



Universidad de Jaén

Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE GRADO

REMAKE DEL VIDEOJUEGO DRAGON QUEST

Alumno

Juan Samblas Iglesias

Tutor

Carlos Javier Ogayar Anguita

(Departamento de Informática)

Septiembre, 2022

(Página intencionalmente en blanco)



Universidad de Jaén

Departamento de Informática

Don Carlos Javier Ogayar Anguita, tutor del Trabajo Fin de Grado titulado: '**Remake del videojuego Dragon Quest**', que presenta Don Juan Samblas Iglesias, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2022

El alumno:

SAMBLAS IGLESIAS JUAN - 21035972B
Firmado digitalmente por SAMBLAS IGLESIAS JUAN - 21035972B
Fecha: 2022.09.08 17:37:04 +02'00'
Juan Samblas Iglesias

El tutor:

OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Firmado digitalmente por OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-26039170L, givenName=CARLOS JAVIER, sn=OGAYAR ANGUITA, cn=OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Fecha: 2022.09.08 17:13:16 +02'00'
Carlos Javier Ogayar Anguita

(Página intencionalmente en blanco)

Agradecimientos

Quiero expresar mis agradecimientos primeramente a mi familia, que me ha apoyado durante todos los años de carrera, a mis tíos, tías, primos, primas, a mi abuela Adora, a mi abuela María que en paz descanse, y sobretodo a mis padres, que a pesar de muchas veces ser demasiado pesados y atosigarme más de la cuenta, sé que lo hacen porque se preocupan por mí y porque quieren que tenga un buen futuro. Aunque no lo suela decir, os quiero mucho a todos y no podría haber llegado hasta aquí sin vosotros.

También quiero agradecer a mis amigos, tanto a Ángel, Pablo y Jesús, mis amigos de toda la vida, que aunque no lo sepan me han ayudado bastante a sobrellevar estos últimos meses, como al resto de amigos que he hecho todos estos años y que me han hecho sentir como en casa fuera de esta. A pesar de que con algunos de vosotros ya no mantenga contacto o que apenas lo haga, me habéis ayudado a lo largo de estos años después de irme de mi casa y de mi pueblo por primera vez. Muchas gracias, os aprecio mucho a todos.

Y también, quiero agradecer a mi tutor por haberse portado tan bien conmigo, respondiendo todas mis dudas a pesar de todas las preguntas que he hecho y de tener más alumnos con los que tratar y más cosas que hacer. Muchas gracias.

En general, quiero agradecer a todos los que de una u otra manera me habéis apoyado durante todos estos años, en los que he pasado momentos buenos y momentos no tan buenos, pero que con ayuda de todos se han hecho sobrellevables.

Muchas gracias a todos.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Tecnologías de la Información
Mención (solo TFG)	Sistemas Gráficos
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Juan Samblas Iglesias
Fecha de asignación	10/11/2021
Descripción corta	Realización de un remake para el videojuego clásico Dragon Quest de 1986. Buscando crear desde cero una versión mejorada del mismo, tanto a nivel gráfico como de mecánicas y de jugabilidad.

NORMAS APLICADAS EN ESTE DOCUMENTO

LOCALES

TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén

NACIONALES E INTERNACIONALES

ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA

NACIONALES

UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información

*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*

Contenido

1	Especificación del trabajo	23
1.1	Introducción.....	23
1.2	Objetivos del trabajo	24
1.3	Antecedentes y estado del arte	26
1.3.1	Principales motores de videojuegos.....	26
1.3.1.1	Unreal Engine	27
1.3.1.2	CryEngine	29
1.3.1.3	Amazon Lumberyard	30
1.3.1.4	GODOT.....	31
1.3.1.5	RPG Maker.....	33
1.3.1.6	Unity3D	34
1.3.2	Estudio de popularidad entre los principales motores.....	36
1.3.3	Elección de motor de videojuegos.....	39
1.4	Descripción de la situación de partida	40
1.4.1	Descripción del entorno actual	40
1.4.2	Resumen de las deficiencias y carencias identificadas	41
1.4.2.1	Carencias.....	41
1.4.2.2	Deficiencias	41
1.5	Requisitos iniciales.....	43
1.6	Alcance.....	45
1.7	Hipótesis y restricciones	46
1.8	Tecnologías utilizadas.....	47
1.9	Estudio de alternativas y viabilidad	47
1.10	Descripción de la solución propuesta	48
1.11	Metodología de desarrollo de software.....	49
1.12	Análisis de riesgos	50
1.13	Estimación del tamaño y esfuerzo	51
1.14	Planificación temporal.....	56
1.14.1	Diagrama de Gantt – Diciembre 2021	57
1.14.2	Diagrama de Gantt – Enero 2022.....	57
1.14.3	Diagrama de Gantt – Julio 2022.....	58
1.14.4	Diagrama de Gantt – Agosto 2022.....	58
1.14.5	Diagrama de Gantt – Septiembre 2022.....	59
1.15	Presupuesto.....	59
2	Diseño inicial	61
2.1	Especificaciones del sistema	61
2.1.1	Requisitos Funcionales	61
2.1.2	Requisitos no funcionales.....	62
2.2	Análisis y diseño del sistema	63
2.2.1	Casos de uso.....	63
2.2.1.1	CU-01 Crear una partida nueva	64
2.2.1.2	CU-02 Cargar una partida	65
2.2.1.3	CU-03 Guardar partida	65
2.2.1.4	CU-04 Mover al jugador	66
2.2.1.5	CU-05 Entablar diálogos	67
2.2.1.6	CU-06 Comprar Objetos tienda	68

2.2.1.7	CU-07 Vender Objetos inventario.....	69
2.2.1.8	CU-08 Obtener Objetos inventario	69
2.2.1.9	CU-09 Usar Objetos inventario.....	70
2.2.1.10	CU-10 Equipar Objetos Inventario	71
2.2.1.11	CU-11 Desequipar objetos inventario	71
2.2.1.12	CU-12 Eliminar objetos	72
2.2.1.13	CU-13 Completar misiones	73
2.2.1.14	CU-14 Atacar	73
2.2.1.15	CU-15 Usar objetos batalla	74
2.2.1.16	CU-16 Huir	75
2.2.1.17	CU-17 Menú pausa	75
2.2.1.18	CU-18 Completar Juego	76
2.2.1.19	CU-19 Matar enemigos	77
2.2.1.20	CU-20 Conseguir experiencia y oro	77
2.2.1.21	CU-21 Recibir daño.....	78
2.2.1.22	CU-22 Infligir daño	79
2.2.1.23	CU-23 Curar.....	79
2.2.1.24	CU-24 Permitir al jugador utilizar conjuros.	80
2.2.1.25	CU-25 Subir de nivel.....	81
2.2.1.26	CU-26 Luchar.....	81
2.2.2	Diagramas de clases	83
2.2.2.1	Diagrama de clases batalla	83
2.2.2.2	Diagrama de clases diálogo	84
2.2.2.3	Diagrama de clases tienda	85
2.2.3	Diseño interfaz.....	85
3	Desarrollo	89
3.1	Primera Iteración – Creación de escenas y mapas	89
3.1.1	Castillo_P1.....	90
3.1.2	Castillo_P0.....	93
3.1.3	Castillo_P-1	95
3.1.4	World_Map	96
3.1.5	Breconary.....	100
3.1.6	Cueva de Erdrick	101
3.1.7	Cueva_Erdrick-P-1	103
3.1.8	Galenholm	104
3.1.9	Galenholm_P-1	106
3.1.10	Galenholm_P-2.....	107
3.1.11	Galenholm_P-3.....	108
3.1.12	Galenholm_P-4.....	109
3.1.13	Kol.....	110
3.1.14	Santuario_de_la_Lluvia	111
3.1.15	Santuario_de_la_Lluvia_P-1	112
3.1.16	Cueva_del_Pantano	113
3.1.17	Rimuldar	115
3.1.18	Santuario_Sagrado.....	116
3.1.19	Santuario_Sagrado_P-1	116
3.1.20	Cueva_Montañas_Rocosas	117
3.1.21	Cueva_Montañas_Rocosas_P-1	118
3.1.22	Ciudad_Destruida_de_Domdora	119

3.1.23	Mercado / Catlin.....	120
3.1.24	Castillo_DragonLord.....	121
3.1.25	Castillo_DragonLord_P-1.....	123
3.1.26	Castillo_DragonLord_P-2.....	124
3.1.27	Castillo_DragonLord_P-3.....	125
3.1.28	Castillo_DragonLord_P-4.....	126
3.1.29	Castillo_DragonLord_P-5A.....	127
3.1.30	Castillo_DragonLord_P-5B.....	128
3.1.31	Castillo_DragonLord_P-6B.....	129
3.1.32	Castillo_DragonLord_P-7.....	130
3.2	Segunda Iteración – Creación del jugador y transición entre escenas.....	131
3.2.1	Creación del Prefab del Jugador.....	132
3.2.2	Creación de colliders y transición de escenas.....	140
3.3	Tercera Iteración – Creación del menú de inicio y de la UI.....	146
3.3.1	Menu_Inicio.....	147
3.3.2	Interfaz para diálogos y mánager de diálogos.....	153
3.3.3	Interfaz Menú de Pausa y Manager Menú de Pausa.....	157
3.3.4	Interfaz de combate.....	165
3.4	Cuarta Iteración – Creación de NPCs, Manager de Batalla, manejo del menú de pausa, etc. 170	
3.4.1	Creación de NPCs.....	171
3.4.1.1	NPC_NO_MOV y NPC_NO_MOV_2.....	172
3.4.1.2	NPC_MOV.....	175
3.4.1.3	NPC_OBJETO.....	178
3.4.1.4	Anciano_Castillo.....	181
3.4.1.5	NPC_TIENDA.....	181
3.4.1.6	NPCs utilizados.....	206
3.4.2	Estadísticas.....	212
3.4.3	Mánager Menú Pausa actualizado.....	216
3.4.3.1	Inventario.....	216
3.4.3.2	Atributos.....	218
3.4.3.3	Magia.....	220
3.4.4	Mánager de Batalla.....	222
3.5	Pruebas finales.....	253
3.5.1	Pruebas de verificación del sistema.....	253
3.5.2	Pruebas de validación del sistema.....	254
4	Conclusiones y trabajos futuros.....	255
5	Apéndices.....	256
5.1	Guía original del Trabajo Fin de Título.....	256
5.1.1	Guía original.....	256
5.1.1.1	Conocimientos previos.....	256
5.1.1.2	Objetivos del TFG.....	256
5.1.1.3	Metodología a Desarrollar.....	257
5.1.1.4	Documentos y Formatos de entrega.....	257
5.1.2	Modificación del TFG.....	258
5.2	Instalación y configuración del sistema.....	258
5.3	Manuales de usuario.....	259

6	Definiciones y abreviaturas	260
7	Bibliografía	261

Índice de ilustraciones

Ilustración 1.1: Demo técnica UE5 en Play Station 5	27
Ilustración 1.2: Juego, Hunt: Showdown.....	29
Ilustración 1.3: Juego MMORPG de Amazon, New World	30
Ilustración 1.4: Juego, Kingdoms of the Dump	31
Ilustración 1.5: Carátula del remake para Nintendo Switch del Juego, Corpse Party.....	33
Ilustración 1.6: Juego, Cuphead	34
Ilustración 1.7: Gráfico, Porcentaje de desarrolladores que utilizan los siguientes motores..	37
Ilustración 1.8: Gráfico, Número de proyecto por motor de videojuegos de Itch.io.....	38
Ilustración 1.9: Gráfico, Captura del remake de DQ para Android	42
Ilustración 1.10: Diagrama de Gantt del proyecto – Diciembre 2021	57
Ilustración 1.11: Diagrama de Gantt del proyecto – Enero 2022.....	57
Ilustración 1.12: Diagrama de Gantt del proyecto – Julio 2022	58
Ilustración 1.13: Diagrama de Gantt del proyecto – Agosto 2022	58
Ilustración 1.14: Diagrama de Gantt del proyecto – Septiembre 2022	59
Ilustración 2.1: Diagrama de clases - Batalla.....	83
Ilustración 2.2: Diagrama de clases - Diálogo.....	84
Ilustración 2.3: Diagrama de clases - Tienda.....	85
Ilustración 2.4: Menú Pausa Dragon Quest XI.....	86
Ilustración 2.5: Atributos Dragon Quest XI.....	86
Ilustración 2.6: Inventario Dragon Quest XI	87
Ilustración 2.7: Diálogo Dragon Quest XI.....	87
Ilustración 2.8: Selección Dragon Quest XI	88
Ilustración 2.9: Inicio Dragon Quest XI	88
Ilustración 3.1: Diagrama Burndown primera iteración.....	89
Ilustración 3.2: Build Settings del proyecto en Unity	90
Ilustración 3.3: Castillo P1 MSX2	90
Ilustración 3.4: Castillo P1 de nuestro remake.....	91
Ilustración 3.5: Paleta Pueblo1	91
Ilustración 3.6: Castillo_P1_manager - entrar_lugar	92
Ilustración 3.7: Castillo_P1_manager - SalirDelMapa.....	92
Ilustración 3.8: Castillo_P1_manager – AnimacionCambioEscena.....	93
Ilustración 3.9: Animación Cambio de Escena.....	93
Ilustración 3.10: Castillo P0 MSX2	94
Ilustración 3.11: Castillo P0 de nuestro remake.....	94
Ilustración 3.12: Ejemplo de casilla animada.....	95
Ilustración 3.13: Castillo P-1 MSX2	95
Ilustración 3.14: Castillo P-1 de nuestro remake.....	96
Ilustración 3.15: Parte superior izquierda - Mapa Mundo - MSX2	97
Ilustración 3.16: Parte superior derecha - Mapa Mundo - MSX2.....	97
Ilustración 3.17: Parte inferior izquierda - Mapa Mundo - MSX2.....	98
Ilustración 3.18: Parte inferior derecha - Mapa Mundo - MSX2.....	98
Ilustración 3.19: World_Map de nuestro remake.....	99
Ilustración 3.20: Paleta WorldMap	100
Ilustración 3.21: Breconary MSX2.....	101

Ilustración 3.22: Breconary de nuestro remake	101
Ilustración 3.23: Cueva de Erdrick MSX2	101
Ilustración 3.24: Cueva de Erdrick de nuestro remake	102
Ilustración 3.25: Paleta Mazmorra1	102
Ilustración 3.26: Modo Cueva	103
Ilustración 3.27: Cueva de Erdrick P-1 MSX2	103
Ilustración 3.28: Cueva de Erdrick P-1 de nuestro remake	104
Ilustración 3.29: Galenholm MSX2	104
Ilustración 3.30: Galenholm de nuestro remake	105
Ilustración 3.31: Paleta Pueblo2	105
Ilustración 3.32: Galenholm P-1 MSX2	106
Ilustración 3.33: Galenholm P-1 de nuestro remake	106
Ilustración 3.34: Ejemplo de "variables escalera"	107
Ilustración 3.35: Galenholm P-2 MSX2	107
Ilustración 3.36: Galenholm P-2 de nuestro remake	108
Ilustración 3.37: Galenholm P-3 MSX2	108
Ilustración 3.38: Galenholm P-3 de nuestro remake	109
Ilustración 3.39: Galenholm P-4 MSX2	109
Ilustración 3.40: Galenholm P-4 de nuestro remake	110
Ilustración 3.41: Kol MSX2	110
Ilustración 3.42: Kol de nuestro remake	111
Ilustración 3.43: Santuario de la Lluvia – remake	111
Ilustración 3.44: Santuario de la Lluvia MSX2	112
Ilustración 3.45: Santuario de la Lluvia P-1 de nuestro remake	112
Ilustración 3.46: Paleta Mazmorra2	113
Ilustración 3.47: Cueva del Pantano MSX2	113
Ilustración 3.48: Cueva del Pantano de nuestro remake	114
Ilustración 3.49: Paleta Mazmorra2A	114
Ilustración 3.50: Rimuldar MSX2	115
Ilustración 3.51: Rimuldar de nuestro remake	115
Ilustración 3.52: Santuario Sagrado de nuestro remake	116
Ilustración 3.53: Santuario Sagrado MSX2	116
Ilustración 3.54: Santuario Sagrado P-1 de nuestro remake	117
Ilustración 3.55: Cueva Montañas Rocosas MSX2	117
Ilustración 3.56: Cueva Montañas Rocosas de nuestro remake	118
Ilustración 3.57: Cueva Montañas Rocosas P-1 MSX2	118
Ilustración 3.58: Cueva Montañas Rocosas P-1 de nuestro remake	119
Ilustración 3.59: Ciudad Destruída de Domdora MSX2	119
Ilustración 3.60: Ciudad Destruída de Domdora de nuestro remake	120
Ilustración 3.61: Mercado MSX2	120
Ilustración 3.62: Mercado de nuestro remake	121
Ilustración 3.63: Castillo DragonLord MSX2	121
Ilustración 3.64: Castillo DragonLord de nuestro remake	122
Ilustración 3.65: Paleta Mazmorra3	122
Ilustración 3.66: Castillo DragonLord P-1 MSX2	123
Ilustración 3.67: Castillo DragonLord P-1 de nuestro remake	123

