaje de Unity | 0 referencias
private void OnDisable()
{
    QuestSO.EventoQuestCompletada -= RespuestaQuestCompletada;
}

```

Ilustración 249 Script manager evento respuesta misión completada

```

private void RespuestaQuestCompletada(QuestSO questCompletada)
{
    if (questCompletada.ID == QuestCargda.ID)
    {
        objetivo.text = $"{QuestCargda.CantidadActual}/{QuestCargda.Cantidad}";
        gameObject.SetActive(false);
    }
}

```

Ilustración 250 Script respuesta misión completa

También fue necesario actualizar la interfaz del panel de misión completa. Para ello, creé un método que carga en este panel el nombre de la misión terminada, la experiencia, el dinero y el objeto de recompensa. Así es como se muestra en el juego este panel al completar una misión:



Ilustración 251 Interfaz misión completada actualizada

Para que funcione el sistema de misiones, es importante que el protagonista pueda conseguir las recompensas tras terminarlas y hacer clic en el botón para reclamar. Para ello, creé un método en el manager de misiones que añade el oro de recompensa al dinero total del jugador, la experiencia a la experiencia total del protagonista, e instancia el objeto (junto a su cantidad especificada) de recompensa en el inventario. Tras esto, desactiva la misión.

Este método se llama desde el botón reclamar del panel de misión completada.

```
public void ReclamarRecompensa()
{
    if (QuestPorReclamar == null)
    {
        return;
    }
    OroManager.Instance.AddOro(QuestPorReclamar.RecompensaOro);
    prota.ProtaExp.AñadirExp(QuestPorReclamar.RecompensaExp);
    Inventario.Instance.AddItem(QuestPorReclamar.RecompensaItem.Item,
        QuestPorReclamar.RecompensaItem.Cantidad);
    menuQuestCompletado.SetActive(false);
    QuestPorReclamar = null;
}
```

Ilustración 252 Script manager reclamar recompensa

Con todo el sistema de misiones programado, creé todas las misiones que aparecerían en mi videojuego. Para crearlas y tener todo organizado, creé una carpeta para las misiones dentro de objetos encriptables en assets. Como creé las misiones como objetos encriptables, pude crear cada una de ellas directamente desde esta carpeta.



Ilustración 253 Assets misiones

Así se ve desde el inspector el objeto encriptable de misión:

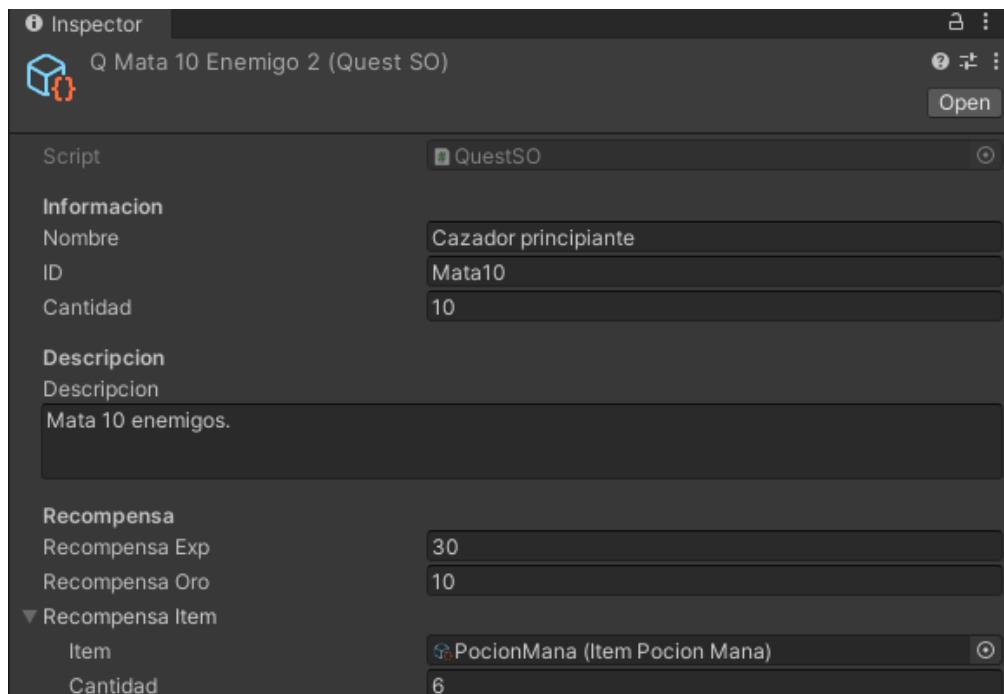


Ilustración 254 Inspector misión

Por último, creé y asigné diálogos con interacción extra de tipo misión. De esta forma el protagonista puede acceder al panel de misiones para aceptarlas hablando con los NPCs que tengan este diálogo asignado. Asigné el diálogo a un NPC en cada aldea para poder aceptar las misiones en distintos momentos del juego.

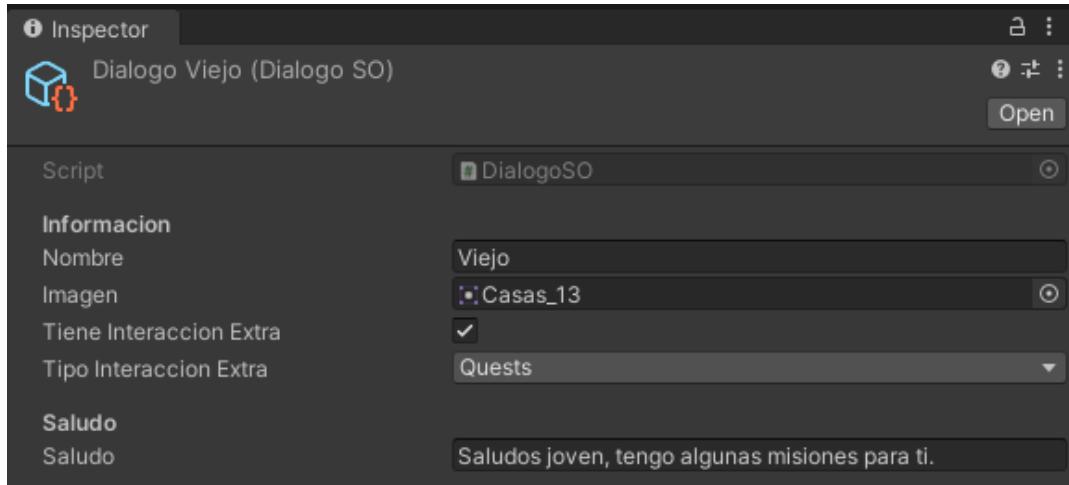


Ilustración 255 Diálogo misiones

Sistema de tienda

Añadí un sistema de tienda para que el protagonista pudiera adquirir objetos a cambio del dinero que consigue matando enemigos y completando misiones. La tienda permitirá al protagonista obtener objetos que le facilitarán avanzar por la historia, como pociones o armas.

Para empezar, creé la interfaz de la tienda en el canvas.

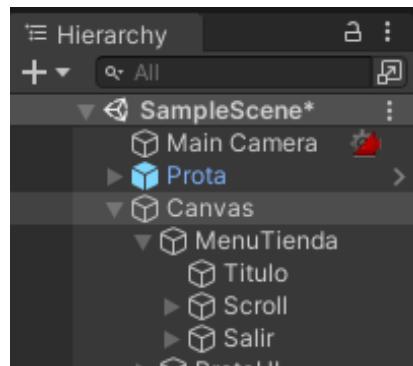


Ilustración 256 Jerarquía interfaz tienda

Esta interfaz es simple y sigue con la estética del resto del juego. Es un panel en el que aparecerán contenedores con cada uno de los objetos disponibles en la tienda junto a sus precios y los botones para comprarlos. Este panel se abre tras hablar con un NPC con un diálogo con interacción extra de tipo tienda y se cierra en la “X” de la parte superior derecha del mismo panel.

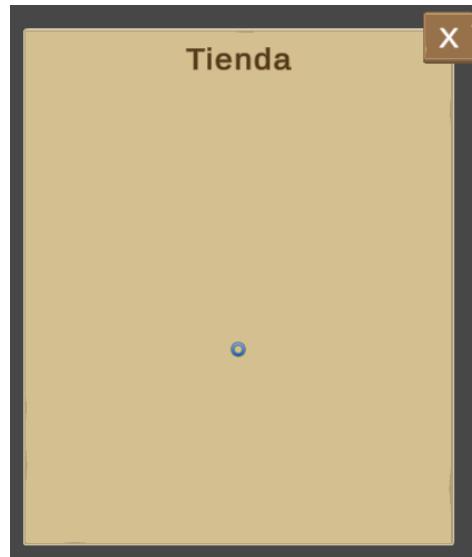


Ilustración 257 Interfaz tienda

Para gestionar la tienda y los objetos disponibles en ella, creé un script manager de tienda. Este script hace referencia a una lista con objetos existentes en el juego en la que añadiremos los objetos que queremos que estén a la venta. También hace referencia a los contenedores en los que se mostrará individualmente cada objeto dentro del panel de tienda.

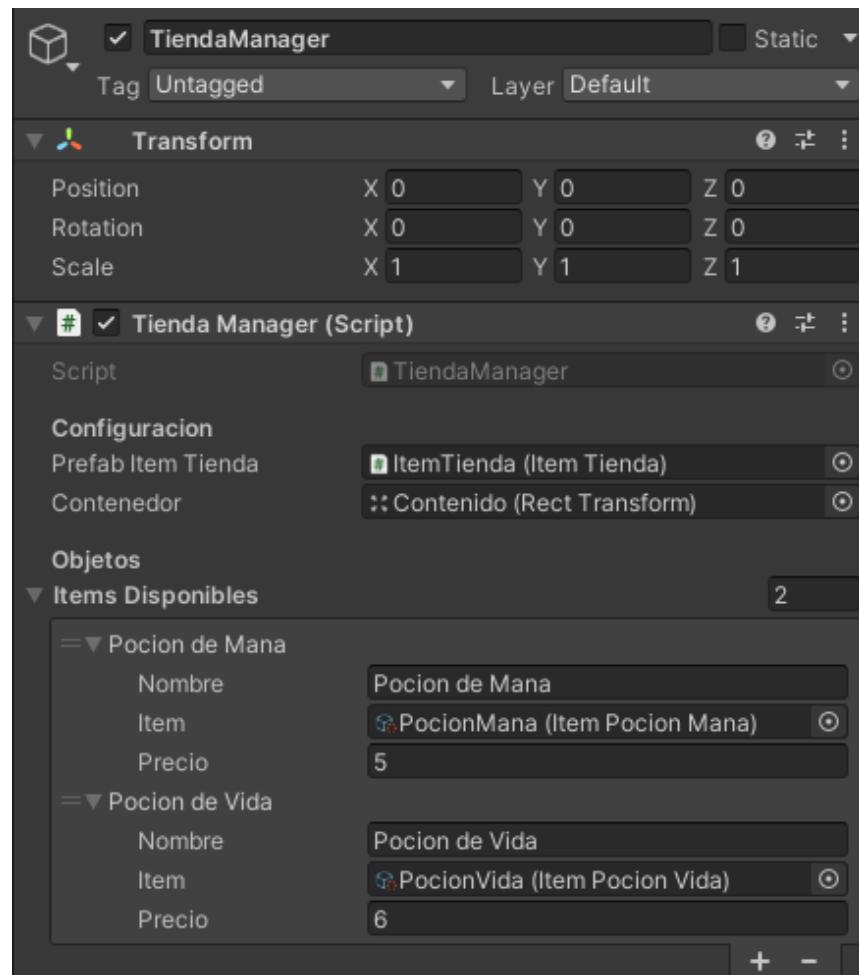


Ilustración 258 Inspector manager tienda

El en script manager de la tienda creé un método para añadir a esta los objetos que se podrán comprar. Para ello, se instancian los objetos dentro de contenedores que se añaden dentro del panel de la tienda.

```
private void CargarItemALaTienda()
{
    for (int i = 0; i < itemsDisponibles.Length; i++)
    {
        ItemTienda itemTienda = Instantiate(prefabItemTienda, contenedor);
        itemTienda.ActualizarItemVenta(itemsDisponibles[i]);
    }
}
```

Ilustración 259 Script añadir objetos a la tienda

Los objetos formarán parte de la tienda serán objetos ya existentes en el juego, por lo que tienen imagen y nombre. También cuentan con atributos adicionales como el precio y la cantidad, con los que se puede calcular el precio inicial de un objeto y el precio actual al añadir más o menos cantidad de ese objeto a la compra.

Traes crear los objetos de la tienda, creé el método que permite comprarlos, quitando de nuestra cantidad total de dinero el precio del producto que compramos e instanciando este último en nuestro inventario. Por defecto, la cantidad de compra es de una unidad.

```
public void ComprarItem()
{
    if (OroManager.Instance.OroTotal >= precioActual)
    {
        Inventario.Instance.AddItem(ItemCargado.Item, cantidad);
        OroManager.Instance.QuitarOro(precioActual);
        cantidad = 1;
        precioActual = precioInicial;
    }
}
```

Ilustración 260 Script tienda comprar objeto

Además,uento con un método para editar la cantidad de unidades al comprar un objeto. Nos permite sumar unidades a la cantidad total de compra mientras el precio total sea inferior que el dinero que tenemos. El precio total se calcula sumando la cantidad de unidades por el precio de cada unidad. También nos permite restar unidades a la cantidad total y vuelve a calcular el precio total.

```

public void SumarItem()
{
    int precioDeCompra = precioInicial * (cantidad + 1);
    if (OroManager.Instance.OroTotal >= precioDeCompra)
    {
        cantidad++;
        precioActual = precioInicial * cantidad;
    }
}

0 referencias
public void RestarItem()
{
    if (cantidad == 1)
    {
        return;
    }
    cantidad--;
    precioActual = precioInicial * cantidad;
}

```

Ilustración 261 Script tienda sumar y restar cantidad

Una vez que los objetos están creados y asignados a la tienda, un método se encarga de mostrar en la interfaz los datos correspondientes de cada objeto: el propio objeto en sí, su imagen, su nombre y su precio.

```

public void ActualizarItemVenta(ItemVenta item)
{
    ItemCargado = item;
    itemImagen.sprite = item.Item.Imagen;
    itemNombre.text = item.Item.Nombre;
    itemPrecio.text = item.Precio.ToString();

    cantidad = 1;
    precioInicial = item.Precio;
    precioActual = item.Precio;
}

```

Ilustración 262 Script interfaz tienda

También es necesario un método para abrir y cerrar el panel de la tienda, que se llama tras hablar con un NPC con una conversación con interacción extra del tipo tienda, y se vuelve a llamar para cerrar este panel al pulsar en la “X” de la esquina superior derecha del mismo.

```

public void AbrirCerrarPanelTienda()
{
    panelTienda.SetActive(!panelTienda.activeSelf);
}

```

Ilustración 263 Script interfaz abrir y cerrar tienda

Así es como muestra el juego la tienda con objetos cargados y la interfaz actualizada:



Ilustración 264 Interfaz tienda actualizada

Por último, para permitir al jugador acceder a la tienda, creé los diálogos con interacción extra del tipo tienda y se lo asigné a un NPC en cada aldea, para poder comprar en distintos momentos de la historia del juego.

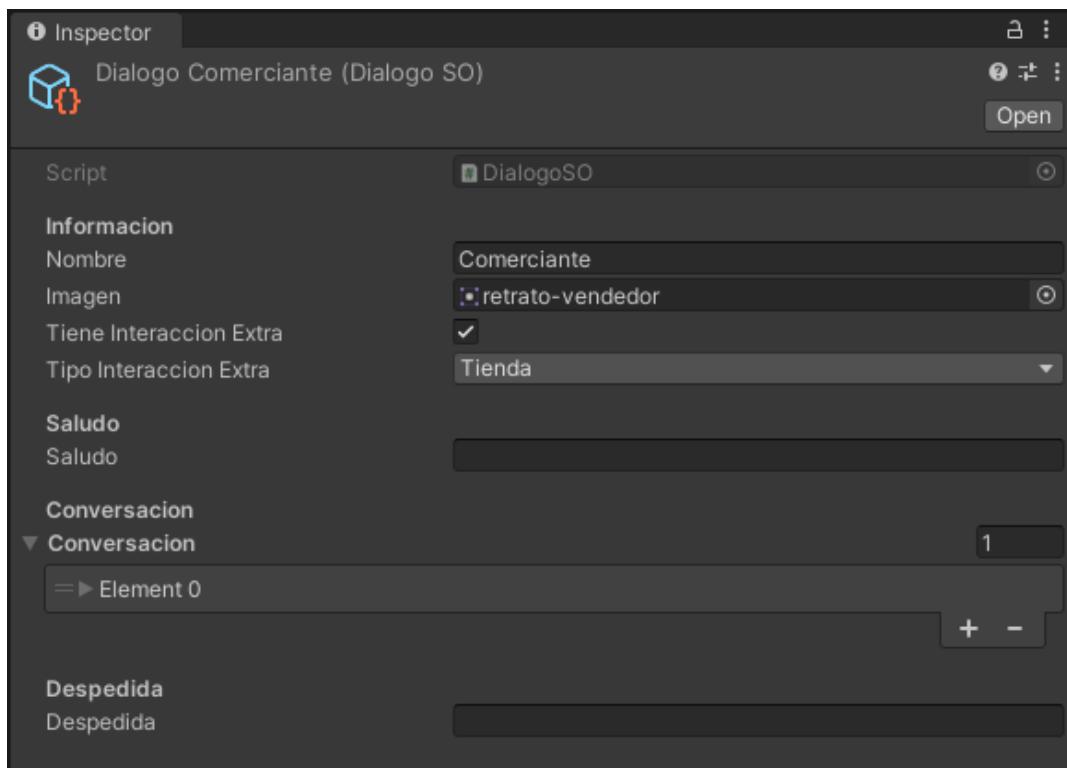


Ilustración 265 Inspector diálogo tienda

Sistema de crafteo

El último sistema que añadí a mi videojuego es el de crafteo. Este sistema está presente en algunos videojuegos de tipo RPG, pero no es tan común como los programados anteriormente. He incluido este sistema para permitir a los jugadores crear objetos con

otros que hayan recogido anteriormente, de forma que puedan conseguir pociones o armas más fuertes que les permitan avanzar más rápidamente por el juego.

Para comenzar con este sistema, creé la interfaz en el canvas.

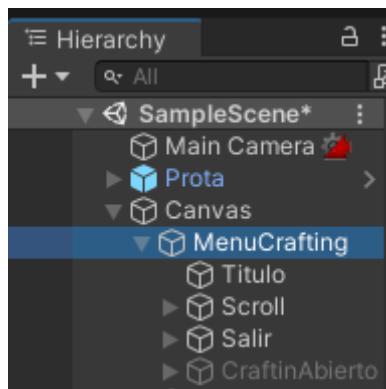


Ilustración 266 Jerarquía interfaz crafting

La interfaz del sistema de crafteo sigue la estética de todas las interfaces del juego. Este panel se divide en dos columnas: la de la izquierda contiene la imagen y el nombre de las recetas disponibles y la de la derecha contiene la información de los materiales necesarios y el resultado de la receta seleccionada, junto al botón para crearla.

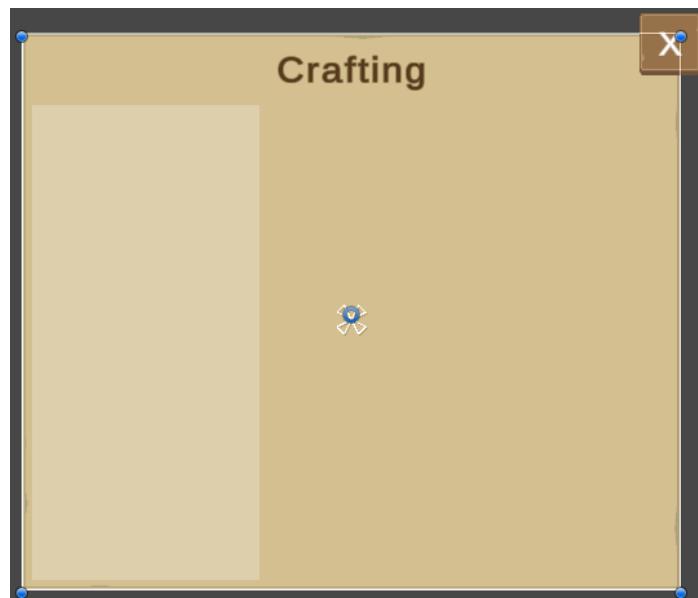


Ilustración 267 Interfaz crafting

Para comenzar con la programación del sistema de crafteo, hice un script de tipo objeto encriptable para las recetas. Estas recetas pertenecen a una lista que se puede crear desde el proyecto y llenar con todas las recetas que queramos. Cada receta contiene los atributos correspondientes al objeto resultante junto a la cantidad y a los dos objetos necesarios para crearlas con sus cantidades correspondientes. También hace referencia al nombre de la receta.

Desde el inspector podemos modificar la lista de recetas y se ve de la siguiente forma:

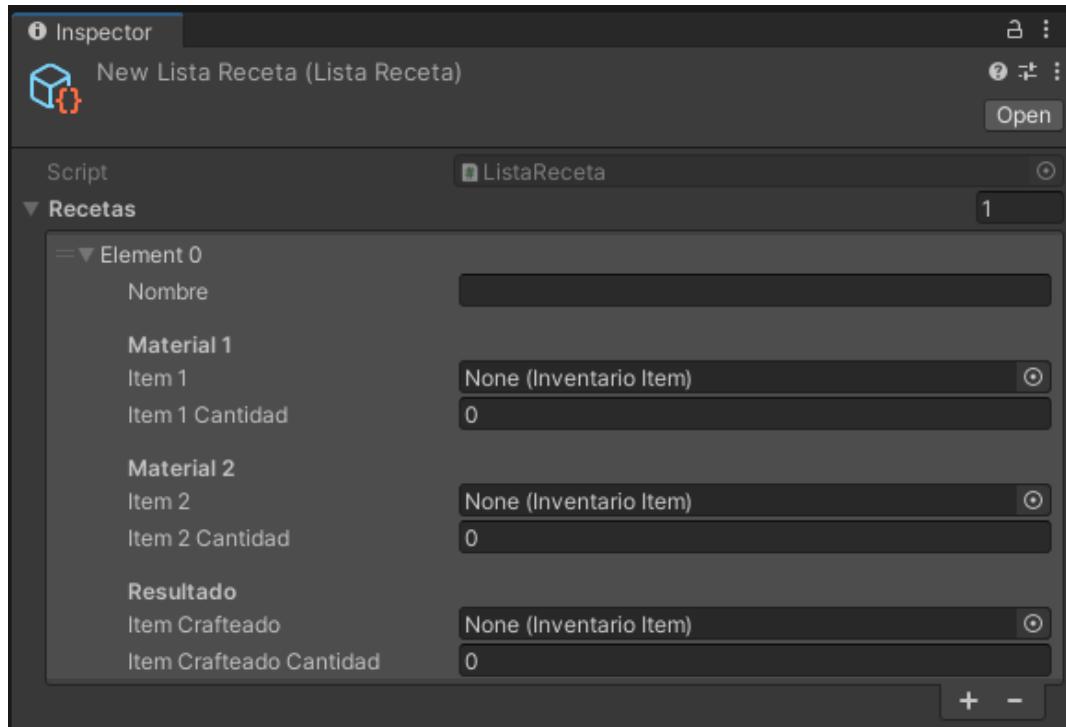


Ilustración 268 Inspector lista de recetas

Más tarde, creé un script manager para gestionar el sistema de crafteo. Este manager tiene referencia de lista de receta y de los objetos necesarios para cada crear cada receta y del objeto resultante de cada una. También tiene referencia al contenedor en el que se muestra cada receta dentro del panel de crafteo y un método que permite cargar cada una de las recetas en su contenedor correspondiente.