Ilustración 3.68: Castillo DragonLord P-2 MSX2.....	124
Ilustración 3.69: Castillo DragonLord de nuestro remake	124
Ilustración 3.70: Castillo DragonLord P-3 MSX2.....	125
Ilustración 3.71: Castillo DragonLord P-3 de nuestro remake	125
Ilustración 3.72: Castillo DragonLord P-4 MSX2.....	126
Ilustración 3.73: Castillo DragonLord P-4 de nuestro remake	126
Ilustración 3.74: Castillo DragonLord P-5 MSX2.....	127
Ilustración 3.75: Castillo DragonLord P-5A de nuestro remake.....	127
Ilustración 3.76: Castillo DragonLord P-5 MSX2.....	128
Ilustración 3.77: Castillo DragonLord P-5B de nuestro remake.....	128
Ilustración 3.78: Castillo DragonLord P-6B MSX2	129
Ilustración 3.79: Castillo DragonLord P-6B de nuestro remake.....	129
Ilustración 3.80: Castillo DragonLord P-7 MSX2.....	130
Ilustración 3.81: Castillo DragonLord P-7 de nuestro remake	130
Ilustración 3.82: Diagrama Burndown - Creación del jugador y transición entre escenas ...	131
Ilustración 3.83: Jugador	132
Ilustración 3.84: Sprites jugador	133
Ilustración 3.85: Código del Script "Movimiento" 1.....	133
Ilustración 3.86: Código del Script "Movimiento" 2.....	134
Ilustración 3.87: Código del Script "Movimiento" 3.....	135
Ilustración 3.88: Código del Script "Movimiento" 4.....	135
Ilustración 3.89: Código del Script "Movimiento" 5.....	136
Ilustración 3.90: Código del Script "Movimiento" 6.....	136
Ilustración 3.91: Animación jugador.....	137
Ilustración 3.92: Código del Script "Movimiento" 7.....	137
Ilustración 3.93: Código del Script "Movimiento" 8.....	137
Ilustración 3.94: Código del Script "Movimiento" 9.....	138
Ilustración 3.95: Capa para colliders del jugador	138
Ilustración 3.96: Código del Script "Movimiento" 10.....	139
Ilustración 3.97: Colliders Breconary	140
Ilustración 3.98: Colliders Breconary usando Composite	141
Ilustración 3.99: Colliders Breconary sin usar Composite	141
Ilustración 3.100: Asignación de capa Collider	141
Ilustración 3.101: Script DontDestroyOnLoad.....	142
Ilustración 3.102: AnimacionCambioEscena.....	143
Ilustración 3.103: Código Script "Castillo_P1_manager".....	143
Ilustración 3.104: Comprobar escalera de entrada Galenholm P-2.....	144
Ilustración 3.105: lugarAparicion	144
Ilustración 3.106: SistemaGuardado.....	145
Ilustración 3.107: DatosLugar.....	145
Ilustración 3.108: Diagrama Burndown - Creación del menú inicial y la UI	146
Ilustración 3.109: Elementos UI Menú Inicio.....	147
Ilustración 3.110: Background Menu_Inicio	147
Ilustración 3.111: Dragon_azul	148
Ilustración 3.112: Título	148
Ilustración 3.113: Llamas.....	148

Ilustración 3.114: TM	148
Ilustración 3.115: Dragon-Quest-Menu_Background	149
Ilustración 3.116: Cursor	149
Ilustración 3.117: Menu_Inicio editor	149
Ilustración 3.118: Código Script "Gestor_Menu" 1	150
Ilustración 3.119: Código Script "Gestor_Menu" 2	151
Ilustración 3.120: Menú al pulsar	151
Ilustración 3.121: Código Script "Gestor_Menu" 3	151
Ilustración 3.122: Código del Script "Gestor_Menu" 4	152
Ilustración 3.123: Código del Script "Gestor_Menu" 5	153
Ilustración 3.124: Dragon-Quest-Dialogue_Opcion	153
Ilustración 3.125: Dragon-Quest-Dialogue_Background	154
Ilustración 3.126: Elementos Tienda	154
Ilustración 3.127: Tienda	154
Ilustración 3.128: Vertical Layout – Slots	155
Ilustración 3.129: Prefab Item	155
Ilustración 3.130: Clase Dialogo	155
Ilustración 3.131: Clase Manager_Dialogos	156
Ilustración 3.132: Código del Script "Manager_Dialogos" 1	156
Ilustración 3.133: Código del Script "Manager_Dialogos" 2	157
Ilustración 3.134: Código del Script "Manager_Dialogos" 3	157
Ilustración 3.135: Menu_inicial del Menú de Pausa	157
Ilustración 3.136: Dragon-Quest-Pause-Menu	158
Ilustración 3.137: Interfaz Inventario	158
Ilustración 3.138: Dragon-Quest-Eleccion-Inventario	158
Ilustración 3.139: Dragon-Quest-Inventario-Dinero	158
Ilustración 3.140: Dragon-Quest-Dialogo-Inventario	159
Ilustración 3.141: Dragon-Quest-Inventario	159
Ilustración 3.142: Estadísticas Jugador	159
Ilustración 3.143: Dragon-Quest-Estadisticas	160
Ilustración 3.144: Main_Character	160
Ilustración 3.145: Icono Espada	160
Ilustración 3.146: Icono Espadón	160
Ilustración 3.147: Icono Hacha	161
Ilustración 3.148: Icono Lanza	161
Ilustración 3.149: Icono Daga	161
Ilustración 3.150: Icono Casco	161
Ilustración 3.151: Icono Armadura	161
Ilustración 3.152: Icono Escudo	161
Ilustración 3.153: Icono Anillo	161
Ilustración 3.154: Icono Collar	161
Ilustración 3.155: Menú Magias	162
Ilustración 3.156: Menú Miscelánea	162
Ilustración 3.157: Código del Script "Menu_Pausa" 1	163
Ilustración 3.158: Código del Script "Menu_Pausa" 2	163
Ilustración 3.159: Código del Script "Menu_Pausa" 2	164

Ilustración 3.160: Código del Script "Menu_Pausa" 3	164
Ilustración 3.161: Código del Script "Menu_Pausa" 4	165
Ilustración 3.162: Objetos de Manager_Batalla	166
Ilustración 3.163: Estado - Manager_Batalla	166
Ilustración 3.164: Acciones Batalla	166
Ilustración 3.165: Diálogo batalla	167
Ilustración 3.166: Escenario Combate	167
Ilustración 3.167: Animación corte_espada_bronce	168
Ilustración 3.168: Animación corte_espada_fuego	168
Ilustración 3.169: Animación corte_espada_normal	168
Ilustración 3.170: Animación golpe_sin_arma	168
Ilustración 3.171: Animación Incendio	169
Ilustración 3.172: Animación Mini-Incendio	169
Ilustración 3.173: Animación Mini-Curación	169
Ilustración 3.174: Animación Negación Mágica	169
Ilustración 3.175: Animación Semi-Curación	170
Ilustración 3.176: Animación Sueño	170
Ilustración 3.177: Diagrama Burndown - Creación de NPCs, Manager de batalla, manejo del menú de pausa, etc.	171
Ilustración 3.178: Composición NPC	171
Ilustración 3.179: Código del Script "NPC_NO_NOV" 1	172
Ilustración 3.180: Código del Script "NPC_NO_NOV" 2	173
Ilustración 3.181: Código del Script "NPC_NO_MOV" 3	173
Ilustración 3.182: Código del Script "NPC_NO_MOV" 4	174
Ilustración 3.183: Ejemplo de Diálogo con NPC del tipo NPC_NO_MOV	174
Ilustración 3.184: Ejemplo de Diálogo con NPC del tipo NPC_NO_MOV_2	175
Ilustración 3.185: Código del Script "NPC_MOV" 1	176
Ilustración 3.186: Colliders NPC	176
Ilustración 3.187: Colliders NPCs en Breconary	177
Ilustración 3.188: Código del Script "NPC_MOV" 2	178
Ilustración 3.189: Código del Script "NPC_OBJETO" 1	178
Ilustración 3.190: LoadCofres	179
Ilustración 3.191: GuardarCofres	179
Ilustración 3.192: Clase DatosCofres	180
Ilustración 3.193: Código del Script "NPC_OBJETO" 2	180
Ilustración 3.194: Código del Script "Anciano_Castillo"	181
Ilustración 3.195: Código del Script "NPC_TIENDA" 1	182
Ilustración 3.196: Clase Item	182
Ilustración 3.197: Pasivas y tipoObjeto	183
Ilustración 3.198: Item como objeto Scripteable	183
Ilustración 3.199: Agua mágica	184
Ilustración 3.200: Ala de quimera	184
Ilustración 3.201: Medicina fuerte	184
Ilustración 3.202: Planta medicinal	185
Ilustración 3.203: Semilla de agilidad	185
Ilustración 3.204: Semilla de defensa	185

Ilustración 3.205: Semilla de fuerza.....	186
Ilustración 3.206: Semilla de vitalidad.....	186
Ilustración 3.207: Semilla mágica	186
Ilustración 3.208: Garrote	187
Ilustración 3.209: Espada de bronce	187
Ilustración 3.210: Daga de cobre.....	187
Ilustración 3.211: Espadón	187
Ilustración 3.212: Espada flamígera	188
Ilustración 3.213: Espada sagrada	188
Ilustración 3.214: Hacha.....	188
Ilustración 3.215: Espada de Erdrick	189
Ilustración 3.216: Armadura completa	189
Ilustración 3.217: Armadura de cuero.....	189
Ilustración 3.218: Ropa de viaje	189
Ilustración 3.219: Cota de malla	190
Ilustración 3.220: Pechera.....	190
Ilustración 3.221: Armadura sagrada.....	190
Ilustración 3.222: Armadura de Erdrick.....	191
Ilustración 3.223: Escudo de madera	191
Ilustración 3.224: Escudo de cuero.....	191
Ilustración 3.225: Escudo de acero.....	192
Ilustración 3.226: Escudo de guardián.....	192
Ilustración 3.227: Escudo de Erdrick	192
Ilustración 3.228: Casco de cuero	193
Ilustración 3.229: Casco de hierro	193
Ilustración 3.230: Casco de matadragones.....	193
Ilustración 3.231: Tiara de Erdrick	193
Ilustración 3.232: Anillo de agilidad	194
Ilustración 3.233: Anillo de fuerza.....	194
Ilustración 3.234: Anillo mágico	194
Ilustración 3.235: Collar de la diosa.....	195
Ilustración 3.236: Collar sacro	195
Ilustración 3.237: Collar maldito	195
Ilustración 3.238: Arpa de Galen	195
Ilustración 3.239: Lágrima de la Diosa.....	196
Ilustración 3.240: Báculo de la Lluvia	196
Ilustración 3.241: Marca de Erdrick	196
Ilustración 3.242: Piedra Solar.....	196
Ilustración 3.243: Código del Script "Inventario" 1	197
Ilustración 3.244: Código del Script "Inventario" 2	197
Ilustración 3.245: Clase DatosInventario	198
Ilustración 3.246: Métodos del Script "Inventario".....	199
Ilustración 3.247: Código del Script "Inventario" 3	200
Ilustración 3.248: Enum estadoCompra.....	200
Ilustración 3.249: Código del Script "Inventario" 4	201
Ilustración 3.250: Código del Script "Inventario" 5	201

Ilustración 3.251: Código del Script "NPC_TIENDA" 2	202
Ilustración 3.252: Código del Script "NPC_TIENDA" 3	202
Ilustración 3.253: Código del Script "NPC_TIENDA" 4	203
Ilustración 3.254: Código del Script "NPC_TIENDA" 5	203
Ilustración 3.255: Actualización del mostrador de objetos al movernos por la tienda.....	204
Ilustración 3.256: Tienda - Vender.....	205
Ilustración 3.257: Código del Script "NPC_TIENDA" 6	205
Ilustración 3.258: Código del Script "NPC_TIENDA" 7	205
Ilustración 3.259: Sprites Aldeana_1	206
Ilustración 3.260: Sprites Aldeana_2	206
Ilustración 3.261: Sprites Aldeano_1	207
Ilustración 3.262: Sprites Aldeano_2	207
Ilustración 3.263: Sprites Aldeano_3 y Tienda.....	207
Ilustración 3.264: Sprites Anciana_1	208
Ilustración 3.265: Sprites Anciano_1, Anciano_durmiendo y Sabios.....	208
Ilustración 3.266: Sprites Hombre, Hombre_durmiendo y Armero	208
Ilustración 3.267: Sprites Bardo.....	209
Ilustración 3.268: Sprites Caballero	209
Ilustración 3.269: Sprites Guardia y Guardia_durmiendo.....	209
Ilustración 3.270: Sprites Monja	210
Ilustración 3.271: Sprites Canciller	210
Ilustración 3.272: Sprites Rey.....	210
Ilustración 3.273: Sprites Princesa	211
Ilustración 3.274: Sprites Sacerdote	211
Ilustración 3.275: Sprites Dragón Verde Jefe	211
Ilustración 3.276: Sprites DragonLord	212
Ilustración 3.277: Clase Stats_personaje	212
Ilustración 3.278: Clase Stat.....	213
Ilustración 3.279: Código del Script "Stats_personajes" 1	213
Ilustración 3.280: Clase Stats_jugador	215
Ilustración 3.281: addExp	216
Ilustración 3.282: Interfaz Inventario Completa.....	217
Ilustración 3.283: Usar, Equipar, Desechar objetos	217
Ilustración 3.284: equipar	218
Ilustración 3.285: Sliders y atributos	219
Ilustración 3.286: Modificación de Sprites según equipamiento.....	219
Ilustración 3.287: Atributos con Garrote equipado	220
Ilustración 3.288: Atributos con Garrote desequipado	220
Ilustración 3.289: Hechizos fuera de combate	221
Ilustración 3.290: Hechizos bloqueados	221
Ilustración 3.291: Todos los hechizos desbloqueados.....	221
Ilustración 3.292: Atributos Manager_Batalla	222
Ilustración 3.293: Atributos nuevos Movimiento 1	223
Ilustración 3.294: Atributos nuevos Movimiento 2.....	223
Ilustración 3.295: Fondo de Batalla – Castillo.....	224
Ilustración 3.296: Fondo de Batalla – Arena	224

Ilustración 3.297: Fondo de Batalla - Arena amanecer	225
Ilustración 3.298: Fondo de Batalla - Arena Noche.....	225
Ilustración 3.299: Fondo de Batalla - Arena Tarde.....	225
Ilustración 3.300: Fondo de Batalla – Bosque	226
Ilustración 3.301: Fondo de Batalla - Bosque Amanecer	226
Ilustración 3.302: Fondo de Batalla - Bosque Noche	226
Ilustración 3.303: Fondo de Batalla - Bosque Tarde	227
Ilustración 3.304: Fondo de Batalla – Cueva	227
Ilustración 3.305: Fondo de Batalla - Cueva 2	227
Ilustración 3.306: Fondo de Batalla - Ciudad Destruída de Domdora.....	228
Ilustración 3.307: Fondo de Batalla – Montañas	228
Ilustración 3.308: Fondo de Batalla - Montañas Amanecer.....	228
Ilustración 3.309: Fondo de Batalla - Montañas Noche.....	229
Ilustración 3.310: Fondo de Batalla - Montañas Tarde.....	229
Ilustración 3.311: Fondo de Batalla – Pradera.....	229
Ilustración 3.312: Fondo de Batalla - Pradera Amanecer.....	230
Ilustración 3.313: Fondo de Batalla - Pradera Noche	230
Ilustración 3.314: Fondo de Batalla - Pradera Tarde	230
Ilustración 3.315: Fondo de Batalla - Pradera Oscura	231
Ilustración 3.316: Fondo de Batalla - Pradera Oscura Amanecer	231
Ilustración 3.317: Fondo de Batalla - Pradera Oscura Noche	231
Ilustración 3.318: Fondo de Batalla - Pradera Oscura Tarde	232
Ilustración 3.319: Iniciar Batalla (?)	232
Ilustración 3.320: spawnBatalla	233
Ilustración 3.321: triggerBatalla	233
Ilustración 3.322: prepararBatalla	234
Ilustración 3.323: Limo	235
Ilustración 3.324: Lima	235
Ilustración 3.325: Dracanino	235
Ilustración 3.326: Fantasma	235
Ilustración 3.327: Prestidigitador	236
Ilustración 3.328: Dracólito	236
Ilustración 3.329: Escorpión	236
Ilustración 3.330: Esqueleto	237
Ilustración 3.331: Selenita	237
Ilustración 3.332: Espectro Pendenciero	237
Ilustración 3.333: Babosidad	238
Ilustración 3.334: Dracanina.....	238
Ilustración 3.335: Ilusionista	238
Ilustración 3.336: Lobobo	239
Ilustración 3.337: Escorpión Férreo	239
Ilustración 3.338: Asesino Óseo	239
Ilustración 3.339: Lobobo Espantajo.....	240
Ilustración 3.340: Gólem de Oro	240
Ilustración 3.341: Quimera	240
Ilustración 3.342: Espectro Infernal	241

Ilustración 3.343: Selenita Lunático	241
Ilustración 3.344: Diablosidad.....	241
Ilustración 3.345: Esqueleto Soldado	241
Ilustración 3.346: Escorpión Letal.....	242
Ilustración 3.347: Caballero Andante.....	242
Ilustración 3.348: Esqueleto Oscuro	242
Ilustración 3.349: Quimera Engañosa.....	243
Ilustración 3.350: Limo Metálico	243
Ilustración 3.351: Lobobo Desgarrador.....	243
Ilustración 3.352: Quimera Cósmica.....	244
Ilustración 3.353: Dragón Verde	244
Ilustración 3.354: Mago Mayor	244
Ilustración 3.355: Gólem.....	245
Ilustración 3.356: Caballero Aberrante	245
Ilustración 3.357: Dragón Azul.....	245
Ilustración 3.358: Gólem de Piedra.....	246
Ilustración 3.359: Caballero Abominable	246
Ilustración 3.360: Dragón Temido.....	246
Ilustración 3.361: Draconarius - Primera Forma	247
Ilustración 3.362: Draconarius - Segunda Forma.....	247
Ilustración 3.363: Código del Script "Manager_Batalla" 1	248
Ilustración 3.364: Interfaz de Batalla activa	248
Ilustración 3.365: Código del Script "Manager_Batalla" 2	249
Ilustración 3.366: Uso de objetos en combate	249
Ilustración 3.367: Código del Script "Manager_Batalla" 3	250
Ilustración 3.368: Código del Script "Manager_Batalla" 4	250
Ilustración 3.369: Código del Script "Manager_Batalla" 5	251
Ilustración 3.370: Código del Script "Manager_Batalla" 6	251
Ilustración 3.371: Código del Script "Manager_Batalla" 7	252
Ilustración 5.1: Controles Dragon Quest remake	259

Índice de tablas

Tabla 1.1: Datos de encuesta “Game Engines used by video game developers in the UK” ..36	36
Tabla 1.2: Datos del sitio Itch.io, Engines más utilizados.....38	38
Tabla 1.3: Tipos de proyectos software y valores según COCOMO52	52
Tabla 1.4: Conductores de Coste para COCOMO53	53
Tabla 1.5: Costes asociados al proyecto60	60
Tabla 2.1: Casos de Uso64	64
Tabla 2.2: CU-01 Crear una partida nueva64	64
Tabla 2.3: CU-02 Cargar una partida.....65	65
Tabla 2.4: CU-03 Guardar partida66	66
Tabla 2.5: CU-04 Mover al jugador.....67	67
Tabla 2.6: CU-05 Entablar diálogo68	68
Tabla 2.7: CU-06 Comprar Objeto Tienda68	68
Tabla 2.8: CU-07 Vender Objetos Inventario69	69
Tabla 2.9: CU-08 Obtener Objetos inventario.....70	70
Tabla 2.10: CU-09 Usar Objetos inventario70	70
Tabla 2.11: CU-10 Equipar Objetos Inventario71	71
Tabla 2.12: CU-11 Desequipar objetos inventario72	72
Tabla 2.13: CU-12 Eliminar objetos72	72
Tabla 2.14: CU-13 Completar misiones.....73	73
Tabla 2.15: CU-14 Atacar74	74
Tabla 2.16: CU-15 Usar objetos batalla.....74	74
Tabla 2.17: CU-16 Huir.....75	75
Tabla 2.18: CU-17 Menú pausa.....76	76
Tabla 2.19: CU-18 Completar Juego76	76
Tabla 2.20: CU-19 Matar enemigos.....77	77
Tabla 2.21: CU-20 Conseguir experiencia y oro78	78
Tabla 2.22: CU-21 Recibir daño78	78
Tabla 2.23: CU-22 Infligir daño.....79	79
Tabla 2.24: CU-23 Curar80	80
Tabla 2.25: CU-24 Permitir al jugador utilizar conjuros80	80
Tabla 2.26: CU-25 Subir de nivel.....81	81
Tabla 2.27: CU-26 Luchar82	82

1 ESPECIFICACIÓN DEL TRABAJO

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

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

1.1 Introducción

El objetivo principal de este proyecto ha sido la creación de un remake del videojuego Dragon Quest (DQ), inicialmente lanzado al mercado para la consola Nintendo Entertainment System (NES) y para el microordenador MSX (llamado así por el intérprete de BASIC que integraba) el 27 de mayo de 1986.

Como videojuego, a pesar de su no demasiado buen recibimiento en occidente y sus evidentes carencias, se estableció como un referente de los juegos de rol (RPG). Esto unido a su banda sonora original (OST) del compositor japonés Koichi Sugiyama (すぎやまこういち *Sugiyama Kō'ichi*) y a su arte, creado por el famoso mangaka Akira Toriyama (鳥山 明 *Toriyama Akira*), creador de obras como Dragon Ball, hizo que el juego se volviese muy popular, sobre todo en sus posteriores versiones para otras consolas donde, evidentemente, solucionaban algunas de las anteriormente mencionadas carencias.

Dada su importancia y el momento de la historia de los videojuegos en el que se creó, hemos creído conveniente crear este proyecto que servirá a la vez para demostrar los conocimientos adquiridos durante los años de estudio en el grado de ingeniería informática, creando desde cero una versión del juego con funcionalidad añadida, mejorando funciones ya existentes, mejora de gráficos, de banda sonora, etc... Así como, para hacer un homenaje a uno de los videojuegos en los que muchos

de los juegos actuales y no tan actuales se inspiran, y que tanto me acompañaron de pequeño e hicieron en parte que acabase estudiando informática.

1.2 Objetivos del trabajo

El objetivo principal de este proyecto es construir desde cero una versión de DQ mejorada respecto de la original.

Para ello buscaremos cumplir con los siguientes objetivos:

1. Mejora gráfica del videojuego manteniendo el estilo pixelart.
2. Mejora del OST puesto que la original estaba limitada por la tecnología de la época siendo máquinas de 8-bits.
3. Mejora gráfica de la interfaz.
4. Hacer la interfaz más intuitiva y fácil de utilizar.
5. Hacer una interfaz más reactiva.
6. Hacer más intuitiva la forma de jugar al juego.
7. Mejorar la interacción del personaje principal con los objetos del mundo.
8. Eliminar mecánicas tediosas como el sistema de “llaves mágicas”, mediante el cual necesitas llaves que deben comprarse para abrir puertas, pero en el momento que abandonas el mapa se vuelve a cerrar la puerta y necesitas de nuevo gastar otra llave (pudiendo portar un máximo de 5 llaves).
9. Mejora en el sistema de progresión del juego.
10. Modificación de la dificultad del juego en las distintas etapas de la progresión de la historia.
11. Añadir soporte para jugar tanto con teclado como con mando.
12. Mejora del sistema de compraventa del juego.
13. Reajuste de precios para que estos se correspondan con el oro que otorgan los enemigos de un nivel similar.

- 14.** Mejora en el sistema de guardado, añadiendo la opción de hacerlo desde el menú de pausa, a diferencia del original en el que tenías que volver a determinado lugar para poder guardar, muchas veces tardando más de 10-15 minutos para poder hacerlo.
- 15.** Añadir guardado automático para evitar pérdidas de datos en situaciones en las que se produzca un error no recuperable, se apague el sistema de golpe, etc.
- 16.** Reajuste de la visibilidad en cuevas y mazmorras.
- 17.** Modificación del equipamiento para permitir llevar casco, armadura, escudo, arma y accesorio; En contraste con el original donde no se puede llevar casco.
- 18.** Añadir nuevo equipamiento (principalmente cascos y accesorios) y objetos consumibles.
- 19.** Ajuste en las estadísticas tanto del personaje principal como de los enemigos (incluyendo cantidad de experiencia y oro otorgada por los enemigos).
- 20.** Cambiar el sistema mediante el cual al morir eres transportado al inicio del juego ya que se hace extremadamente tedioso y para alguien que solo vaya a jugar un poco le hace perder el tiempo que puede dedicarle.
- 21.** Modificación y balanceo de hechizos.
- 22.** Adición de estadísticas como “poder mágico” y “defensa mágica” para diferenciar la resistencia a ataques físicos y mágicos.
- 23.** Balancear el sistema de combate en general creando funciones para cálculo de daño, daño crítico, daño mágico, crítico mágico, etc.
- 24.** Añadir diferencias entre amanecer, mañana, tarde y noche que se correspondan con la hora a la que se juegue.
- 25.** Traducir el juego completamente al español, siendo que el original no estaba disponible en este idioma. Además, que en la versión en inglés se utilizaba jerga típica de la Inglaterra del siglo XVII-XVIII haciendo bastante difícil la comprensión de algunas frases.

1.3 Antecedentes y estado del arte

En este apartado pondremos en una balanza algunos de los principales motores de videojuegos, analizando sus características, sus puntos fuertes y aquello en lo que flaquean. Realizaremos un pequeño estudio comparativo y posteriormente indicaremos cuál de ellos hemos decidido utilizar para el desarrollo de nuestro remake y porqué.

Hay que aclarar que no tendremos en cuenta, por ejemplo, si se trata de software libre o si es privativo. Nos centraremos en ver sus características y posteriormente valorar cuál de ellos es más adecuado para nuestro proyecto en concreto. Aunque sí se tendrá en cuenta que se pueda utilizar el motor como desarrollador independiente, dícese, por ejemplo, en el caso del motor FrostBite cuyo uso es exclusivo de Electronic Arts (EA), no se tendría en cuenta ya que no existe la posibilidad de que otra empresa o usuario fuera de EA lo utilice.

1.3.1 Principales motores de videojuegos

Hoy en día resulta impensable desarrollar un videojuego desde cero, sin hacer uso de un motor de videojuegos, ya que estos incluyen funcionalidades muy importantes y necesarias implementadas por defecto, como pueden ser el motor de físicas, el de sonido, etc. Y que de otro modo habría que implementar para cada juego. Haciendo uso de un motor de videojuegos nos ahorramos esta parte, lo que nos permite centrarnos en el resto de elementos de nuestro videojuego, como la parte gráfica, interacción entre objetos y demás.

Teniendo en cuenta lo que acabamos de exponer, a continuación, veremos algunos de los principales motores de videojuegos utilizados en videojuegos estos últimos años.

1.3.1.1 Unreal Engine

Este motor es probablemente el más famoso de los que vamos a ver en este breve estado del arte que estamos haciendo, junto con Unity.

Unreal Engine (Unreal) fue creado en 1998 por Epic Games (Epic) para ser utilizado en el shooter “Unreal”. Desde ese momento Unreal ha ido evolucionando hasta el recientemente publicado en abril de 2022, Unreal Engine 5 (UE5).



Ilustración 1.1: Demo técnica UE5 en Play Station 5

Muchos de los juegos triple A que se crean a día de hoy utilizan este motor como su base. Unreal se utiliza principalmente en videojuegos 3D, y entre algunas de sus principales características destacan técnicas avanzadas de renderizado como el Renderizado Dinámico de Alto Rango (HDRR), iluminación por píxel, sombras dinámicas, etc. También cuenta con oclusión ambiental con un alto nivel de personalización y configuración que nos permite dar una sensación de realismo a las imágenes, además de físicas y colisiones realistas. En esta última versión de Unreal, UE5, se han incluido dos nuevas tecnologías:

- **Nanite:** Es un sistema de geometría micro poligonal que permite cargar elementos con millones de polígonos sin mermar la capacidad de carga y pudiendo incluirlos varias veces en caso de ser necesario sin disminuir la

calidad y sin un coste de rendimiento muy superior. Aunque para aprovechar esta tecnología se necesitan de gráficas con arquitectura Nvidia Maxwell, AMD GCN o superior.

- **Lumen:** Es un sistema de iluminación de nueva generación. Permite crear escenas increíblemente realistas. Se trata de una herramienta de iluminación global.

En cuanto a la facilidad de uso hay que decir que, si bien no es sencillo, teniendo muchísimas características y personalización, es esto mismo lo que permite alcanzar unos mejores resultados, aunque por supuesto, requiere de una mayor curva de aprendizaje.

La tecnología Unreal de Epic es especialmente útil para creación de videojuegos 3D en PC, consolas y dispositivos móviles, así como crear experiencias de calidad en Realidad Virtual (RV) o Aumentada (RA).

Algunos de los principales juegos siendo desarrollados que utilizan este motor son:

- **Tomb Raider**
- **The Witcher IV**
- **Senua's Saga: Hellblade 2**
- **Mass Effect**
- **Dragon Quest XII**

1.3.1.2 CryEngine

CryEngine fue creado en 2006 por la desarrolladora de software alemana CryTek y que se utilizó por primera vez en el desarrollo del videojuego Far Cry.

La versión más actual de este motor es CryEngine V, a la espera de la ya anunciada nueva versión, CryEngine VI.

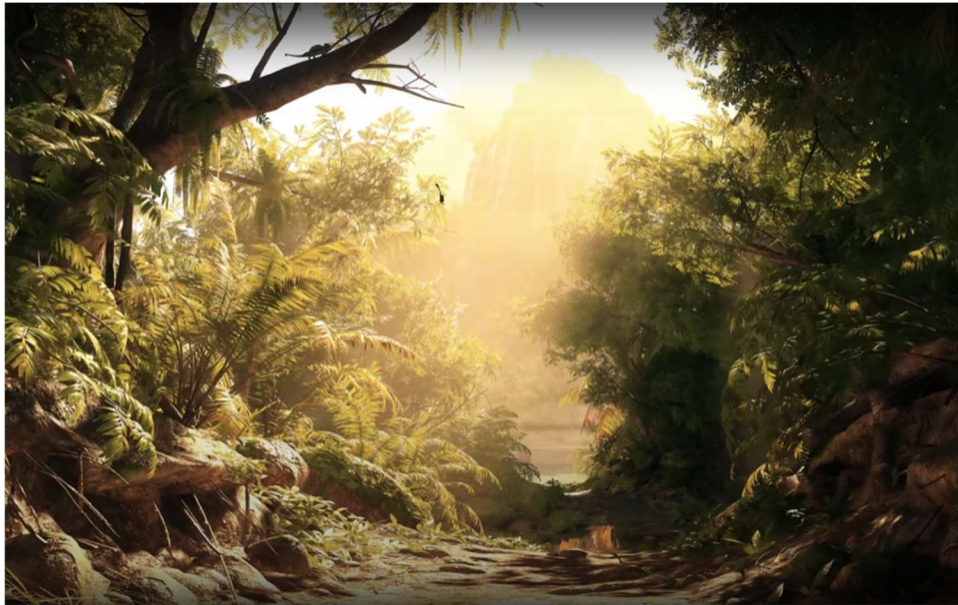


Ilustración 1.2: Juego, Hunt: Showdown

CryEngine tiene soporte HDRR al igual que Unreal, un sistema de iluminación global indirecta en cascada, destrucción dinámica, etc. Además de todo esto, posee un sistema de tratamiento de partículas y animaciones muy avanzado que da resultados impresionantemente bellos.

El motor CryEngine es habitualmente utilizado en desarrollo de videojuegos 3D para consolas, así como para PC. Aunque también permite desarrollo de aplicaciones de RV y RA.

Algunos de los principales juegos desarrollados con CryEngine son:

- **Crysis 4**
- **FarCry 7**

1.3.1.3 Amazon Lumberyard

Este motor de videojuegos para juegos triple A está basado en CryEngine. Desarrollado también por CryTek, pero cuya licencia fue otorgada a Amazon en 2015.



Ilustración 1.3: Juego MMORPG de Amazon, New World

Este motor es el único con integración nativa de Amazon Web Services (AWS) e integración nativa con Twitch como algunas de sus características principales.

Lumberyard permite crear juegos de calidad gráfica bastante buena, con capacidad de edición en tiempo real, así como funciones específicas para controlar momentos del día, el clima, etc. Lumberyard tiene la capacidad de crear videojuegos que al tener integración nativa con Twitch, permite interactuar en tiempo real con el chat de esta plataforma.

Tal como se ve, Amazon Lumberyard parece estar hecho para desarrollo de videojuegos en tiempo real, como un Juego de rol masivo en línea (MMORPG), que permiten interactuar con el público de Twitch, plataforma que en los últimos años ha triunfado mucho. Lo que le da mucho potencial de crecimiento y de mantener el público a medio-largo plazo.

Algunos de los principales juegos desarrollados o en desarrollo que utilizan el motor son:

- **New World**
- **Squadron 42**
- **Star Citizen**

1.3.1.4 GODOT

GODOT fue desarrollado en 2001 y utilizado únicamente por OKAM Studios, una empresa argentina, hasta el año 2014 cuando su código fue liberado dando acceso a otras empresas y usuarios.