```
private void CargarRecetas()
{
    for (int i = 0; i < recetas.Recetas.Length; i++)
    {
        TarjetaReceta receta = Instantiate(prefabTarjetaReceta, contenedor);
        receta.ActualizarTarjetaReceta(recetas.Recetas[i]);
    }
}
```

Ilustración 269 Script manager crafteo cargar recetas

Para usar este manager, creé un objeto vacío en la jerarquía y le añadí el script. En el inspector se ve así:

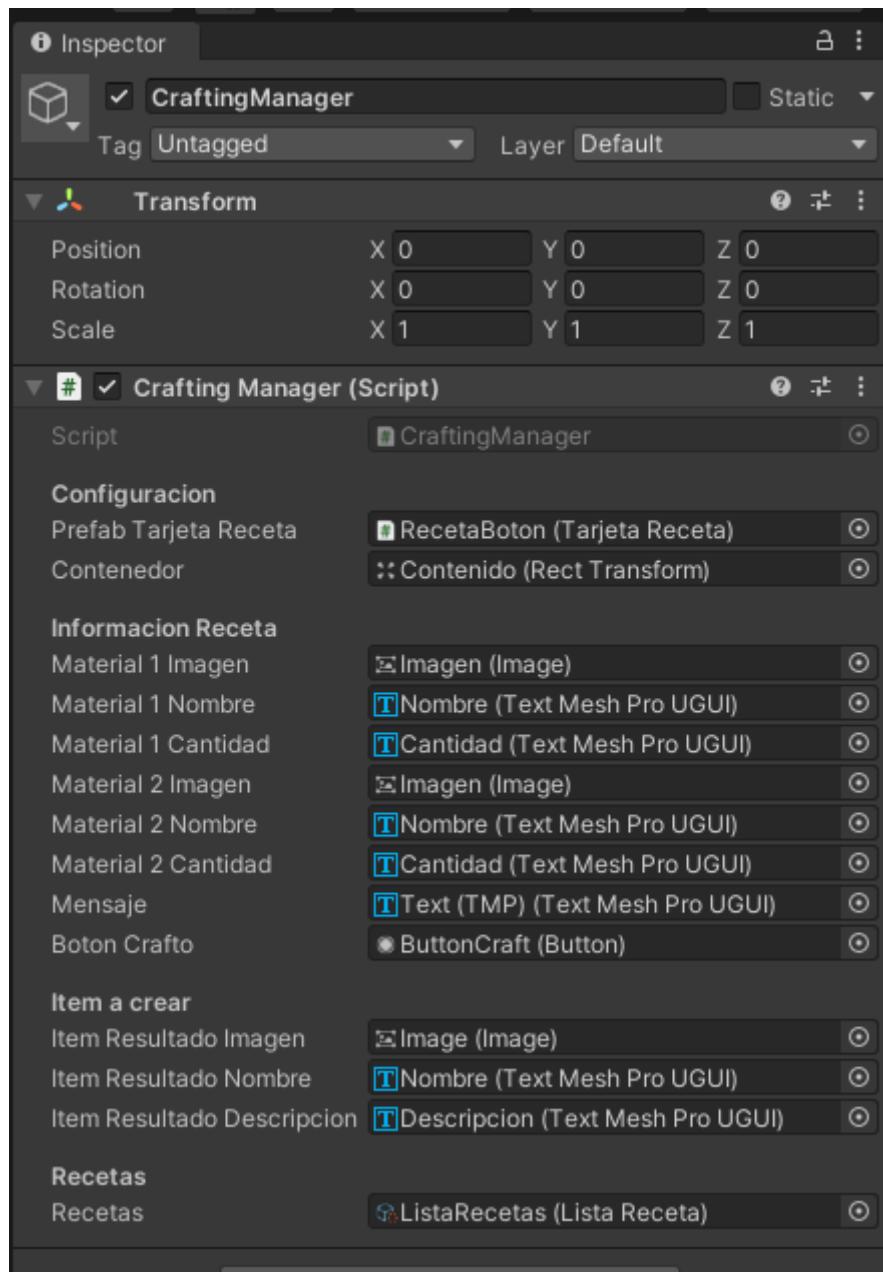


Ilustración 270 Inspector manager crafteo

Para poder craftear una receta es necesario tener todos los ingredientes en el inventario, por lo que creé un método que comprueba los materiales necesarios de cada receta y sus cantidades y los compara con los objetos de nuestro inventario. Si tenemos todos los ingredientes devuelve verdadero y nos permite craftear.

```

public bool Crafteable(Receta receta)
{
    if (Inventario.Instance.ObtenerCantidadItem(receta.Item1.ID) >=
        receta.Item1Cantidad && Inventario.Instance.ObtenerCantidadItem(receta.Item2.ID)
        >= receta.Item2Cantidad)
    {
        return true;
    }
    return false;
}

```

Ilustración 271 Script manager crafting comprobar materiales

También creé un método en el manager que permite craftear la receta. Para ello quita del inventario los objetos necesarios para crear la receta y también coloca en el inventario el objeto resultante.