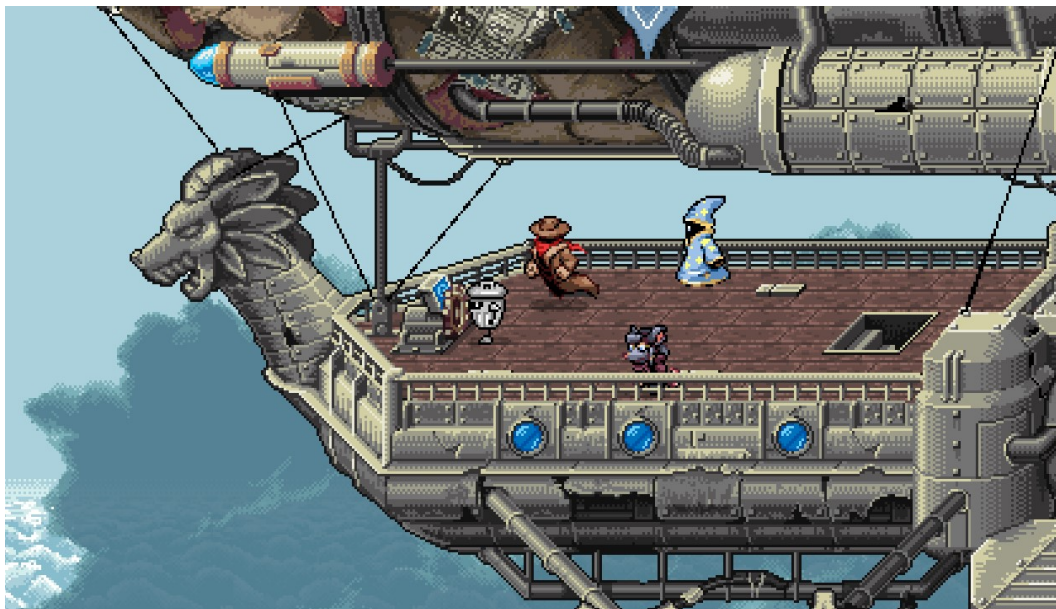


Ilustración 1.4: Juego, Kingdoms of the Dump

Permite desarrollo para una amplia variedad de plataformas:

- **Escritorio:** Windows, macOS, Linux, UWP, BSD.
- **Plataformas móviles:** iOS y Android.
- **Consolas:** Nintendo Switch, Play Station 4, Xbox One.
- **Web:** HTML5 y WebAssembly

Tiene una visual muy intuitiva que facilita la labor de aprendizaje y hace más ameno trabajar con la herramienta, así como motores dedicados tanto para creación de juegos 2D como 3D. El motor no es el más indicado si lo que queremos es crear un juego realista, pero es perfecto para juegos con una estética más cartoon o 2D. Permite la animación de prácticamente cualquier cosa con herramientas incluidas en el motor, lo que facilita el trabajo y lo mantiene más ordenado.

A diferencia de otros motores como Unreal o Unity cuyos scripts son en C++ y en C# respectivamente, en GODOT, si bien son C++, C# y VisualScript los que tienen soporte completo, se pueden utilizar más lenguajes.

También tiene soporte para desarrollo de aplicaciones de RV y RA, aunque no es su especialidad.

En los últimos años ha ido ganando popularidad, sobre todo en el terreno de los videojuegos 2D, ya que su herramienta es sencilla de entender, su motor especializado en desarrollo de videojuegos 2D que nos permite trabajar con unidades de píxeles, y su entorno 2D nativo (no existe necesidad de emular un espacio 2D en un espacio 3D) hace que merezca la pena elegirlo por sobre otros.

Algunos de los juegos más conocidos creados o en desarrollo con GODOT son:

- **Deponia**
- **Sonic Colors: Ultimate**
- **Kingdoms of the Dump**

1.3.1.5 RPG Maker

RPG Maker es muy conocido en el mundo del desarrollo de videojuegos 2D, ya que permite a prácticamente cualquiera desarrollar un juego desde cero a pesar de no saber programar, razón por la que no goza de mucha popularidad.



Ilustración 1.5: Carátula del remake para Nintendo Switch del Juego, Corpse Party

Si bien como hemos dicho lo mejor que tiene RPG Maker es permitir a casi cualquiera crear un juego desde cero, en muchos casos se queda corto ya que no tiene la misma capacidad de personalización que tendrían otros motores.

Apenas se ha llegado a utilizar en proyectos conocidos y si podemos destacar alguno sería el juego **Corpse Party**.

1.3.1.6 Unity3D

Unity3D (Unity) fue creado en 2005 por Unity Technologies y es el motor de videojuegos más popular actualmente, ya que es ampliamente utilizado, y tiene una gran comunidad de usuarios y desarrolladores detrás, que se ayudan entre sí o que comparten sus proyectos.



Ilustración 1.6: Juego, Cuphead

Hay muchos juegos de renombre que utilizan Unity, pero sobre todo se suele usar en empresas Indies o en desarrollo de juegos para dispositivos móviles. Aunque permite crear videojuegos para una cantidad ingente de plataformas:

- **Escritorio:** Windows, macOS, Linux.
- **Plataformas móviles:** iOS, Android, Windows Phone, Tizen.
- **Smart TV:** tvOS, Samsung Smart TV, Android TV.
- **Dispositivos de realidad extendida:** Oculus Rift, CardBoard de Google, HTC Vive, PlayStation VR, Samsung Gear VR, Microsoft Hololens.
- **Consolas:** Nintendo Switch, Play Station 4, Xbox One, Wii U, Wii, Nintendo 3DS.
- **Web:** WebGL

Este tiene características que lo hacen ideal en la mayoría de los proyectos que se nos puedan ocurrir. Puedes obtener unos gráficos realistas de alta calidad con una iluminación en tiempo real muy conseguida, gráficos HDRR, oclusión ambiental, etc.

Además de ser uno de los mejores motores para desarrollo en 3D, también lo es en desarrollo 2D. Tiene una gran cantidad de herramientas que facilitan la creación de juegos 2D, y Físicas especiales para videojuegos en 2D. Podemos modificar los sprites e imágenes que usemos en nuestro juego directamente desde el editor de Unity, algo que resulta muy cómodo y útil. Permite crear animaciones e interfaces de manera muy sencilla y posee una gran cantidad de librerías que hacen muy ameno el manejar todos los elementos de la escena desde los distintos scripts.

Algunos videojuegos populares creados con Unity son:

- **Temple Run**
- **Rust**
- **Pokemon Go**
- **HearthStone**
- **Cuphead**

1.3.2 Estudio de popularidad entre los principales motores

Para hacer un estudio que pueda servir para entender qué motores son más populares hemos recurrido a diversas fuentes. Como por ejemplo a PR NewsWire Donde encontramos la siguiente información de (TIGA, 2019):

Engine	Percentage of developers using engine
Unity 3D Engine (Unity Technologies)	72
Unreal Engine (Epic Games)	27
Internal proprietary engine	27
Amazon Lumberyard (Amazon)	4
Cocos2d-x (Cocos)	4
Swift	4
Corona SDK	3
CryEngine (Crytek)	3
GameMaker Studio	3
Phyre Engine (Sony)	2
PixiJS	2
playCanvas	2
Octane	1
Clickteam Fusion 2.5	1
Marmalade SDK (Marmalade)	1
Ren'PY	1
Virtygon	1

Tabla 1.1: Datos de encuesta “Game Engines used by video game developers in the UK”

Se trata de una encuesta hecha a desarrolladores de videojuegos en el Reino Unido en 2019, en la que se les preguntaba qué motor de videojuegos utilizaban

actualmente. Era una pregunta con posibilidad de elegir múltiples opciones, por ello que el porcentaje total excede el 100%.

Utilizando los datos de dicha tabla hemos construido una gráfica donde encontramos que todos los motores que explicamos con anterioridad, a excepción de GODOT, se encuentran en esta lista.

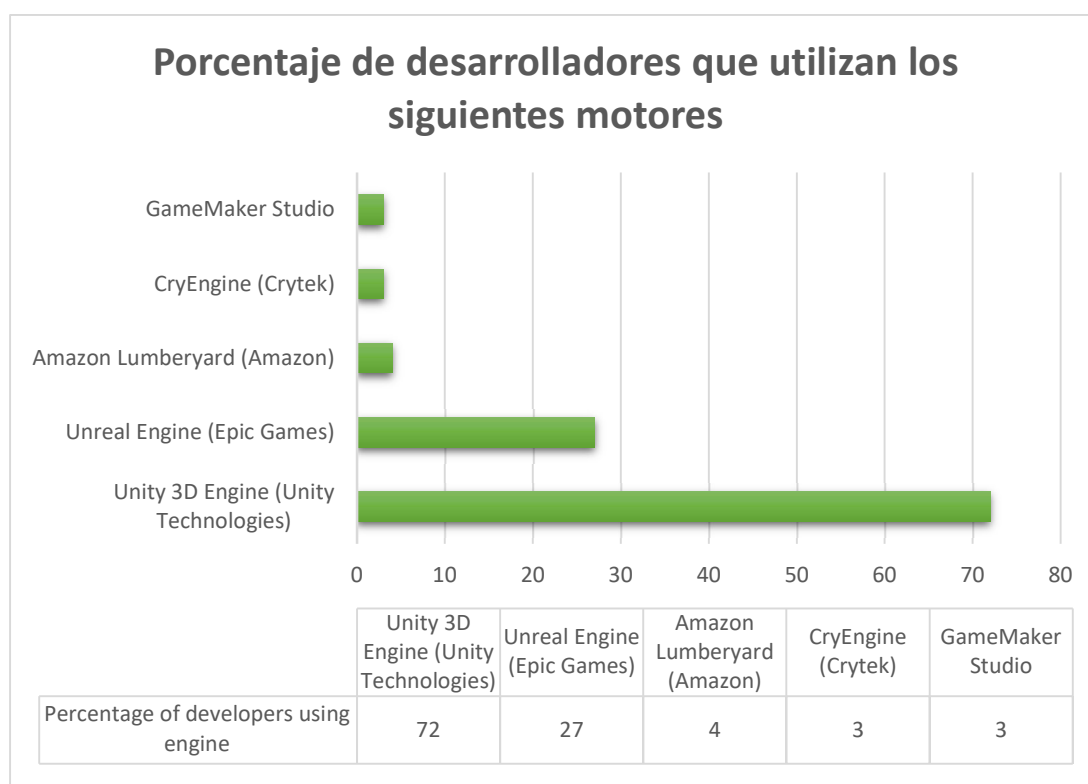


Ilustración 1.7: Gráfico, Porcentaje de desarrolladores que utilizan los siguientes motores

Se puede observar que Unity y Unreal son los más utilizados, y es normal, puesto que como dijimos, Unreal es el motor de desarrollo de videojuegos que permite crear juegos más fotorrealistas que hay actualmente y tiene muchísimas características que lo hacen atractivo para los desarrolladores.

En cuanto a Unity solo tenemos que ver los datos que nos proporciona la anterior encuesta, y también los que podemos encontrar en el apartado “Most used Engines” de Itch.io (Itch.io, 2022) sobre la cantidad de proyectos subidos que se corresponden con los diferentes motores de videojuegos:

Engine	Number of projects per Engine	Percentage
Unity 3D Engine	93793	50,8404477
Construct	24852	13,4710139
GameMaker: Studio	13267	7,19137057
GODOT	9966	5,40206521
Twine	9031	4,89524894
Bitsy	6075	3,29295065
Unreal Engine	5615	3,04360788
RPG Maker	5355	2,90267501
PICO-8	5185	2,8105266
Ren'PY	3735	2,02455484
LÖVE	1640	0,88896116
PuzzleScript	1485	0,80494349
GDevelop	1378	0,7469442
Clickteam Fusion	992	0,53771309
libGDX	835	0,45261132
TyranoBuilder	685	0,3713039
TIC-80	596	0,3230615

Tabla 1.2: Datos del sitio Itch.io, Engines más utilizados

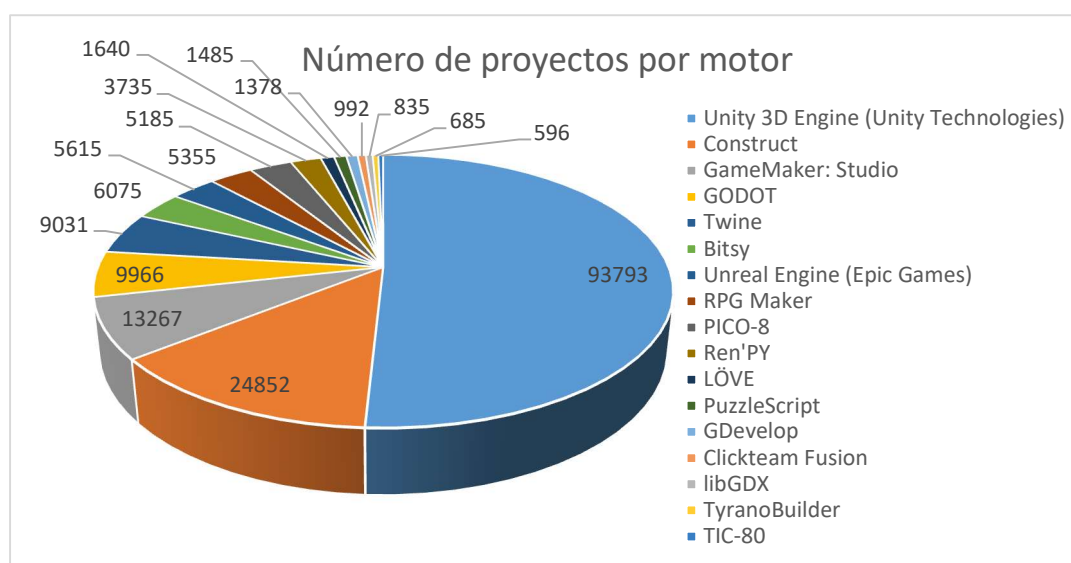


Ilustración 1.8: Gráfico, Número de proyecto por motor de videojuegos de Itch.io

Como vemos Unity sigue siendo el más usado, y a diferencia de los datos anteriores, aparece GODOT en un cuarto puesto superando a Unreal, RPG Maker, CryEngine que solo cuenta con 88 proyectos y Lumberyard que ni siquiera aparece. Hay que entender que estos datos se han obtenido de Itch.io, una página dedicada a desarrolladores y artistas indies para compartir o vender sus juegos/assets, y aquí radica la diferencia. Pero al contrastar los valores de estas dos tablas vemos que ya sea en el ámbito corporativo como en el más indie o de desarrollador independiente, Unity es con diferencia el más popular.

1.3.3 Elección de motor de videojuegos

En este punto ya hemos visto algunos de los principales motores utilizados en la creación de videojuegos, tanto a nivel de una desarrolladora Indie, como a nivel de un desarrollador más grande. Ahora solo nos queda valorar las opciones y elegir la mejor para nuestro caso.

Para elegir cuál es el motor más conveniente para nosotros, tenemos que pensar en el tipo de juego que queremos hacer y cómo lo vamos a hacer. En este caso, al tratarse de un remake del primer juego de la saga Dragon Quest, queremos que mantenga su esencia, pero darle un toque fresco y nuevo. Por tanto, lo que necesitamos es un motor que nos permita hacer un juego 2D de calidad, y los motores que mejor cumplen con esto son Unity y GODOT.

Después de meditarlo y estudiar las funcionalidades que nos aportan ambos hemos llegado a la conclusión de que la mejor opción es Unity por las siguientes razones:

1. **Unity es un motor más conocido**, lo que hace esté más pulido ya que mucha más gente lo usa, y se descubren y mejoran fallas constantemente.
2. **Tiene una comunidad muy grande de gente detrás**, lo que hace que haya más gente dedicada a publicar ya sean libros, tutoriales, problemas que les han surgido y respuestas que otros usuarios les han dado, etc.

3. Aunque GODOT tiene un motor dedicado para el desarrollo de videojuegos 2D, el motor de Unity para videojuegos 2D (si bien emula el 2D en un espacio 3D) está al nivel o incluso **supera en personalización y opciones al de GODOT** porque como hemos dicho, cuenta con muchísimos usuarios que, inevitablemente, aportan y hacen que con el paso del tiempo el motor no deje de mejorar.
4. **Experiencia previa con el entorno de Unity**, significando que ya conocemos muchas de sus funcionalidades y sabemos cómo operar con él.
5. **Al tener experiencia en Unity también la tenemos en su sistema de Scripting en C# y de manejo de entidades**, lo cual también es una ventaja frente a GODOT donde tendríamos que aprender todo esto de cero.

Estas son las principales razones por las que hemos creído conveniente utilizar Unity por sobre GODOT para la creación de nuestro proyecto. Además de razones secundarias más subjetivas como que la interfaz está más cuidada y es más entendible en Unity, entre otras.

1.4 Descripción de la situación de partida

1.4.1 Descripción del entorno actual

Actualmente ya existen diversos remakes oficiales para no tan solo la primera entrega de DQ, que es la que nos atañe, sino también para muchas otras. Aunque en todos los casos se trata de un simple lavado de cara, mejorado la calidad gráfica y mejorando lo más imprescindible, pero manteniendo muchas de las carencias y deficiencias que tenían los títulos originales.

En resumen, a pesar de que ya existen remakes para DQ en Android y en Nintendo Switch, ninguno soluciona los problemas más graves que tenía el original.

1.4.2 Resumen de las deficiencias y carencias identificadas

Vamos a diferenciar entre carencias y deficiencias, por ello, primero veremos qué es lo que le falta al juego, para posteriormente indicar qué partes del juego necesitan ser modificadas puesto que son, por supuesto, utilizables, pero deficientes.

1.4.2.1 Carencias

Como hemos dicho, los remakes actuales del juego se han centrado siempre en mejorar el apartado gráfico, pero nada más allá, por ello es que todos estos remakes tienen algunas carencias:

- A pesar de las mejoras gráficas tanto al juego como a la interfaz, esta última permanece muy similar a la original, y por similar nos referimos a que mantiene carencias que tenía la original, como bien, en el menú, no mostrar las estadísticas que te proporciona un arma o una armadura al equiparlas y teniendo que comprobarlas por ti mismo después en el apartado de Estadísticas.
- No permiten guardar la partida fuera del castillo inicial donde comienza el juego, por lo tanto tienes que volver atrás para guardar.
- No podemos saber cuanta experiencia nos falta para subir de nivel, a no ser que, al igual que para guardar la partida, volvamos al punto inicial y hablemos con el Rey. Y esto, en niveles avanzados donde tardas bastante en subir de nivel se hace bastante pesado porque nunca sabes el tiempo que te falta para subir de nivel.

1.4.2.2 Deficiencias

Ahora que ya hemos visto algunas de las carencias más importantes que tienen tanto DQ original, como sus remakes, veremos las deficiencias que persisten hasta el día de hoy y que necesitan cambiarse:

- El menú tiene un problema, y es que no existe un apartado de equipo.

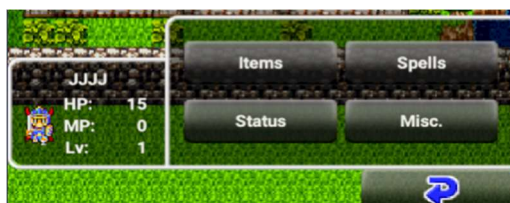


Ilustración 1.9: Gráfico, Captura del remake de DQ para Android

Como vemos en la imagen tenemos cuatro apartados. Uno pensaría que dentro del inventario podríamos encontrar la opción para equiparnos con nuestras armas y armaduras, pero no es así. Simplemente se nos muestran todos los objetos, sin diferenciar unos de otros, y en muchos casos no es fácil saber qué es un objeto equipable, y qué no. El apartado de equipo se encuentra dentro del apartado de Miscelánea, el mismo donde se encuentran las opciones del juego, cosa que no tiene mucho sentido y, por tanto, que necesita cambiarse.

- Otro problema bastante grande es que los precios de las tiendas no están para nada balanceados con respecto al oro que otorgan los enemigos. Al principio los enemigos otorgan en torno a 3 de oro de media, pero para comprar la peor arma del juego (a parte de la inicial), o una armadura (que ni siquiera portamos al principio), tenemos que matar muchísimos enemigos. Esto sin tener en cuenta el tener que volver a la ciudad a curarnos cada pocas batallas y tener que pagar para hacerlo, ya que si morimos perdemos aún más dinero. Y cuando estamos en etapas posteriores del juego nos sobra mucho oro.
- Los enemigos a lo largo del juego necesitan un cambio en sus estadísticas. Al inicio son extremadamente difíciles, pudiendo ser que el primer enemigo que te encuentres en el juego te mate con un poco de mala suerte, esto asumiendo que te hayas equipado el arma que tienes en el inventario, si no te has dado cuenta de que la tienes y te enfrentas a uno, no quitarás nada de vida (literalmente cero). Y los enemigos del final del juego, incluyendo al jefe final son demasiado fáciles en relación con como escalan las estadísticas del jugador.
- En el juego hay un sistema, como ya indicamos anteriormente en el apartado 1.2 de 'objetivos del trabajo', en el que se utilizan unos objetos

llamados “llaves mágicas”. Estas llaves sirven para abrir las puertas que nos encontramos a lo largo del juego y que son imposibles de abrir de otro modo. Las llaves no se encuentran, tienes que comprarlas y solo puedes llevar un máximo de cinco unidades. El problema es que cuando tú abres una puerta, esta no se mantiene abierta para la próxima vez que vayas, si no que cuando salgas del mapa habrás perdido tu llave y la puerta se habrá cerrado de nuevo. Al poder llevar solo 5 llaves es muy tedioso, por tanto, vamos a eliminar esta mecánica defectuosa / tediosa.

- Hay un problema a la hora de hablar con un Non-Player Character (NPC) o de recoger algún objeto / abrir un cofre, y es que, si resulta haber más de un NPC a tu lado, o un NPC y un cofre, por ejemplo, aunque tú estés mirando a un NPC (o al cofre), existe la posibilidad de que se gire para hablar con el otro (o para recoger el cofre), a pesar de que tú estés mirando en otra dirección. Esto ocurre porque el juego está implementado de tal forma que simplemente con estar al lado de un NPC ya puedes hablarle, no es necesario que estés mirándolo.

Todas estas cosas y más, pero de menor importancia, son problemas que han persistido a pesar de los diversos remakes que ha recibido el juego, y problemas que trataremos de solucionar con nuestra versión del juego.

1.5 Requisitos iniciales

En este apartado solo indicaremos los requisitos principales a alto nivel sin entrar en detalle, ya que la descripción completa de requisitos funcionales y no funcionales la haremos en apartados posteriores.

Nuestro proyecto **debe** cumplir con los siguientes requisitos:

- Crear una partida nueva.
- Cargar una partida.
- Guardar el estado de una partida (esto incluye inventario, nivel de historia, lugar, etc.).

- Permitir al jugador moverse por, y, entre escenas.
- Permitir al jugador hablar con los NPCs del juego.
- Permitir al jugador comprar en la tienda.
- Permitir al jugador vender en la tienda.
- Permitir al jugador conseguir objetos.
- Permitir al jugador utilizar objetos.
- Permitir al jugador equipar objetos.
- Permitir al jugador desequipar objetos.
- Permitir al jugador eliminar objetos.
- Permitir al jugador completar misiones.
- Permitir al jugador subir de nivel.
- Permitir al jugador luchar contra enemigos.
- Permitir al jugador utilizar conjuros.
- Permitir al jugador poder atacar.
- Permitir al jugador usar objetos en batalla.
- Permitir al jugador huir.
- Permitir al jugador usar el menú de pausa.
- Permitir al jugador completar el juego.
- Permitir al jugador matar enemigos.
- Permitir al jugador conseguir oro y experiencia.
- Permitir al jugador recibir daño.
- Permitir al jugador infligir daño.
- Permitir al jugador curarse.
- Permitir a los enemigos poder atacar.
- Permitir a los enemigos recibir daño.

- Permitir a los enemigos infligir daño.

Nuestro proyecto **debería** de cumplir los siguientes requisitos:

- Colocar al jugador en la última ciudad visitada en caso de morir.
- Crear autoguardado para perder pérdidas de información graves.
- Animar batallas.
- Permitir jugar con mando y con teclado.

1.6 Alcance

El proyecto se entregará únicamente una vez se haya completado su desarrollo. A la hora de la entrega del proyecto, este tendrá la capacidad, una vez creado el ejecutable, de correr una versión remake de DQ.

En ella podremos crear una nueva partida o continuar con la anterior, controlaremos al protagonista que se moverá por los escenarios del mundo de Alefgard (nombre del mundo en el videojuego original) y que podrá enfrentarse a los mismos enemigos que en el juego original (41 monstruos diferentes) por medio de un sistema de combate mejorado (por la adición de dos estadísticas nuevas que no aparecen en el original, el poder mágico y la defensa mágica que ayudarán a balancear los ataques y, resistencia físicas y mágicas de los enemigos y la propia del jugador, así como por la adición de nuevas fórmulas para el cálculo de daño), pero que implementa las mismas opciones que el original, en resumen, tendremos la posibilidad de atacar, utilizar objetos del inventario, usar magias y huir.

El personaje principal podrá entablar conversación con los NPCs, así como comprar en las diferentes tiendas de los diferentes pueblos del juego, que también coinciden con los del juego original.

Se incluirán un surtido de objetos y equipamiento mayor y, diferente en algunos casos del original, también se añadirá la posibilidad de llevar una pieza de equipamiento adicional, el casco.

Se presentará un menú de pausa con opciones para inventario, donde podremos ver los objetos y utilizar, equipar, desequipar o desechar los objetos. Además del inventario, en el menú de pausa también podremos acceder a las estadísticas de nuestro personaje, a los conjuros, y al apartado de Miscelánea donde podremos guardar la partida y salir del juego.

La narrativa y eventos del juego serán los mismos que en el original, si bien el juego será traducido completamente al español.

1.7 Hipótesis y restricciones

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

Aunque en este proyecto se ha tenido en cuenta esta restricción, para solucionar los bugs y problemas que aparecían al ir implementando las diferentes funcionalidades de nuestro proyecto se ha dedicado un mayor número de horas en su desarrollo.

El proyecto se ha desarrollado completamente en un equipo con Windows 10 y la plataforma objetivo para la que se ha desarrollado también ha sido Windows, por tanto, no consideraremos el equipo como una limitación.

Decidimos utilizar Unity como motor para el desarrollo de nuestro juego, pero dado que la versión Individual, ya sea la personal o la de estudiante es gratuita, no consideraremos el costo del motor como una posible limitación.

En el caso de los sprites y los assets para crear el apartado visual de nuestro remake, sí podría ser considerada una limitación económica, pero en nuestro caso los extrajimos de la versión remake de DQ, DQ II y DQ III para la consola Nintendo Switch por lo que no son limitantes. Cabe recalcar que no poseemos los derechos de ninguno

de estos assets y sprites, y que solo estamos haciendo uso de ellos con un propósito académico.

1.8 Tecnologías utilizadas

Para llevar a cabo nuestro proyecto hemos utilizado diferentes tecnologías. Para la construcción de nuestro mundo y manejo de todos los elementos del juego hemos utilizado Unity. Por tanto, en cuanto a lenguajes de programación hemos utilizado única y exclusivamente C#, el lenguaje de Scripting que utiliza Unity.

En cuanto a las librerías que hemos utilizado fueron, por un lado, las propias de Unity para la creación de los managers de cada escena, así como para los scripts de movimiento de los personajes y detección de colisiones. Estas han sido:

- **SceneManagement**, para controlar la transición entre escenas o el acceso a datos como nombres de escena, entre otros.
- **InputSystem**, para agregar la posibilidad de jugar con mando.
- **UI**, para manejo de los componentes de la interfaz.

A parte de estas librerías de Unity, hemos utilizado **System.Collections**.

También, para la creación del sistema de guardado hemos utilizado **System.IO** y **System.Runtime.Serialization.Formatters.Binary**. Este último para cifrar los ficheros y hacer que no se puedan modificar fácilmente.

No solo hemos utilizado Unity para hacer nuestro proyecto. Ya que nos propusimos mejorar la interfaz, y hacerla más intuitiva y nueva, hicimos uso de Adobe Photoshop 2021 (Photoshop) para la creación de todos los elementos de nuestra interfaz.

1.9 Estudio de alternativas y viabilidad

En el punto anterior dijimos las tecnologías que utilizamos para desarrollar nuestro proyecto, ahora comentaremos brevemente las diferentes alternativas que barajamos antes de decantarnos por unas u otras.

En cuanto a los motores de videojuegos, como vimos en el apartado 1.3 de este documento, tuvimos en cuenta Unreal Engine, RPG Maker, GODOT, CryEngine, Amazon Lumberyard y Unity. Para valorarlos primero decidimos elegir si nuestro proyecto sería un proyecto 3D o 2D. Pero al ser un remake para DQ, quedaba claro desde el inicio que sería 2D y no consideramos la opción de hacerlo 3D, lo que en un principio ya decantaba más la balanza hacia aquellos motores más especializados en apartados 2D. Además de esto valoramos su popularidad, cantidad de material disponible (material de aprendizaje) y lenguaje, por lo que elegimos Unity.

Para programas de edición de imágenes valoramos si utilizar GIMP o Photoshop. Ambas son viables, pero decidimos utilizar Photoshop.

Una vez entrados en el proyecto también valoramos cómo hacer el manejo de eventos y la interacción entre el personajes y resto de elementos del juego. Decidimos realizarlo todo por medio de código en vez de utilizar algunas de las funciones y elementos que Unity proporciona en sus librerías (como podría ser, por ejemplo, `OnTriggerEnter`).

También para el control por mando, el cual podía hacerse sin escribir código, por medio de una de las herramientas de Unity, decidimos hacerlo por medio de código, ya que nos permitía controlarlo más fácilmente y permitirnos personalizar la cadencia de los clics, cuándo contar como que se ha pulsado un botón, etc.

1.10 Descripción de la solución propuesta

Al final decidimos quedarnos con Unity por razones ya mencionadas. Tenemos experiencia con él, conocemos su funcionamiento y su lenguaje de Scripting (C#), hay muchísimo material tanto escrito como multimedia sobre este motor, etc. Además de todo esto, como queremos hacer un remake que mantenga la esencia del original, teníamos claro que utilizaríamos un entorno 2D, lo cual también es una ventaja para Unity al compararla con sus competidores más grandes como puede ser Unreal que destaca sobre todo en el apartado 3D.

Para decidir el editor de imágenes tampoco tuvimos muchos problemas ya que también teníamos experiencia previa con el uso de Photoshop. Además de esto, de forma subjetiva, Photoshop es más intuitivo, elegante y fácil de utilizar.

En un principio no teníamos pensado hacer el juego compatible con mando, pero pensando en que hoy en día, incluso en PC, la mayoría de los juegos se pueden jugar con mando, decidimos incluirlo. Creemos que fue buena decisión, porque nunca está de más dar opciones a los jugadores.

1.11 Metodología de desarrollo de software

Una de las primeras cosas que nos planteamos antes de realizar nuestro proyecto fue qué metodología emplear. Lo primero que hicimos ante esta disyuntiva fue comparar primeramente metodologías tradicionales con metodologías de desarrollo ágiles.

Las metodologías tradicionales siguen un enfoque más predictivo y en ellas solo hacemos la estimación de requisitos una única vez. El problema que plantea esta situación es que para hacer esto debemos saber medir muy bien estas estimaciones.

Las metodologías de desarrollo ágiles son más flexibles y pueden producirse cambios en cualquier momento. Esto es así porque al contrario que en las tradicionales, aquí lo esperado es que surjan imprevistos de cualquier tipo.

En un proyecto como el nuestro no podemos estar seguros desde el principio de cómo va a ir el desarrollo. No podemos asegurar que no nos van a surgir problemas y que el proyecto no se vaya a retrasar. Por este motivo es que es más sensato seguir una metodología ágil.

Sabiendo esto, y valorando nuestras opciones acabamos decidiendo utilizar una metodología ágil. En nuestro caso hemos elegido la metodología Scrum.

Creemos que Scrum es ideal para nuestro proyecto ya que podemos dividir la creación de este en distintas etapas, como en la metodología incremental, y en cada

sprint añadir nueva funcionalidad (nuevas etapas), resolviendo los problemas que aparezcan.

1.12 Análisis de riesgos

Dado que ya teníamos experiencia trabajando con el motor de videojuegos escogido, sabíamos que cumpliría con las expectativas propuestas, al menos en lo que se refiere a herramientas que nos permitan crear nuestro remake, por tanto, no supondría un riesgo para el proyecto la hipotética incapacidad del motor para cumplir con lo que se le pide.

Aunque sí existe la posibilidad de que al ejecutar en la herramienta de Unity el proyecto, si se produce un error fatal, ya sea por culpa de un problema en el código, o por culpa de la herramienta, llegar a perder el trabajo hasta el último guardado realizado. Por experiencia propia en otros proyectos, puede tener un impacto de entre 10 minutos a 1 hora de pérdida de trabajo. Aunque si aumentamos la frecuencia con la que guardamos los cambios (a pesar de que lo que estemos haciendo no lo hayamos completado), podemos reducir el impacto.

Además, también podría corromperse el proyecto si el programa cierra de forma abrupta por un fallo del tipo que acabamos de comentar. En este caso el problema sería irremediable y perderíamos todo el trabajo hasta el momento, o la mayoría exceptuando scripts.

En ambos casos mencionados, la probabilidad de ocurrencia es muy baja, ya que Unity suele ser bastante estable.