```

public void Craftear()
{
    for (int i = 0; i < RecetaSeleccionada.Item1Cantidad; i++)
    {
        Inventario.Instance.ConsumirItem(RecetaSeleccionada.Item1.ID);
    }
    for (int i = 0; i < RecetaSeleccionada.Item2Cantidad; i++)
    {
        Inventario.Instance.ConsumirItem(RecetaSeleccionada.Item2.ID);
    }
    Inventario.Instance.AddItem(RecetaSeleccionada.ItemCrafteado,
        RecetaSeleccionada.ItemCrafteadoCantidad);
    MostrarReceta(RecetaSeleccionada);
}

```

Ilustración 272 Script manager crafting crear

También fue necesario actualizar la interfaz del panel de crafteo. Además de mostrar el nombre de la receta y los datos de los ingredientes, la interfaz comprueba si la receta es crafteable (si tenemos los objetos necesarios) y muestra un mensaje que dice si podemos crearlo o no. En el caso de que una receta sea crafteable, también aparece el botón que llama al método craftear y permite crearla.

```

public void MostrarReceta(Receta receta)
{
    RecetaSeleccionada = receta;
    material1Imagen.sprite = receta.Item1.Imagen;
    material2Imagen.sprite = receta.Item2.Imagen;
    material1Nombre.text = receta.Item1.Nombre;
    material2Nombre.text = receta.Item2.Nombre;
    material1Cantidad.text = $"{Inventario.Instance.ObtenerCantidadItem(receta.Item1.ID)}" +
        $"{receta.Item1Cantidad}";
    material2Cantidad.text = $"{Inventario.Instance.ObtenerCantidadItem(receta.Item2.ID)}" +
        $"{receta.Item2Cantidad}";

    if (Crafteable(receta))
    {
        mensaje.text = "Receta disponible";
        botonCrafto.interactable = true;
    }
    else
    {
        mensaje.text = "No tienes los materiales necesarios";
        botonCrafto.interactable = false;
    }
    itemResultadoImagen.sprite = receta.ItemCrafteado.Imagen;
    itemResultadoNombre.text = receta.ItemCrafteado.Nombre;
    itemResultadoDescripcion.text = receta.ItemCrafteado.DescripcionItemCraft();
}

```

Ilustración 273 Script interfaz crafting

Así es como muestra el juego la interfaz de crafting cuando hay recetas cargadas (en este caso la receta no es crafteable):



Ilustración 274 Interfaz crafting actualizada

A continuación, creé desde el proyecto la lista de recetas que estaría disponible en el videojuego y rellené los datos de cada una de estas recetas. Este es un ejemplo de una de las recetas del juego vista desde el inspector:

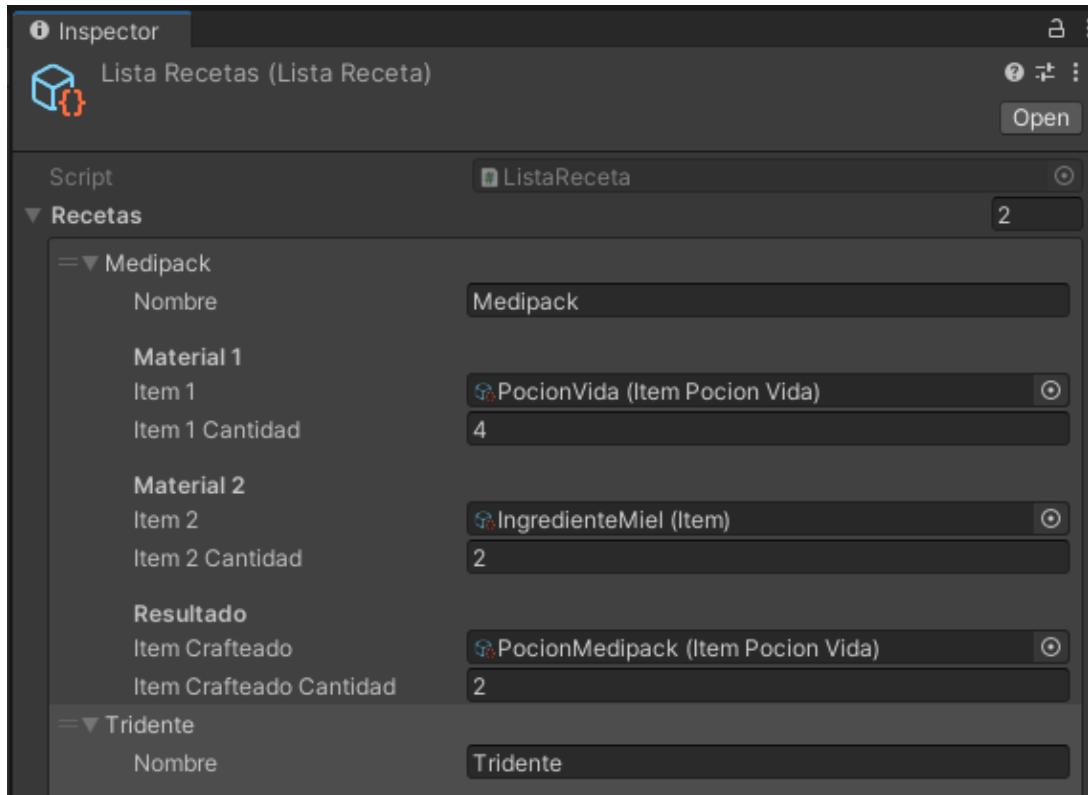


Ilustración 275 Inspector lista de recetas

Para permitir al jugador crear recetas, primero deben acceder al panel de crafting. Para ello, creé los diálogos con interacción extra del tipo crafting y se la asigné a NPCs en distintas aldeas del juego.

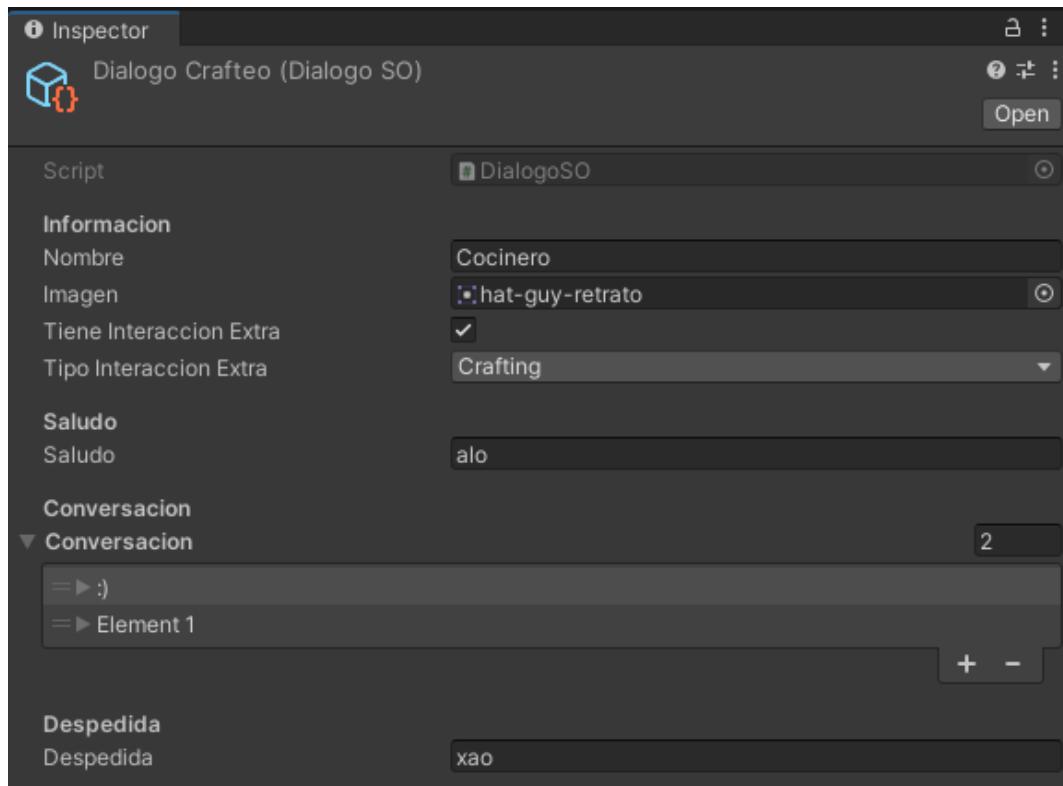


Ilustración 276 Inspector diálogo crafting

Cuando el protagonista habla con uno de los NPCs con diálogo con interacción extra de tipo crafting, se abre el panel de crafteo. Para cerrarlo, se pulsa la “X” de la parte superior derecha del mismo panel.

Por último, creé los edificios donde están los NPCs que nos permiten craftear, simulando una herrería.

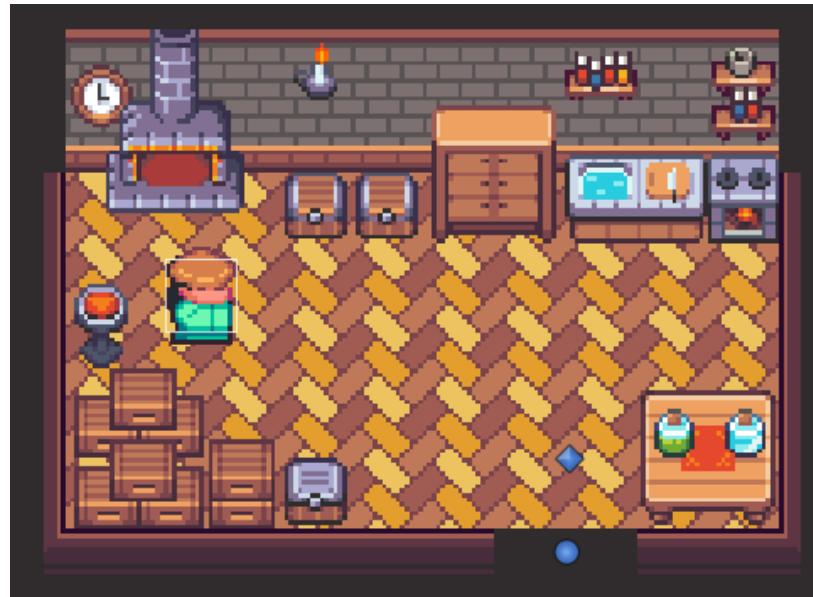


Ilustración 277 Edificio crafting

Flujo y menús

Añadí al juego menús para inicio, pausa y configuración, todos manteniendo la estética del juego: colores marrones y aspecto rural, antiguo. En cada uno de los menús aparece el nombre de este y botones que nos permiten navegar entre ellos y el propio juego. También creé un botón para poder acceder al menú de pausa desde dentro del juego.