Teniendo en cuenta que esto puede ocurrir, y a pesar de que la probabilidad de que esto ocurra sea baja, para la realización del proyecto hemos usado un sistema (que ya teníamos previo al proyecto), para realizar copias de seguridad en la nube con cada cambio en los archivos de la carpeta del proyecto, reduciendo al mínimo el impacto de un fallo de los tipos arriba mencionados.

1.13 Estimación del tamaño y esfuerzo

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

Para este proyecto utilizaremos una estimación por COCOMO clásico. Este es un modelo algorítmico que intenta realizar una estimación de esfuerzo y tiempo estableciendo una relación matemática.

Este método define tres modos de desarrollo distintos según el tipo de proyecto:

- **Orgánico:** Son proyectos relativamente pequeños de menos de 50 Kilo Líneas de Código (KLDC), experiencia en proyectos similares y entorno estable.
- **Semi-acoplado:** Proyecto de mayor tamaño, entre 50 y 300 KLDC, proyectos con menor cantidad de experiencia y restricciones intermedias.
- **Empotrado:** Proyecto complejos, donde apenas se cuenta con experiencia, con muchas restricciones y muy volátiles.

A su vez, también existen tres modelos diferentes de COCOMO:

- **Básico:** Se basa únicamente en las LDC.
- **Intermedio:** Además del tamaño en LDC, también se tienen en cuenta un conjunto de medidas dependientes del proyecto en concreto.
- **Avanzado:** Incluye lo que el intermedio, pero también se tiene en cuenta el impacto de cada conductor de coste en el desarrollo de las distantes fases del proyecto.

En nuestro caso estimaremos el esfuerzo haciendo uso de un COCOMO Orgánico Intermedio.

Utilizaremos las siguientes ecuaciones para calcular Esfuerzo, Tiempo y Personal respectivamente:

$$E = a * KLDC^b * FAE \quad (1.1)$$

$$T = c * E^d \quad (1.2)$$

$$P = \frac{E}{T} \quad (1.3)$$

Proyecto Software	a	b	c	d
Orgánico	3.2	1.05	2.5	0.38
Semi-acoplado	3.0	1.12	2.5	0.35
Empotrado	2.8	1.20	2.5	0.32

Tabla 1.3: Tipos de proyectos software y valores según COCOMO

En esta tabla aparecen las constantes utilizadas en las anteriores ecuaciones matemáticas.

Primeramente, necesitamos calcular nuestras KLDC, para ello, resolvemos la siguiente ecuación:

$$KLDC = \frac{LDC}{1000} = \frac{15000}{1000} = 15 \text{ KLDC} \quad (1.4)$$

Hemos asumido un total de 15 KLDC. Ahora que tenemos las KLDC, solo nos falta calcular el Factor de Ajuste de Esfuerzo (FAE), para ello, tenemos que multiplicar

los 15 valores que correspondan de la siguiente tabla. Son los que se encuentran marcados:

Conductores de Coste	Valoración					
	Muy bajo	Bajo	Nominal	Alto	Muy Alto	Ext.
Fiabilidad requerida del Software	0.75	0.88	1	1.15	1.40	-
Tamaño de la base de datos	-	0.94	1	1.08	1.16	-
Complejidad del producto	0.70	0.85	1	1.15	1.30	1.65
Restricciones del tiempo de ejecución	-	-	1	1.11	1.30	1.66
Restricciones del almacenamiento principal	-	-	1	1.06	1.21	1.56
Volatilidad de la máquina virtual	-	0.87	1	1.15	1.30	-
Tiempo de respuesta del ordenador	-	0.87	1	1.07	1.15	-
Capacidad del analista	1.46	1.19	1	0.86	0.71	-
Experiencia en la aplicación	1.29	1.13	1	0.91	0.82	-
Capacidad de los programadores	1.42	1.17	1	0.86	0.70	-
Experiencia en S.O. utilizado	1.21	1.10	1	0.90	-	-
Experiencia en el lenguaje de programación	1.14	1.07	1	0.95	-	-
Prácticas de programación modernas	1.24	1.10	1	0.91	0.82	-
Utilización de herramientas software	1.24	1.10	1	0.91	0.83	-
Limitaciones de planificación del proyecto	1.23	1.08	1	1.04	1.10	-

Tabla 1.4: Conductores de Coste para COCOMO

$$FAE = 0.88 * 0.85 * 1 * 1.07 * 1 * 1.13 * 1 * 1 * 1 * 0.91 * 1.08 \approx \mathbf{0.89} \quad (1.5)$$

Justificación de los valores:

- **Atributos Software:**

- **Fiabilidad Requerida del Software:** Se refiere a la capacidad de pérdidas que puede generar un fallo en el sistema que estamos desarrollando. En nuestro caso puede provocar pérdidas menores de información ya que guardamos los datos con frecuencia para evitar situaciones de este tipo.
- **Tamaño de la base de datos:** Apenas necesitamos almacenamiento, puesto que los únicos datos que se generarán a parte de los que ocupa el proyecto base son los archivos de guardado.
- **Complejidad del producto:** Nuestro juego no necesita realizar cálculos demasiado complejos, por lo tanto, tiene baja complejidad y no es un software muy exigente.

- **Atributos de Hardware:**

- **Restricciones del tiempo de ejecución:** No se exige un rendimiento especialmente alto.
- **Restricciones del almacenamiento principal:** No hay restricciones de este tipo.
- **Volatilidad de la máquina virtual:** Nuestro remake está pensado para su uso en Windows. Usar emuladores en otros sistemas como macOS podría incrementar la volatilidad.
- **Tiempo de respuesta del ordenador:** Debe de reflejar las acciones del usuario de manera inmediata, por ello requerimos de un tiempo de respuesta bajo (Valoración alta).

- **Atributos del personal:**

- **Capacidad del analista:** Tenemos algo de experiencia con el desarrollo de videojuegos, pero no la suficiente como para ponernos por encima del punto intermedio.
- **Experiencia en la aplicación:** No tenemos experiencia con aplicaciones de una envergadura similar o superior, por tanto, nos damos una valoración baja en este aspecto.
- **Capacidad de los programadores:** Nos daremos una valoración neutra en este apartado ya que sabemos programar, pero por supuesto, no a nivel experto.
- **Experiencia en S.O. utilizado:** Estamos utilizando Windows tanto como S.O. objetivo, como para el desarrollo. Y es nuestro sistema principal desde hace muchos años.
- **Experiencia en el lenguaje de programación:** En Unity todos los Scripts se hacen en C#, lenguaje en el que no tenemos demasiada experiencia, pero es similar a otros lenguajes que sí que dominamos más, por tanto, y como en su mayor parte estamos utilizando funciones y clases específicas de Unity con las que sí que tenemos algo de experiencia, nos daremos una nota neutra.
- **Atributos del proyecto:**
 - **Prácticas de programación modernas:** A lo largo del proyecto se usarán prácticas de programación tradicionales.
 - **Utilización de herramientas Software:** Utilizamos herramientas como Photoshop y Unity principalmente. Contamos con una experiencia superior a la media.
 - **Limitaciones de planificación del proyecto:** Como ya comentamos antes, tenemos la restricción del tiempo, pero nada más allá de eso.

Ahora pasaremos a calcular el esfuerzo utilizando la ecuación indicada arriba:

$$E = a * KLDC^b * FAE = 3.2 * 15^{1.05} * 0.89 = \mathbf{48.91 \text{ personas/mes}} \quad (1.6)$$

Después de calcular el esfuerzo, pasaremos a calcular el tiempo de desarrollo estimado:

$$T = c * E^d = 2.5 * 48.91^{0.38} = 10.96 \approx \mathbf{11 \text{ meses}} \quad (1.7)$$

También calcularemos la productividad:

$$PR = \frac{LDC}{E} = \frac{15000}{48.91} = 306.68 \approx \mathbf{307 \text{ LDC} * \text{mes/persona}} \quad (1.8)$$

Y, por último, también calcularemos el personal promedio necesario:

$$P = \frac{E}{T} = \frac{48.91}{10.96} = 4.46 \approx \mathbf{5 \text{ personas}} \quad (1.9)$$

Según este estudio necesitaríamos un total de 5 personas trabajando cerca de 10 meses (no 11, ya que 11 eran con 4.46 personas; 9.78 meses con 5 personas). Pero solo disponemos de mí, el propio alumno que está haciendo el trabajo y tenemos un tiempo de aproximadamente 2 meses, por tanto, habría que pasar de 307 LDC/mes a:

$$307 * \frac{T}{\text{meses}} * \frac{P}{\text{personas}} = 307 * \frac{11}{2} * \frac{5}{1} = \mathbf{8443 \text{ LCD} * \text{mes/persona}} \quad (1.10)$$

1.14 Planificación temporal

Como hemos explicado, en el desarrollo de nuestro remake utilizaremos una metodología Scrum.

En este apartado nos limitamos a mostrar una planificación básica mediante el uso de diagramas de Gantt. En ellos tendremos en cuenta los fines de semana, pero no los festivos, ni el tiempo extra dedicado a solventar problemas surgidos al añadir nuevas características.

La etapa de desarrollo se explicará en mayor profundidad en puntos posteriores, donde veremos cada una de las etapas de desarrollo por las que ha pasado nuestro proyecto.

1.14.1 Diagrama de Gantt – Diciembre 2021



Ilustración 1.10: Diagrama de Gantt del proyecto – Diciembre 2021

A finales de diciembre, una vez terminadas las clases del primer cuatrimestre comenzamos a plantear nuestro proyecto.

Buscamos material por la web e información, jugamos a las distintas versiones del juego que hay, vimos vídeos sobre los remakes y sobre el juego original, etc.

1.14.2 Diagrama de Gantt – Enero 2022

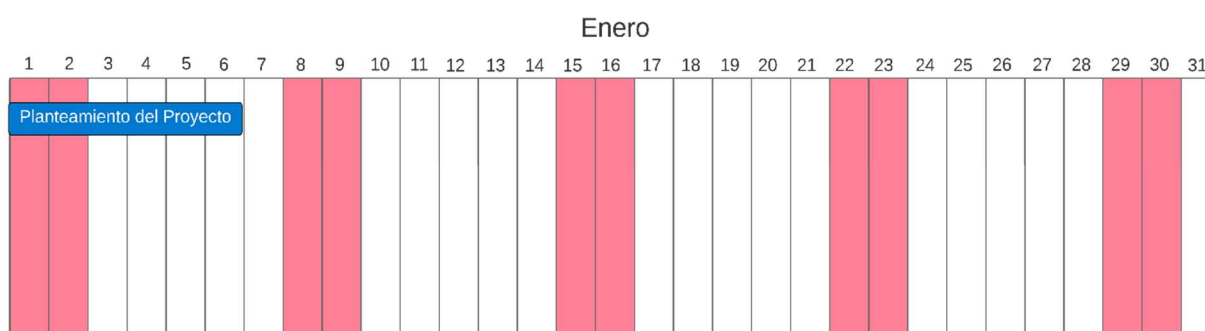


Ilustración 1.11: Diagrama de Gantt del proyecto – Enero 2022

No fue hasta poco antes del comienzo de los exámenes de enero que terminamos de plantear el proyecto. A parte de lo que hicimos en diciembre, en enero

empezamos a pensar en la metodología a aplicar en nuestro proyecto y, una vez supimos que seguiríamos una metodología Scrum, a plantearnos las diferentes etapas/sprints.

1.14.3 Diagrama de Gantt – Julio 2022

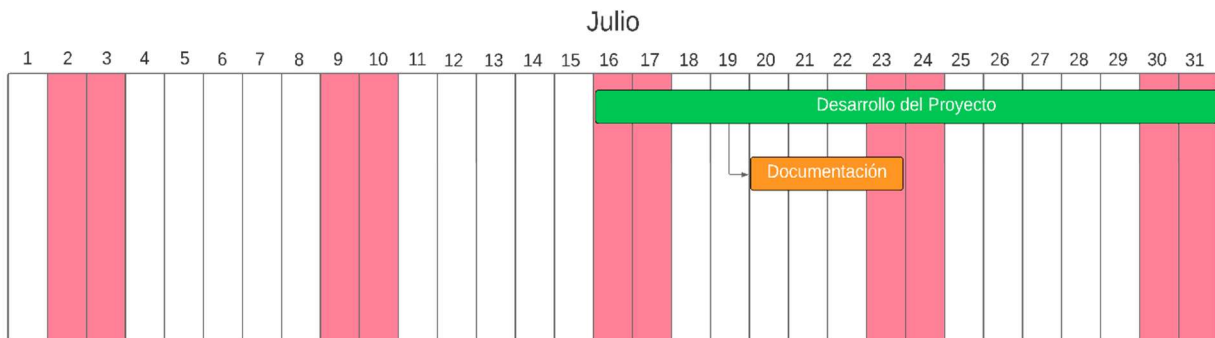


Ilustración 1.12: Diagrama de Gantt del proyecto – Julio 2022

En Julio comenzamos con el desarrollo del proyecto, y a su vez, comenzamos a rellenar ciertas partes de la documentación.

1.14.4 Diagrama de Gantt – Agosto 2022



Ilustración 1.13: Diagrama de Gantt del proyecto – Agosto 2022

En agosto continuamos con el desarrollo del proyecto, y a finales de este mes, según se va iba acercando el límite de plazo empezamos a rellenar más de la documentación.

1.14.5 Diagrama de Gantt – Septiembre 2022

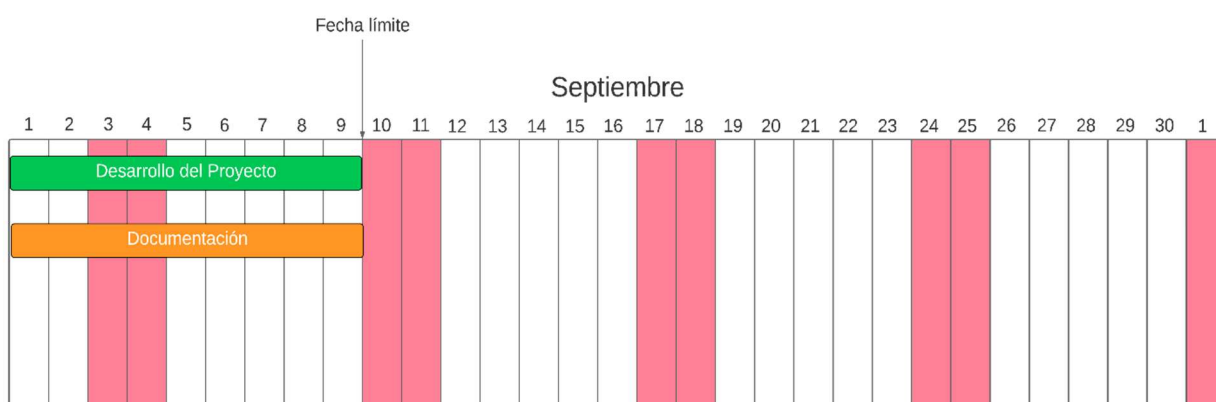


Ilustración 1.14: Diagrama de Gantt del proyecto – Septiembre 2022

En el mes de septiembre nos dedicamos a terminar el desarrollo del proyecto, así como la documentación para su entrega antes de la fecha límite.

1.15 Presupuesto

Para hacer un estimado del coste total del proyecto tendremos en cuenta tanto el coste de las horas de trabajo de los implicados, el coste del equipo, costes de servicios y materiales.

En el caso de la utilización de software gratuito como lo ha sido la versión de Unity con la que hemos desarrollado el proyecto, se descontará del total.

Para el cálculo del coste del personal, es decir, las horas que se han trabajado en total, hemos ido haciendo un seguimiento donde hemos ido apuntando el número de horas dedicadas para llegar a cada uno de los sprints con todo al día. No separaremos por costes de cada sprint, pero hemos llevado la cuenta del tiempo trabajado y hemos hecho un cálculo aproximado de 500 horas totales repartidas en 5 meses distintos.

Para realizar el cálculo del coste que supone el salario de un ingeniero informático trabajando en el desarrollo de un proyecto software como es nuestro caso recurrimos al “Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública” (BOE, s.f.). En nuestro caso se aplicaría El nivel 1 del grupo D.

A parte del coste por salario también debemos tener en cuenta el coste de la oficina o estudio donde trabajemos, así como el precio mensual de todos los servicios, donde se incluye desde el precio mensual por tener acceso a Photoshop, hasta el precio de luz y agua.

No podemos tampoco olvidarnos de los costes materiales de todo el equipo que usamos para el desarrollo de nuestro proyecto (asumiremos amortización a 5 años):

- **Equipo:**
 - Portátil (1800€):
 - Intel Core i7 10750H – 6 núcleos – 2.6 Ghz
 - 32 GB DDR4 3200 MHz
 - Nvidia RTX 2070
 - SSD 256 GB (Windows)
 - SSD 2TB
 - Xiaomi Mi Pad 5 (400€)

Producto	Importe	Duración	Total
Adobe Photoshop	24.19€/mes	4 meses	96.76€
Alquiler estudio	450€/mes	4 meses	1800€
Salario	17.715.65€/año	4 meses	5905.22€
Equipo	2200€/5 años	4 meses	146.67€
Luz	75€/mes	4 meses	300€
Agua	10€/mes	4 meses	40€
Internet	40€/mes	4 meses	160€
Total			8438.65€

Tabla 1.5: Costes asociados al proyecto

Como vemos, el coste total del proyecto sería de aproximadamente 8450€.