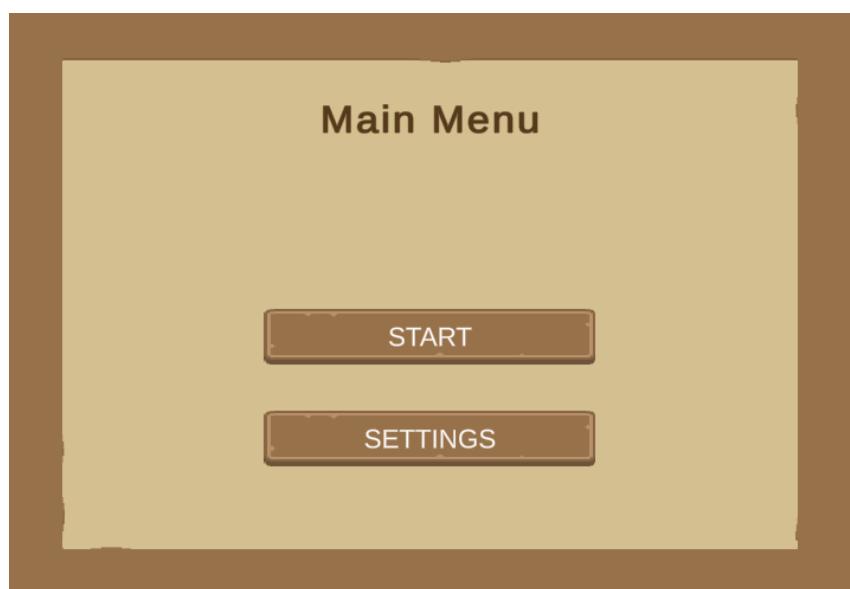


Ilustración 278 Interfaz menú inicio

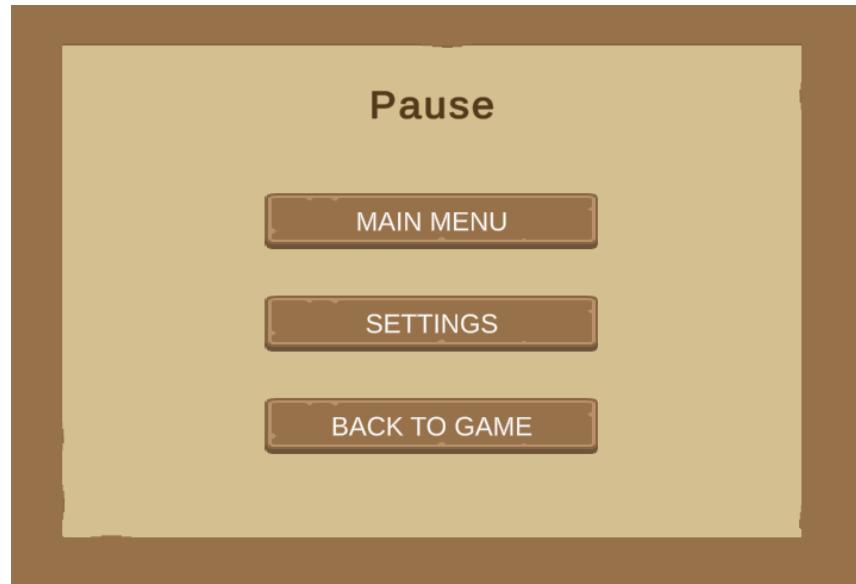


Ilustración 279 Interfaz menú pausa



Ilustración 280 Botón pausa

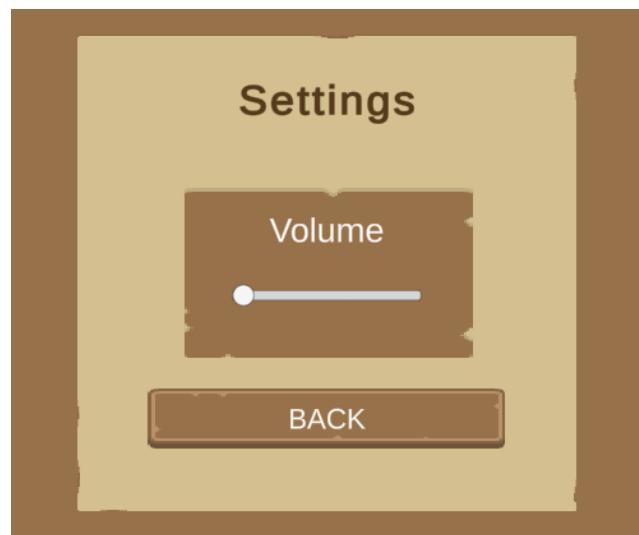


Ilustración 281 Interfaz configuración

La pantalla de fin la creé usando los propios elementos del mapa.

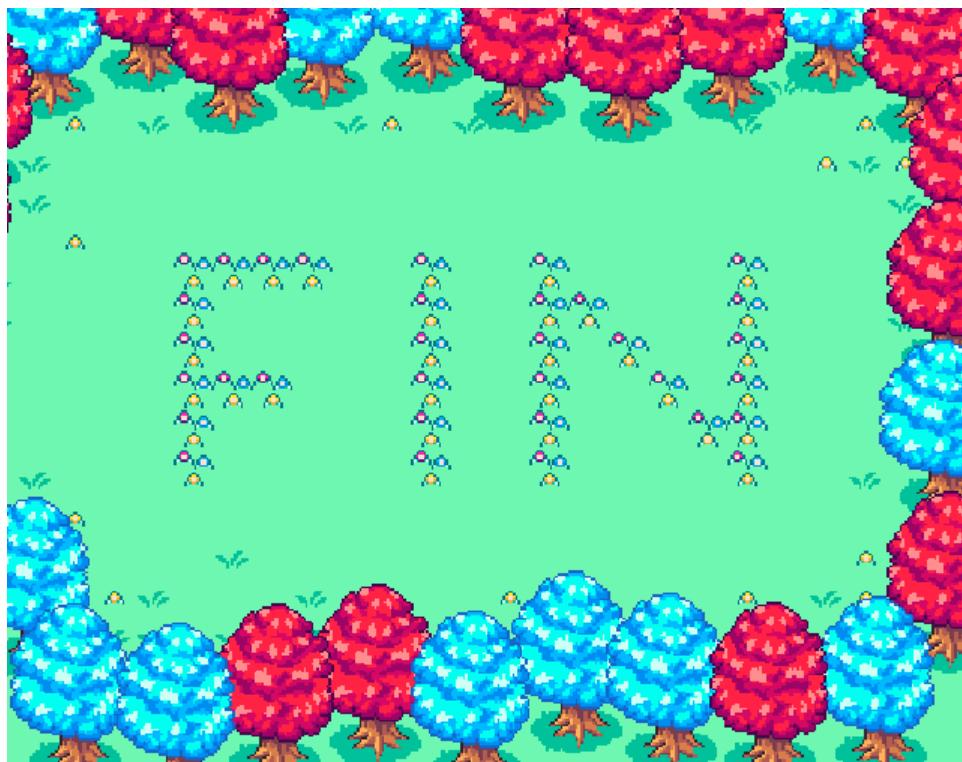


Ilustración 282 Pantalla de fin

Para poder navegar entre los menús y el juego, configuré cada uno de los botones de los distintos menús para que activen y desactiven distintos objetos.

El botón “start” del menú de inicio desactiva el menú inicio y activa la interfaz del juego.

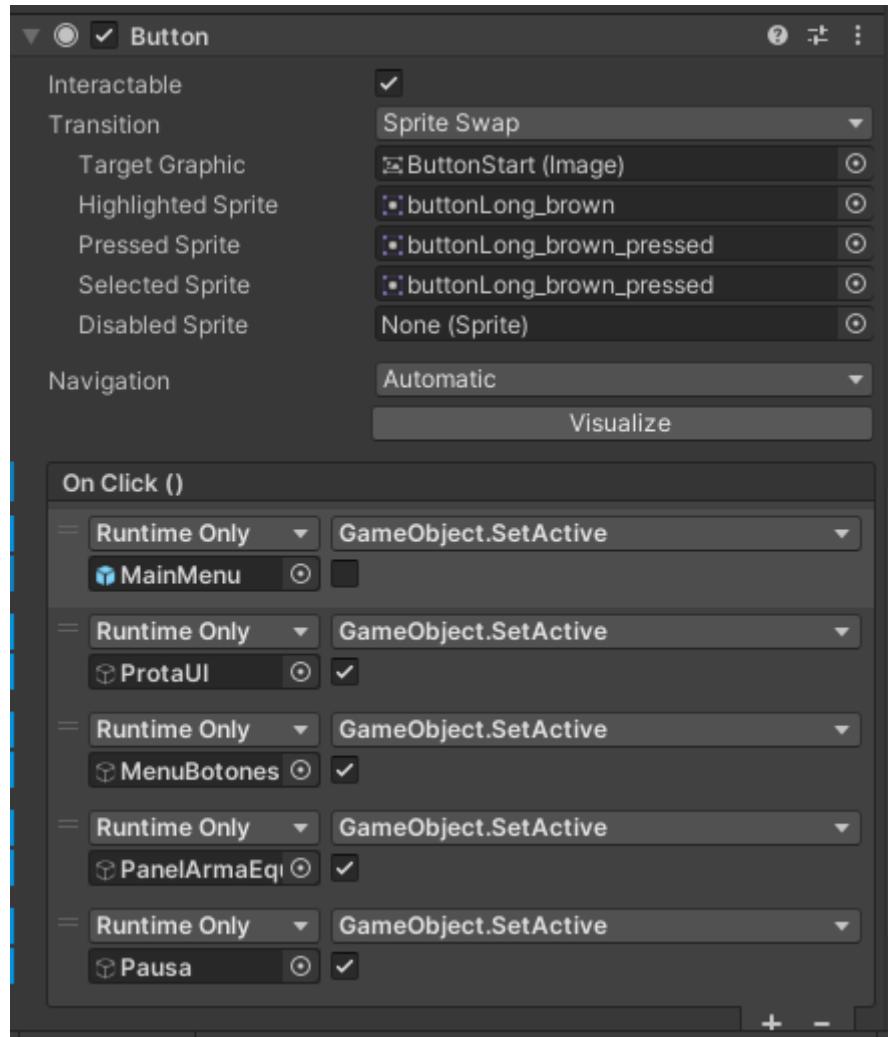


Ilustración 283 Inspector menú inicio botón start

El botón “settings” del menú inicio desactiva el menú inicio y activa el menú de configuración.

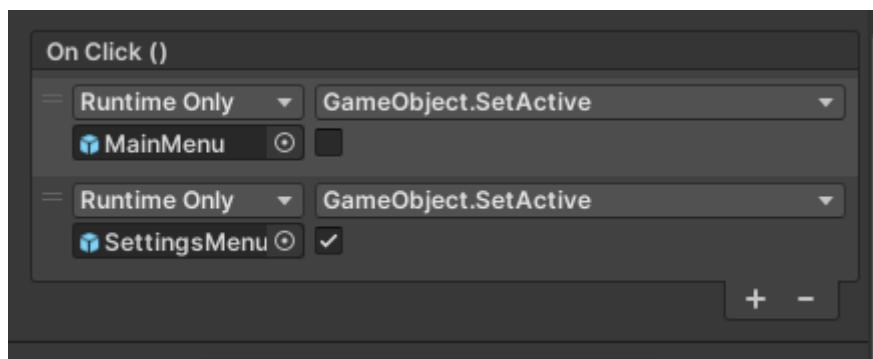


Ilustración 284 Inspector menú inicio botón settings

El botón “back to game” del menú de pausa desactiva el menú de pausa y activa la interfaz del juego.

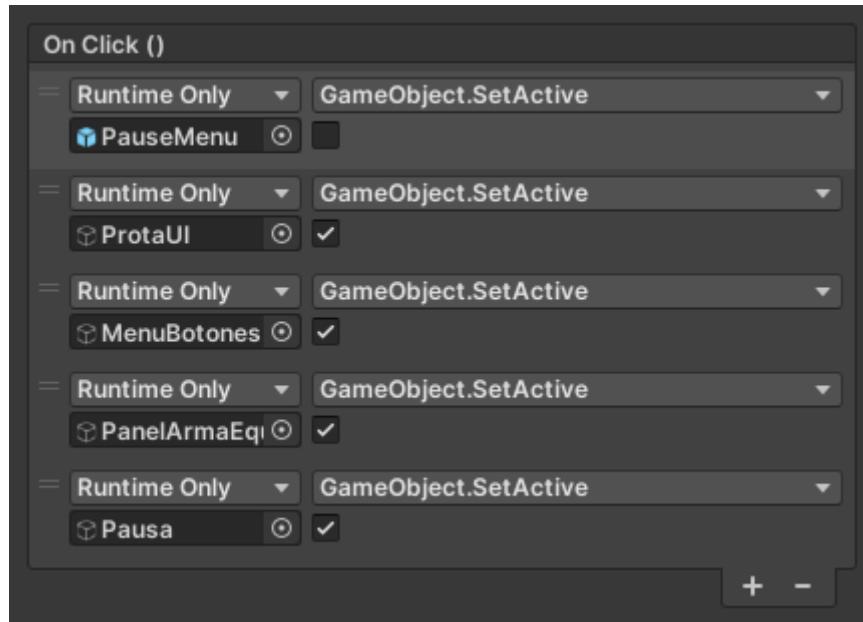


Ilustración 285 Inspector menú pausa botón back to game

El botón “settings” del menú de pausa desactiva el menú de pausa y activa el menú de configuración.

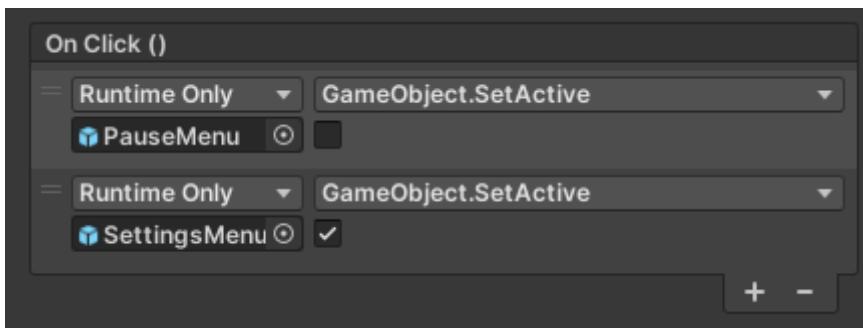


Ilustración 286 Inspector menú pausa botón settings

El botón “main menu” del menú de pausa desactiva el menú de pausa y activa el menú de inicio.

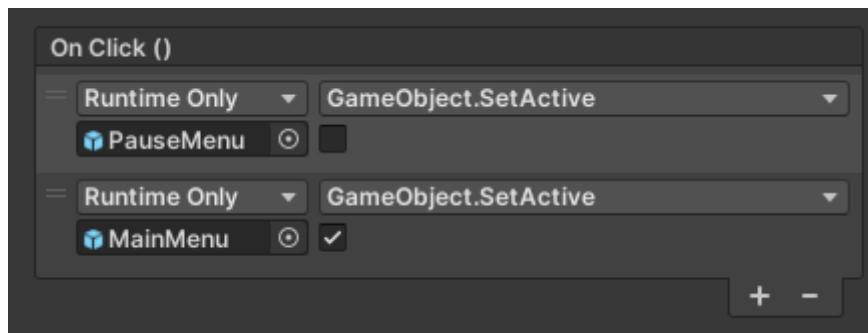


Ilustración 287 Inspector menú pausa botón main menu

El botón de pausa que aparece en la parte inferior derecha de la pantalla durante el juego activa el menú de pausa y desactiva la interfaz del juego.

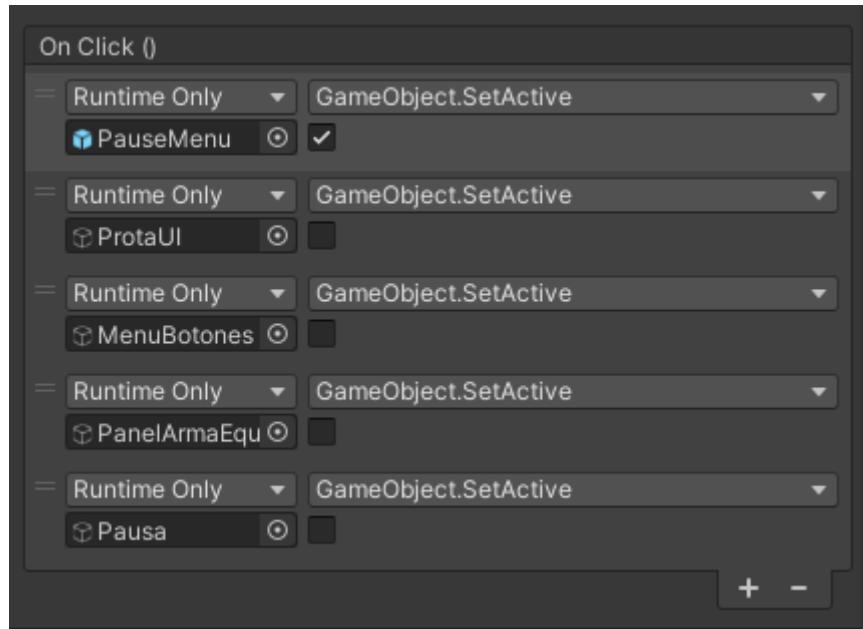


Ilustración 288 Inspector botón pausa

El botón “back” del menú de configuración desactiva el botón de configuración y activa el menú de pausa.

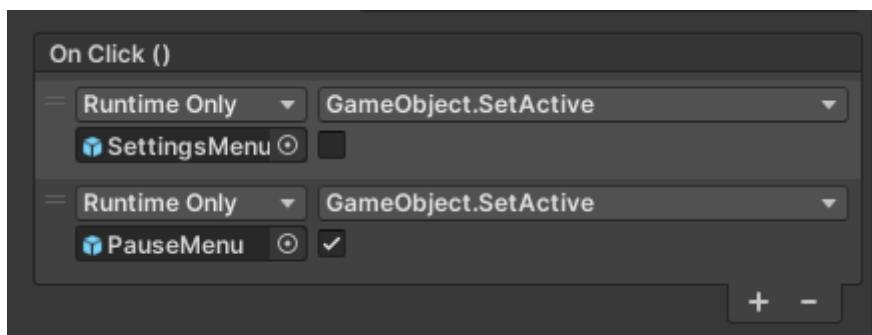


Ilustración 289 Inspector menú de configuración botón back

Música

El último toque para terminar el juego fue añadirle la música. Para ello usé un audio mp3 gratuito de internet.

Dentro de assets creé un audiomixer y una carpeta con el audio que usaría más adelante en el videojuego.

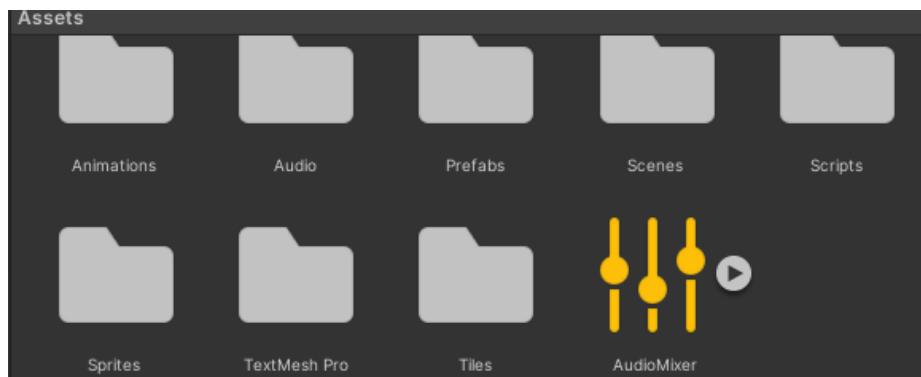


Ilustración 290 AudioMixer y carpeta de audio

Y desde la jerarquía creé un objeto vacío y lo llamé AudioManager.

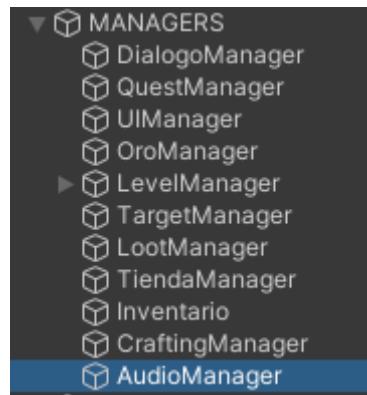


Ilustración 291 Jerarquía manager audio

A este manager de audio le añadí el componente Audio Source y seleccioné el clip de que quería que sonara durante la partida, el mismo que había guardado anteriormente en la carpeta de audio.

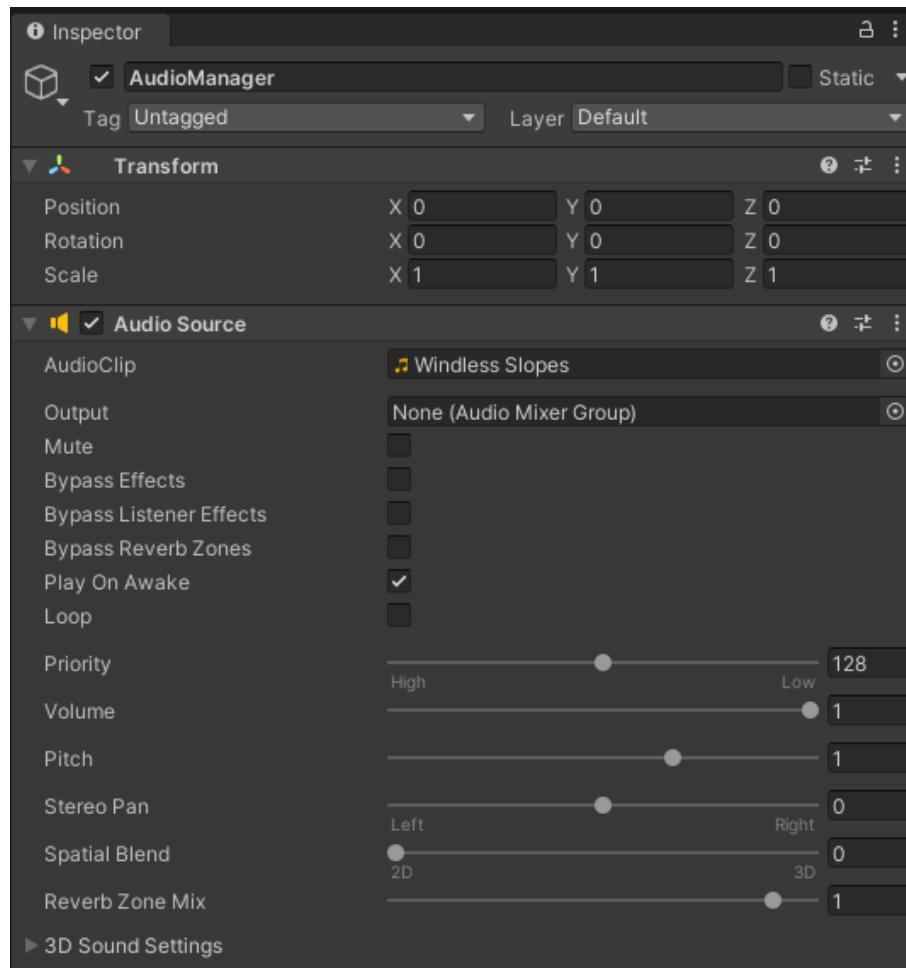


Ilustración 292 Inspector manager audio

Para poder editar desde el juego el volumen del audio, abrí el audiomixer y seleccioné el grupo de Master (el creado por defecto y el único que usaré).

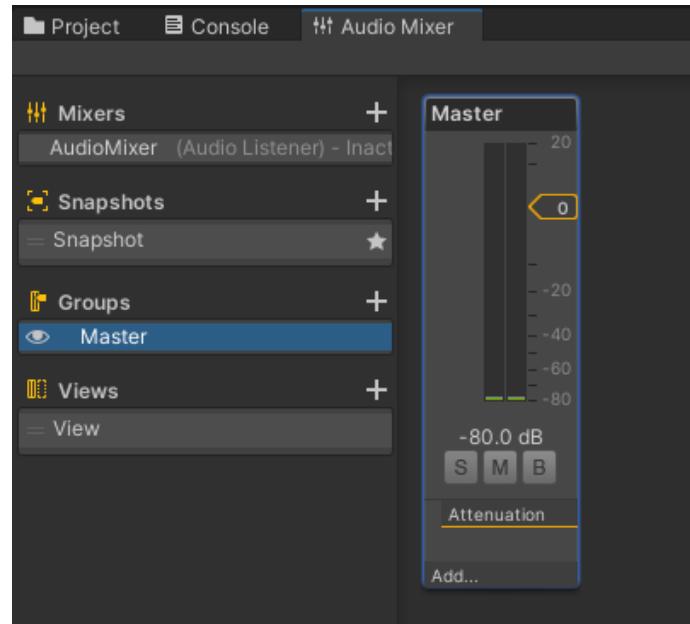


Ilustración 293 AudioMixer

Una vez seleccionado el grupo Master, hice clic derecho en el inspector para seleccionar los parámetros que quería exponer para poder llamar y editar desde un script. Cuando se ha seleccionado algún parámetro, aparece en la parte superior derecha del audiomixer.

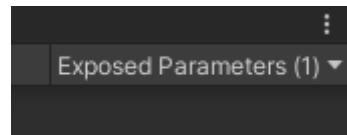


Ilustración 294 AudioMixer parámetros

A continuación, desde el manager de audio seleccioné el grupo editado del audiomixer como salida del audio en el videojuego.

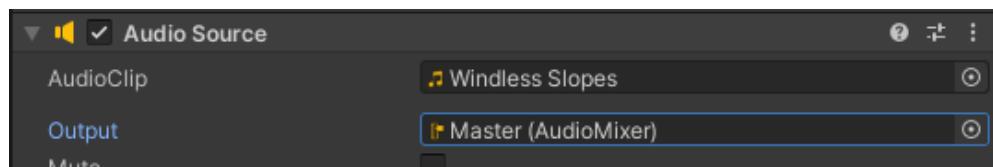


Ilustración 295 Inspector Audio Source

Para editar el parámetro de volumen, creé un script con referencia al audiomixer y al parámetro volumen, con métodos que permiten guardar y cargar el volumen que ha configurado el usuario al sistema de preferencias.