2 DISEÑO INICIAL

Dado que estamos haciendo uso de una metodología ágil como es Scrum, en este apartado simplemente daremos una descripción general del diseño previo a la primera iteración de nuestro desarrollo, que se verá en mayor profundidad en la parte de “Desarrollo” de este documento.

2.1 Especificaciones del sistema

En este subapartado hablaremos de los requisitos funcionales y no funcionales, de los cuales en el siguiente subapartado hablaremos en mayor profundidad.

2.1.1 Requisitos Funcionales

Los requisitos funcionales representan aquello que se nos permite hacer con el proyecto. Deben ser claros para que no haya disputas entre cliente y desarrollador:

- Se debe poder comenzar una nueva aventura.
- Debemos poder cargar una partida anterior.
- Debemos poder guardar la partida, para conservar el progreso.
- El jugador debe poder moverse por los mapas del juego.
- Debemos poder interactuar con los NPCs del juego.
- Se nos debe permitir comprar equipo y objetos en las distintas tiendas que tendremos en el juego.
- También debemos poder vender aquello que no necesite.
- Obtener objetos o equipo de otras fuentes, como por ejemplo cofres o armarios.
- Debemos poder utilizar objetos consumibles.
- Debemos poder equiparnos con armas y armaduras, así como con accesorios.

- Debemos poder desequiparnos algunos objetos o equiparnos otros si lo deseamos.
- Debemos poder eliminar objetos si no los queremos.
- Debemos poder completar misiones de la historia.
- Debemos poder acceder a un menú de pausa con distintas opciones para inventario, magia, guardar la partida y demás.
- Debemos poder librar batallas contra los enemigos.
- Debemos poder Atacar a los enemigos.
- Debemos poder Usar objetos en el combate.
- Debemos tener la posibilidad de intentar huir de un combate.
- Debemos poder acabar con los enemigos y conseguir oro y experiencia.
- Debemos poder subir de nivel.
- Debemos poder curarnos.
- Debemos poder usar conjuros.
- Debemos poder completar el juego.

2.1.2 Requisitos no funcionales

Los requisitos no funcionales hacen especifican criterios utilizados para valorar un sistema, también se los conoce como atributos de calidad de un sistema y suelen ser condiciones o restricciones impuestas por el cliente. En nuestro caso definimos los siguientes requisitos no funcionales:

1. El remake debe tener una interfaz intuitiva usando una interfaz gráfica fácil de entender para el usuario.
2. Los datos de guardado deben estar cifrados para dificultar su modificación.
3. El juego resultante tiene que ser usable.
4. El juego debe ser fiable y reducir al mínimo las posibles pérdidas de información.

5. Desacoplar el código en la medida de lo posible.
6. Reducir al mínimo la complejidad del código dando nombres simples y entendibles para facilitar el mantenimiento.
7. El juego debe ser fácil de entender para el usuario.
8. El juego debe ser compatible con distintos tamaños de pantalla.

2.2 Análisis y diseño del sistema

En esta parte vamos a ver los distintos casos de uso que podemos tener en nuestro proyecto, así como los diagramas de clases de una escena y mostraremos imágenes de la interfaz del juego en el que nos basamos para crear nuestra interfaz (algo de lo que hablaremos con mayor detenimiento y detalles en el apartado correspondiente de la parte de desarrollo). En cuanto a los diagramas de clases, ya que cada escena tendrá sus particularidades y distintas interacciones, veremos un resumen de las diferentes clases principales que pueden aparecer y como interactúan entre ellas en subapartados diferenciados.

2.2.1 Casos de uso

A continuación, se presentan los casos de uso para nuestro proyecto. Hemos usado como referencia el ejemplo que se nos muestra en (Andalucía, s.f.):

Casos de Uso			
CU-01	Crear una partida nueva.	CU-14	Atacar
CU-02	Cargar una partida.	CU-15	Usar objetos batalla
CU-03	Guardar partida.	CU-16	Huir
CU-04	Mover al jugador	CU-17	Menú pausa
CU-05	Entablar diálogos	CU-18	Completar Juego
CU-06	Comprar Objetos tienda	CU-19	Matar enemigos
CU-07	Vender Objetos inventario	CU-20	Conseguir experiencia y oro

CU-08	Obtener Objetos inventario	CU-21	Recibir daño
CU-09	Usar Objetos inventario	CU-22	Infligir daño
CU-10	Equipar Objetos inventario	CU-23	Curar
CU-11	Desequipar objetos inventario	CU-24	Permitir al jugador utilizar conjuros.
CU-12	Eliminar objetos	CU-25	Subir de nivel
CU-13	Completar misiones	CU-26	Luchar

Tabla 2.1: Casos de Uso

2.2.1.1 CU-01 Crear una partida nueva

CU-01	Crear una partida nueva	
Precondición	Estar en el menú de Inicio, dícese, no haber empezado a jugar.	
Descripción	El Usuario puede crear una partida nueva, eliminando la anterior si existiese y comenzar el juego desde el principio.	
Secuencia normal	1. El usuario elige "Nueva Partida" 2. El Sistema comprueba si existen archivos de guardado. 3. El usuario comienza una nueva partida con éxito.	
Postcondición	El usuario comienza el juego desde el principio, y el sistema habrá creado los nuevos archivos de guardado y borrado los antiguos en caso de existir.	
Excepciones	2	Si existen datos de guardado se borran.
Comentarios	-	

Tabla 2.2: CU-01 Crear una partida nueva

2.2.1.2 CU-02 Cargar una partida

CU-02	Cargar una partida	
Precondición	Existen archivos de guardado.	
Descripción	El usuario puede cargar su partida seleccionando "Continuar Partida".	
Secuencia normal	1. El sistema comprueba si existen datos de guardado. 2. El sistema activa el componente de la escena para poder cargar la partida 3. El usuario elige "Continuar Partida". 4. El usuario carga la partida con éxito.	
Postcondición	El sistema carga la partida y el jugador puede jugar desde donde lo dejó la ultima vez.	
Excepciones	1	1. No existen datos de guardado. 2. El sistema no activa el componente para cargar partida. 3. El usuario no puede cargar partida.
Comentarios	El componente de inicio de sesión estará desactivado por defecto, y será el sistema el encargado de activarlo para que el usuario pueda seleccionarlo.	

Tabla 2.3: CU-02 Cargar una partida

2.2.1.3 CU-03 Guardar partida

CU-03	Guardar Partida
Precondición	Se ha iniciado una partida y hemos abierto el menú y seleccionado el apartado "misc". O Cambio de escena.
Descripción	Guardamos los datos de la partida y salimos del juego. El sistema también guarda la partida de forma automática cada vez que cambiamos de escena.

Secuencia normal	1. El usuario selecciona “Guardar”. 2. El sistema guarda todos los datos de forma exitosa. O 1. El usuario entra en algún lugar y provoca un cambio de escena 2. El sistema guarda automáticamente antes de cambiar de escena de forma exitosa.
Postcondición	Los archivos de guardado han sido actualizados y puedes seguir jugando o salir de la partida.
Excepciones	-
Comentarios	-

Tabla 2.4: CU-03 Guardar partida

2.2.1.4 CU-04 Mover al jugador

CU-04	Mover al jugador
Precondición	El usuario está jugando y no tiene bloqueado el movimiento, dícese, no tiene ningún menú abierto, no está en combate, no está comprando o vendiendo y no está conversando con ningún NPC. También debe haber pasado un tiempo x desde el último movimiento, es necesario limitarlo.
Descripción	El jugador se mueve en la dirección elegida.
Secuencia normal	1. El sistema comprueba el tiempo desde el último movimiento 2. El sistema comprueba si el usuario está utilizando mando o no. 3. El usuario se mueve. 4. El sistema lee los movimientos del teclado y del mando si se encuentra conectado 5. El sistema detecta el movimiento y su dirección. 6. El sistema comprueba el movimiento 7. El sistema mueve al jugador a su nueva posición con éxito

Postcondición	El jugador se habrá movido en la dirección deseada	
Excepciones	1	1. No ha pasado el tiempo necesario 2. Se cancela el movimiento del jugador.
	6	1. Se comprueba el movimiento 2. Choca con un collider 3. Se cancela el movimiento del jugador.
Comentarios	-	

Tabla 2.5: CU-04 Mover al jugador

2.2.1.5 CU-05 Entablar diálogos

CU-05	Entablar Diálogos	
Precondición	Teniendo en cuenta que el mapa se divide en casillas, estar a 1 casilla de distancia ya sea en horizontal o en vertical y estar mirando al NPC.	
Descripción	Inicia la conversación con un NPC.	
Secuencia normal	1. El jugador se mueve en frente de un NPC y pulsa el botón de acción 2. El NPC lo detecta y comprueba la posición del jugador 3. Se inicia la conversación. 4. Se muestra la siguiente frase cada vez que se pulse el botón de acción hasta que no queden más. 5. El jugador y el NPC habrán mantenido una conversación con éxito	
Postcondición	El jugador y el NPC habrán hablado.	
Excepciones	2	1. Se comprueba que la posición no es correcta

		2. No se inicia la conversación.
Comentarios	-	

Tabla 2.6: CU-05 Entablar diálogo

2.2.1.6 CU-06 Comprar Objetos tienda

CU-06	Comprar Objetos Tienda	
Precondición	Haber seleccionado la opción “comprar” en el menú de la tienda.	
Descripción	Comprar un objeto de una tienda y añadirlo a tu inventario	
Secuencia normal	1. El usuario pulsa el botón de acción 2. El NPC de la tienda lo detecta e inicia el proceso de compra 3. Añade a tu inventario el objeto y deduce su precio de tu dinero 4. El jugador ha comprado el objeto correctamente	
Postcondición	El jugador habrá comprado correctamente el objeto y podrá seguir comprando.	
Excepciones	2	1. Al iniciar el proceso de compra comprueba si tienes espacio en el inventario o si tienes el máximo número de objetos de ese tipo que puedes llevar, y si tienes dinero. 2. Se cancela la compra 3. El NPC avisa por el cuadro de diálogo al jugador de que no se pudo comprar
Comentarios	-	

Tabla 2.7: CU-06 Comprar Objeto Tienda

2.2.1.7 CU-07 Vender Objetos inventario

CU-07	Vender Objetos Inventario	
Precondición	Tener objetos en el inventario y seleccionar la opción de vender en el menú de la tienda	
Descripción	Vendes objetos de tu inventario al NPC de la tienda	
Secuencia normal	1. El usuario pulsa el botón de acción 2. El NPC lo detecta e inicia el proceso de venta 3. Comprueba el objeto a vender 4. Añade a tu inventario el dinero y elimina el objeto de este. 5. El jugador habrá vendido su objeto correctamente.	
Postcondición	El jugador habrá eliminado el objeto de su inventario y habrá obtenido dinero equivalente al 'x' por ciento del valor original del objeto.	
Excepciones	3	1. Al iniciar el proceso de venta se comprueba si el objeto está equipado o si es un objeto clave, en cuyo caso no se podrá vender. 2. Se cancela la venta 3. Se avisa al jugador de que tiene el objeto equipado o que se trata de un objeto clave.
Comentarios	-	

Tabla 2.8: CU-07 Vender Objetos Inventario

2.2.1.8 CU-08 Obtener Objetos inventario

CU-08	Obtener Objetos Inventario
Precondición	No hayas recibido el objeto anteriormente
Descripción	Recibir objeto ya sea de un NPC o de un cofre

Secuencia normal	1. El jugador pulsa el botón de acción 2. El NPC o cofre comprueba si ya ha otorgado el objeto 3. Entrega el objeto 4. El jugador recibe el objeto correctamente.	
Postcondición	El jugador habrá obtenido su objeto y el NPC o cofre habrá registrado el objeto como entregado	
Excepciones	2	1. Si el objeto ya fue entregado no se entrega 2. Se cancela la entrega.
Comentarios	-	

Tabla 2.9: CU-08 Obtener Objetos inventario

2.2.1.9 CU-09 Usar Objetos inventario

CU-09	Usar Objetos Inventario	
Precondición	Tener objetos en el inventario	
Descripción	Seleccionas uno de los objetos de tu inventario y seleccionas usar.	
Secuencia normal	1. El jugador selecciona un objeto 2. Pulsa usar 3. El sistema analiza el objeto 4. El jugador usa el objeto correctamente	
Postcondición	Habrás usado el objeto y tendrás una copia menos de este o desaparecerá en caso de gastarlos todos.	
Excepciones	3	1. Analiza el objeto y no es usable 2. Cancela el uso 3. Avisa al jugador por el diálogo del inventario.
Comentarios	-	

Tabla 2.10: CU-09 Usar Objetos inventario

2.2.1.10 CU-10 Equipar Objetos Inventario

CU-10	Equipar Objetos Inventario	
Precondición	Tener objetos en el inventario y seleccionar la opción de equipar en el menú de inventario	
Descripción	El jugador se equipa con objetos para mejorar sus estadísticas	
Secuencia normal	1. El jugador selecciona un objeto 2. Pulsa Equipar 3. El sistema analiza el objeto 4. El sistema equipa el objeto y avisa al jugador. 5. El jugador ha equipado un objeto correctamente	
Postcondición	El jugador habrá equipado correctamente un objeto y sus estadísticas habrán mejorado.	
Excepciones	3	1. Al iniciar el proceso de equipar se comprueba si el objeto es equipable o no. 2. Se cancela si no es equipable 3. Avisa al jugador por el diálogo del inventario
Comentarios	-	

Tabla 2.11: CU-10 Equipar Objetos Inventario

2.2.1.11 CU-11 Desequipar objetos inventario

CU-11	Desequipar Objetos Inventario	
Precondición	Tener objetos en el inventario, tener el objeto seleccionado equipado y seleccionar la opción de equipar en el menú de la tienda	
Descripción	Una variación de Equipar. Simplemente se comprueba si el objeto está equipado, en cuyo caso se desequipa.	
Secuencia normal	1. El jugador selecciona un objeto 2. Pulsa Equipar	

	3. El sistema analiza el objeto y ve que está equipado 4. El sistema desequipa el objeto y avisa al jugador. 5. El jugador ha desequipado un objeto correctamente
Postcondición	El jugador habrá desequipado correctamente un objeto y sus estadísticas habrán vuelto a la normalidad.
Excepciones	-
Comentarios	-

Tabla 2.12: CU-11 Desequipar objetos inventario

2.2.1.12 CU-12 Eliminar objetos

CU-12	Eliminar objetos	
Precondición	Tener objetos en el inventario	
Descripción	Escoges y eliminas un objeto del inventario.	
Secuencia normal	1. El jugador selecciona un objeto 2. Pulsa Desechar 3. El sistema comprueba el objeto 4. El sistema borra el objeto 5. El sistema avisa al jugador de que el objeto se ha borrado	
Postcondición	El objeto habrá desaparecido del inventario.	
Excepciones	3	1. Si el objeto está equipado, o es un objeto clave no se podrá eliminar. 2. Se cancela la eliminación 3. Se avisa al jugador.
Comentarios	-	

Tabla 2.13: CU-12 Eliminar objetos

2.2.1.13 CU-13 Completar misiones

CU-13	Completar misiones
Precondición	Cumplir con las condiciones de la misión en concreto
Descripción	El NPC o sistema (dependiendo del tipo de misión) modificará los datos de guardado indicando que se ha completado una misión
Secuencia normal	1. El jugador habla con el NPC o realiza una cierta acción 2. El sistema o NPC detecta que ha cumplido la misión 3. Se modifica en los datos de guardado esta información 4. Si se requiere para esa misión en específico, el NPC o escena harán alguna acción. 5. El jugador habrá completado con éxito una misión.
Postcondición	El jugador habrá completado una misión desbloqueando algo en la historia o consiguiendo acceso a algún lugar
Excepciones	-
Comentarios	-

Tabla 2.14: CU-13 Completar misiones

2.2.1.14 CU-14 Atacar

CU-14	Atacar
Precondición	Estar en combate y no estar dormido
Descripción	El jugador realiza un ataque física contra el enemigo
Secuencia normal	1. El jugador pulsa el botón Atacar 2. El sistema lo detecta y ejecuta el turno

	3. El jugador habrá completado su acción de ataque en ese turno de forma exitosa.	
Postcondición	El jugador habrá realizado un ataque	
Excepciones	2	1. Si el jugador muere antes de atacar se cancelará la acción de ataque. 2. Se avisará al jugador de su muerte 3. Se finalizará el combate.
Comentarios	-	

Tabla 2.15: CU-14 Atacar

2.2.1.15 CU-15 Usar objetos batalla

CU-15	Usar objetos batalla	
Precondición	Estar en batalla, no estar dormido y tener objetos consumibles	
Descripción	Utilizas objetos consumibles desde el menú de batalla.	
Secuencia normal	1. El jugador pulsa el botón Objetos 2. El jugador elige un objeto 3. El sistema lo detecta y ejecuta el turno 4. El jugador habrá completado su acción de usar objeto en ese turno de forma exitosa.	
Postcondición	El jugador habrá usado el objeto elegido.	
Excepciones	-	
Comentarios	En este caso no habría posibilidad de morir antes de utilizar el objeto, ya que al usar objetos o hechizos de curación siempre va primero el jugador.	