```

private void GuardarVolumen(float valor)
{
    this.volumen = valor;
    PlayerPrefs.SetFloat(VOLUME_KEY, volumen);
    PlayerPrefs.Save();
}

2 referencias
private void LoadVolumen()
{
    this.volumen = PlayerPrefs.GetFloat(VOLUME_KEY, 1f);
    this.mix?.SetFloat("VolumenMaster", Mathf.Log10(this.volumen) * 20);
}

```

Ilustración 296 Script preferencias

Y, como último detalle, solo quedaba permitir que el propio jugador tuviera acceso a editar el volumen. Para ello, cargué los métodos creados anteriormente para controlar el volumen en la barra de volumen del menú de configuración.

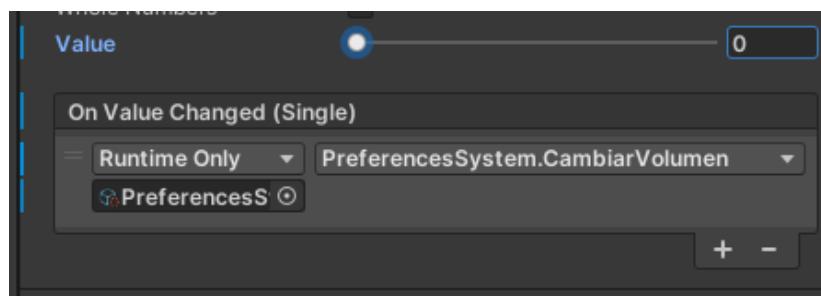


Ilustración 297 Inspector menu configuración barra volumen

TRABAJO A FUTURO

Hay algunas tareas que me gustaría realizar en un futuro para mejorar mi proyecto. Entre ellas se encuentran hacer una versión del videojuego para plataformas móviles, balancear la dificultad y otros pequeños detalles.

Plataformas móviles

Ya que este videojuego es un RPG al que poder jugar partidas cortas, creo que sería cómodo jugarlo en plataformas móviles, además de en el ordenador. Para ello, tendría que hacer algunos pequeños cambios en la interfaz y en los controles

Controles

Mi idea es crear un nuevo sistema de inputs que, en lugar de usar las teclas para moverse, use los controles de un joystick. Este joystick lo crearía en una nueva capa del canvas, poniendo dos círculos concéntricos. El círculo exterior estaría fijo, mientras el de dentro de movería dentro del círculo exterior y tendría asociado los controles del joystick. Este joystick estaría siempre presente en la parte inferior derecha de la pantalla. Además del joystick para controlar el movimiento, el click derecho se cambiaría por un toque en la pantalla táctil. Por último, añadiría al lado del joystick un botón en la misma interfaz para poder atacar a los enemigos, sirviendo de sustituto de la barra espaciadora en la versión de Windows.

Interfaz

Tendría que cambiar los menús creados para la versión de Windows. Las funcionalidades de cada menú seguirían siendo las mismas, pero tendrían una forma más alargada verticalmente para encajar en la pantalla de un móvil.

Cámara

Puesto que jugar en una pantalla en vertical nos quita mucho rango de visión a los lados del personaje, alejaría un poco la cámara para aumentar el contenido que entra en el campo de visión. También fijaría la cámara de forma que siga al protagonista cada vez que se mueve a la derecha y a la izquierda, pero le daría más rango para que se moviera libremente hacia arriba y hacia abajo sin ser perseguido por la cámara.

Balancear dificultad

Hasta ahora le he dado mucha más importancia a la planificación y la ejecución de las mecánicas, siendo la programación el principal reto del proyecto. Es una decisión lógica ya que se trata de un trabajo para Ingeniería Informática; sin embargo, creo que a la hora de desarrollar un juego hay otros roles muy importantes, como el de game design.

Un juego RPG generalmente está pensado para ofrecer un reto al jugador, además de ofrecerle diversión. Para ofrecer este reto, me gustaría en un futuro balancear la dificultad del juego, estudiando de una forma más consciente la cantidad de enemigos necesaria, sus posiciones, su vida, daño, etc. De esta forma pretendo que el juego sea más difícil (no imposible) y requiera de cualidades para poder terminarlo. Además, me gustaría añadir

más requisitos para que el protagonista pueda pasar de un lugar a otro del mapa, evitando que simplemente tenga que matar enemigos en un orden casi aleatorio.

Creo que este trabajo a futuro, a pesar de ser para mí el más importante, también será más difícil: tengo experiencia como jugadora de videojuegos RPG pero nunca hasta ahora había intentado diseñar uno, y creo que es un trabajo que requiere bastante experiencia para poder ser desarrollado correctamente.

Otros

Aunque sean detalles menos importantes, me gustaría añadir pequeñas funcionalidades como poder cerrar los menús con el botón escape, ya que mientras jugaba me he dado cuenta de que intento cerrarlos a ese botón por costumbre, aún habiéndolos creado yo misma. Además, me gustaría que, al tener abierto un menú, no se pudieran abrir otros a la vez. También quiero desactivar el movimiento del protagonista mientras los menús estén abiertos.

CONCLUSIONES

Como planteé en los objetivos, quería desarrollar como proyecto un videojuego RPG con los conocimientos adquiridos a lo largo de la carrera de Ingeniería del Software. Definitivamente, esta carrera me ha preparado para desarrollar correctamente proyectos con tecnologías no usadas durante la carrera y he conseguido crear mi primer videojuego RPG cumpliendo con todos los requisitos establecidos.

Haber programado durante años con varios lenguajes muy distintos entre ellos me ha dado la soltura para aprender rápidamente otro lenguaje nuevo. He tenido que ayudarme de documentación y he tenido que buscar muchas cosas por internet, pero he podido desarrollar un videojuego satisfactoriamente usando un lenguaje que era nuevo para mí, C#.

Por otro lado, haber hecho muchos proyectos distintos durante la carrera me ha ayudado mucho a organizarme y planificar todas las tareas necesarias.

Sin embargo, me he dado cuenta de que es muy difícil estimar tareas específicas cuando hay que realizarlas con tecnologías poco conocidas. Por mi parte, creo que he estimado con un tiempo demasiado largo algunas tareas de programación que luego he terminado rápidamente. Creo que las he estimado de esta forma porque tenía miedo de no saber enfrentarme a este nuevo lenguaje de programación. Por otro lado, he estimado con un tiempo menor algunas tareas más relacionadas con arte o que solo necesitaba el entorno de Unity para realizarlas (sin programar) y he tardado bastante más tiempo en realizarlas.

Sé que es normal no estimar bien las tareas cuando no tienes experiencia realizándolas, y en mi caso no tenía ninguna experiencia en Unity. Sin embargo, analizando en qué tipo de tareas he cometido errores de estimación baja y alta, puedo darme cuenta de que, aún estando cerca de terminar la carrera, le sigo teniendo miedo a programar y pensando que soy peor programadora de lo que soy realmente.

A pesar de estos errores de estimación, he podido realizar el trabajo a tiempo: es importante dejar un tiempo de margen para errores o estimar siempre pensando que vas a tardar más tiempo, por si acaso. En mi caso, pude realizar el trabajo con éxito gracias a lo primero.

En general estoy muy contenta con mi proyecto. Sigo teniendo mucho miedo a no saber realizar tareas o adaptarme a algunos proyectos, pero me he puesto a prueba con algo completamente nuevo y he sabido planificarlo, organizarlo y desarrollarlo con éxito. Me siento muy agradecida de haber podido estudiar esta carrera y de que me hayan enseñado tantas cosas, y también me siento orgullosa de haberme retado a mí misma y haber conseguido hacer realidad este proyecto.

DICCIONARIO

ADD: Documento de Diseño de Arte, parte del Documento de Diseño de Videojuegos referente a los elementos de arte que componen el videojuego.

Arma física: arma con la que se ataca de cerca a los enemigos.

Arma mágica: arma con la que se ataca de lejos a los enemigos a cambio de maná.

Asset: elementos que se pueden usar dentro del proyecto. Pueden ser scripts, animaciones, audios, imágenes, etc.

Build: versión ejecutable y funcional del código del videojuego.

Canvas (Unity): GameObject del que heredan todos los elementos de la interfaz de usuario.

Cinemachine (Unity): conjunto de cámaras inteligentes y dinámicas que se pueden usar y modificar sin necesidad de código.

Collider: componente que define la forma de un objeto con propósito de controlar las interacciones y las colisiones físicas.

Craftear: fabricar objetos a partir de otros ya existentes o de elementos básicos recolectables.

CRPG: videojuego de rol para ordenador

Game Designer: persona encargada de diseñar los elementos de un videojuego, incluyendo mecánicas y niveles.

GameObject: objetos fundamentales de Unity que funcionan de contenedores para los componentes.

GDD: Documento de Diseño de Videojuegos, documento vivo que sintetiza lo que va a ser el videojuego. Es la base para el desarrollo de un videojuego.

Grid (Unity): capas que permiten colocar sus elementos hijos dentro de cuadrículas. Se suele utilizar para pintar mapas y elementos en videojuegos de estética píxel.

HUD: barra de estado, información visible en todo momento durante la partida que contiene información importante para el jugador, como la vida del personaje.

Idle: animación en la que un personaje está parado en un mismo sitio realizando una animación de espera, como respirar.

Lootear: saquear, recoger objetos que suelta un enemigo al ser matado.

Maná: energía ficticia que se recupera sola y que se debe gastar para usar habilidades mágicas o usar armas mágicas.

Manager: scripts principales desde los que se llaman a los métodos de otros scripts.

NPC: Personaje No Jugable, personajes que forman parte de la historia de un videojuego o que nos ayudan al desarrollo interactuando con ellos. Estos personajes no pueden ser manejados por el jugador.

Objeto encriptable (ScriptableObject): clase que permite almacenar datos independientes de los instanciados en los scripts.

Prefab: asset que permite almacenar un objeto con componentes y propiedades asignados. Sirven de plantilla para instanciar los objetos en la escena.

Respawn: lugar del escenario de un videojuego donde el jugador reaparece tras morir.

RPG: videojuego de rol donde el jugador controla las acciones del protagonista dentro de un mundo típicamente de fantasía.

Rigidbody: componente que permite a los objetos actuar según la física.

Script: código que da órdenes a partes del programa, obligándole a realizar ciertas acciones.

Slime: enemigo con forma de baba redonda.

Slot: cada una de las ranuras del inventario que contienen objetos.

Spawn: zona donde aparece el jugador cuando empieza la partida.

Sprite: recurso gráfico en forma de mapa de bits que representan elementos del juego mediante arte píxel.

Spritesheet: archivo que contiene varios sprites dispuestos en forma de cuadrícula con los que se pueden crear animaciones.

TextMesh Pro (Unity): tipo de texto que nos permite reemplazar componentes existentes por texto que se muestra en la interfaz del juego.

Tilemap: paleta creada a partir de teselas que permiten pintar un escenario sobre un grid usando cada una de estas teselas.

Unity: motor de creación de videojuegos, uno de los más usados en la actualidad.

Unity Hub: activador de Unity y gestor que nos permite acceder a nuestros proyectos.

REFERENCIAS

- i https://es.wikipedia.org/wiki/Documento_de_dise%C3%B1o_de_videojuegos
- ii https://es.wikipedia.org/wiki/Videojuego_de_rol
- iii https://es.wikipedia.org/wiki/Pan_European_Game_Information
- iv <https://www.glassdoor.es/index.htm>
- v <https://unity.com/es>
- vi <https://www.aseprite.org/>
- vii <https://www.adobe.com/es/products/photoshop.html>
- viii <https://docs.unity3d.com/Manual/index.html>
- ix <https://www.domestika.org/es>
- x <https://www.udemy.com/es/>
- xi <https://www.soloempleo.com/diferencias-entre-unity-y-unreal-engine>
- xii <https://unity.com/es/download>
- xiii <https://itch.io/>