Tabla 2.16: CU-15 Usar objetos batalla

2.2.1.16 CU-16 Huir

CU-16	Huir	
Precondición	Estar en batalla y no estar dormido	
Descripción	El jugador intenta huir de la batalla	
Secuencia normal	1. El jugador selecciona la opción de Huir en el menú de batalla 2. El sistema lo detecta y ejecuta la opción de Huir 3. El jugador huye de forma satisfactoria.	
Postcondición	El jugador escapa del combate.	
Excepciones	2	1. Al ejecutar la opción de huir se tira un random y dependiendo del resultado huye o no, si sale que no. 2. Se cancela la huida 3. Se avisa al jugador
Comentarios	-	

*Tabla 2.17: CU-16 Huir***2.2.1.17 CU-17 Menú pausa**

CU-17	Menú pausa	
Precondición	No tener el movimiento bloqueado (no estar realizando ninguna acción del tipo dialogar, comprar, luchar, etc.)	
Descripción	Accedes al menú de pausa donde puedes elegir entre ir al inventario, ver las estadísticas, ver las magias y ver miscelánea.	
Secuencia normal	1. El jugador pulsa el botón esc. o start si usa mando. 2. El sistema lo detecta e inicia el menú bloqueando al jugador	

	3. El jugador habrá accedido al menú correctamente
Postcondición	El jugador habrá accedido al menú y podrá elegir entre las opciones disponibles
Excepciones	-
Comentarios	-

Tabla 2.18: CU-17 Menú pausa

2.2.1.18 CU-18 Completar Juego

CU-18	Completar Juego	
Precondición	Derrotar al jefe final e ir a la zona inicial	
Descripción	El jugador habiendo derrotado al jefe final vuelve al inicio y se ejecuta la secuencia de completar juego.	
Secuencia normal	1. El jugador vuelve al inicio 2. El sistema comprueba si ha derrotado al jefe final 3. Se ejecuta la secuencia de fin del juego 4. Vuelves al menú y se borra el progreso	
Postcondición	El jugador habrá completado el juego y volverá a la pantalla inicial.	
Excepciones	2	1. Si no ha derrotado al jefe final no ocurrirá nada.
Comentarios	-	

Tabla 2.19: CU-18 Completar Juego

2.2.1.19 CU-19 Matar enemigos

CU-19	Matar enemigos	
Precondición	Estar en batalla	
Descripción	El jugador derrota a un enemigo	
Secuencia normal	1. El jugador ataca al enemigo 2. El enemigo recibe daño 3. El sistema comprueba si el enemigo ha sido derrotado. 4. Asigna la experiencia y oro 5. Avisa al jugador 6. Termina la batalla correctamente.	
Postcondición	El jugador habrá derrotado al enemigo y habrá recibido la experiencia y el oro indicado.	
Excepciones	3	1. Si el enemigo no ha sido derrotado, no se termina 2. Se da señal para continuar la batalla.
Comentarios	-	

Tabla 2.20: CU-19 Matar enemigos

2.2.1.20 CU-20 Conseguir experiencia y oro

CU-20	Conseguir experiencia y oro	
Precondición	Matar un enemigo	
Descripción	Se asigna el oro y experiencia correspondientes por el enemigo derrotado	
Secuencia normal	1. El enemigo es derrotado 2. El sistema asigna al jugador la experiencia y oro 3. Avisa al jugador por orden de todo lo que ha conseguido y espera su respuesta con el botón de acción	

	4. Termina el combate.	
Postcondición	El jugador habrá obtenido experiencia y oro	
Excepciones	2	1. Si el jugador ha alcanzado el nivel máximo no se asignará experiencia 2. Se asigna oro 3. Se avisa al jugador 4. Se termina el combate.
Comentarios	-	

Tabla 2.21: CU-20 Conseguir experiencia y oro

2.2.1.21 CU-21 Recibir daño

CU-21	Recibir daño	
Precondición	Estar en combate	
Descripción	Eres golpeado por un enemigo	
Secuencia normal		1. El enemigo golpea al jugador 2. El sistema se encarga de calcular el daño según el ataque del enemigo, dependiendo de si es ataque mágico o físico 3. Se encarga de infligir al jugador el daño correspondiente calculado a partir de la defensa del jugador y el daño calculado anteriormente 4. Se avisa al jugador del daño recibido
Postcondición	Recibirás daño del enemigo	
Excepciones	-	
Comentarios	-	

Tabla 2.22: CU-21 Recibir daño

2.2.1.22 CU-22 Infligir daño

CU-22	Infligir daño
Precondición	Estar en combate
Descripción	Se inflige daño al enemigo
Secuencia normal	1. El jugador golpea al enemigo 2. El sistema se encarga de calcular el daño según el ataque del jugador, dependiendo de si es ataque mágico o físico 3. Se encarga de infligir al enemigo el daño correspondiente calculado a partir de la defensa del enemigo y el daño calculado anteriormente 4. Se avisa al jugador del daño infligido
Postcondición	El jugador habrá golpeado al enemigo
Excepciones	-
Comentarios	-

Tabla 2.23: CU-22 Infligir daño

2.2.1.23 CU-23 Curar

CU-23	Curar
Precondición	Estar en combate
Descripción	Ya sea por medio de un hechizo o por medio de un objeto, restauras hp
Secuencia normal	1. El jugador utiliza un hechizo de curación o un objeto de curación 2. El sistema lo detecta 3. El sistema incrementa su vida y comprueba que no supere el límite de vida máxima

	4. El jugador ha sido curado correctamente.
Postcondición	El jugador habrá recibido una curación
Excepciones	-
Comentarios	-

Tabla 2.24: CU-23 Curar

2.2.1.24 CU-24 Permitir al jugador utilizar conjuros.

CU-24	Permitir al jugador utilizar conjuros.	
Precondición	Estar en combate o tener el menú de conjuros del inventario abierto	
Descripción	Se utiliza un conjuro y produce el efecto de ese conjuro	
Secuencia normal	1. El jugador selecciona un conjuro 2. El sistema lo detecta y ejecuta los efectos propios de ese conjuro 3. El jugador habrá usado un conjuro correctamente.	
Postcondición		
Excepciones	2	1. Si el jugador no tiene mana no podrá lanzar un conjuro, al igual que si utiliza un conjuro de ataque fuera de combate. 2. El sistema avisará al jugador de que no es posible realizar esa acción
Comentarios	-	

Tabla 2.25: CU-24 Permitir al jugador utilizar conjuros.

2.2.1.25 CU-25 Subir de nivel

CU-25	Subir de nivel	
Precondición	Tener experiencia suficiente para subir al próximo nivel	
Descripción	Cuando en combate el jugador derrota a un enemigo, si tiene la experiencia suficiente sube de nivel y con ello aumentan sus estadísticas	
Secuencia normal	1. El jugador derrota un enemigo 2. Se asignan los puntos de exp y oro 3. El sistema comprueba si hay puntos de exp suficientes para pasar al siguiente nivel 4. El sistema aumenta el nivel y las estadísticas del jugador 5. El jugador sube de nivel con éxito;	
Postcondición	El jugador habrá subido de nivel y sus estadísticas habrán mejorado.	
Excepciones	3	1. En caso de no tener los puntos suficientes de exp se cancela la subida de nivel. 2. El sistema cancela la subida de nivel.
Comentarios	-	

*Tabla 2.26: CU-25 Subir de nivel***2.2.1.26 CU-26 Luchar**

CU-25	Luchar
Precondición	Estar en una zona que permita el combate
Descripción	Cada vez que se da un paso hay una probabilidad de que se inicie un combate

Secuencia normal	1. El jugador da un paso 2. El sistema comprueba la zona y empieza un combate 3. El sistema bloquea al jugador hasta que termine el combate 4. El jugador comienza un combate correctamente	
Postcondición	El habrá entrado en combate	
Excepciones	2	1. En caso de no permitirse el combate o de haber derrotado al jefe final no se podrá luchar 2. El sistema cancelará el inicio del combate.
Comentarios	-	

Tabla 2.27: CU-26 Luchar

2.2.2 Diagramas de clases

Tal como dijimos, veremos un diagrama de clases general de la composición de una escena.

2.2.2.1 Diagrama de clases batalla

En este caso veremos las clases que intervienen en una batalla:

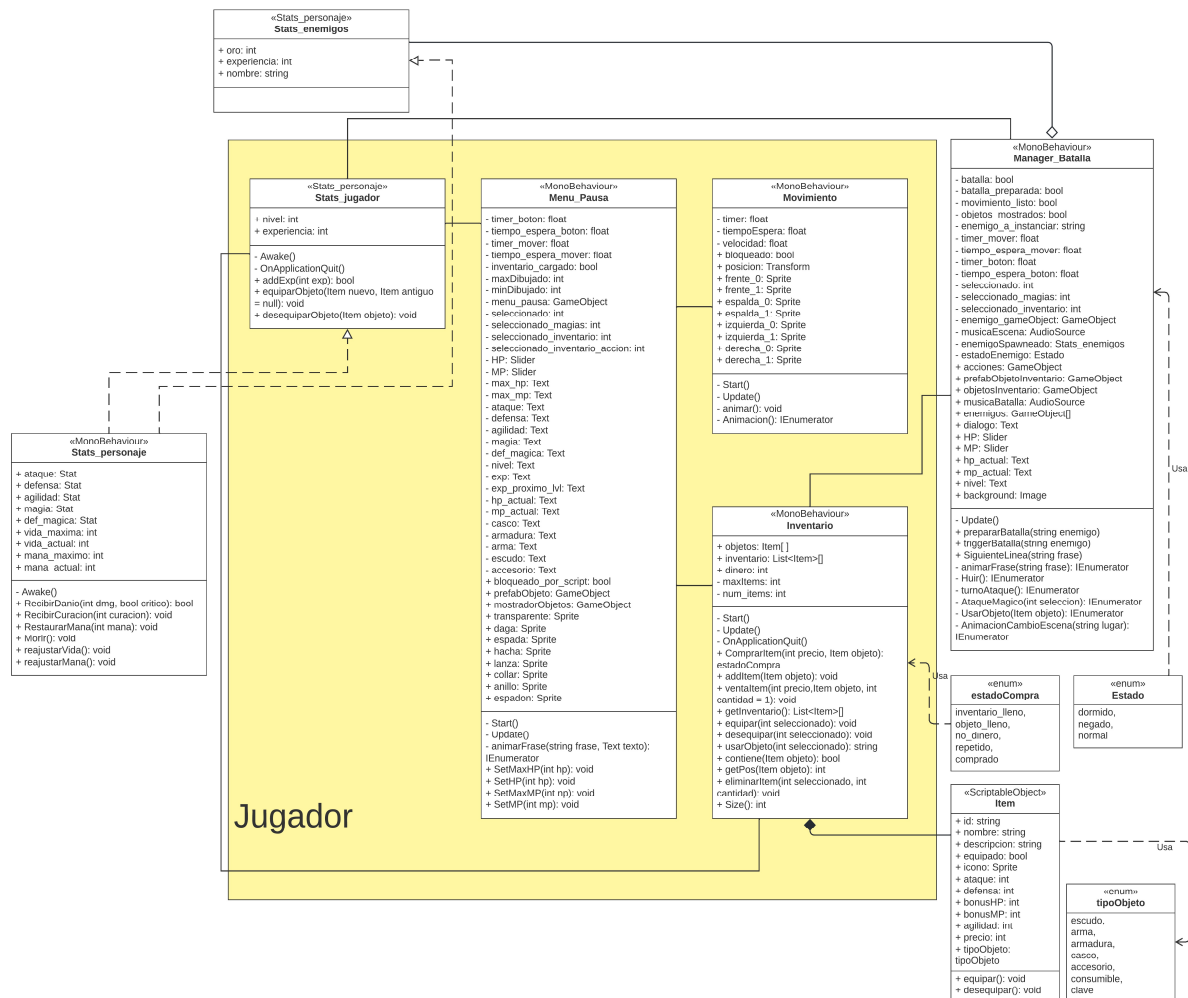


Ilustración 2.1: Diagrama de clases - Batalla

La clase Menu_Pausa no interviene como tal en la batalla. Está incluida para que veamos las clases que porta a modo de Scripts nuestro personaje.

2.2.2.2 Diagrama de clases diálogo

En la siguiente imagen vemos la relación entre las clases del jugador, un NPC y el Manager_Dialogos.

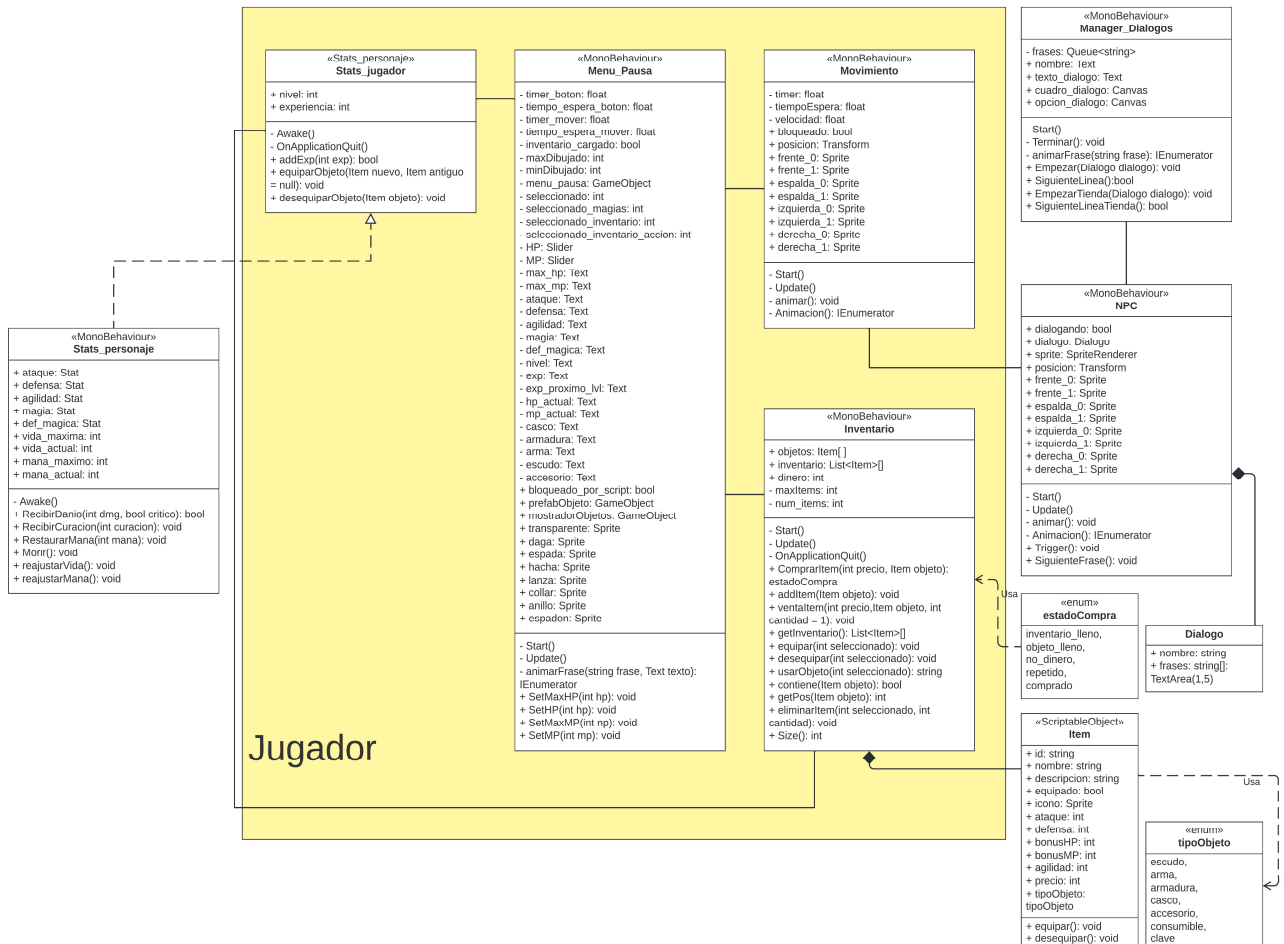


Ilustración 2.2: Diagrama de clases - Diálogo

2.2.2.3 Diagrama de clases tienda

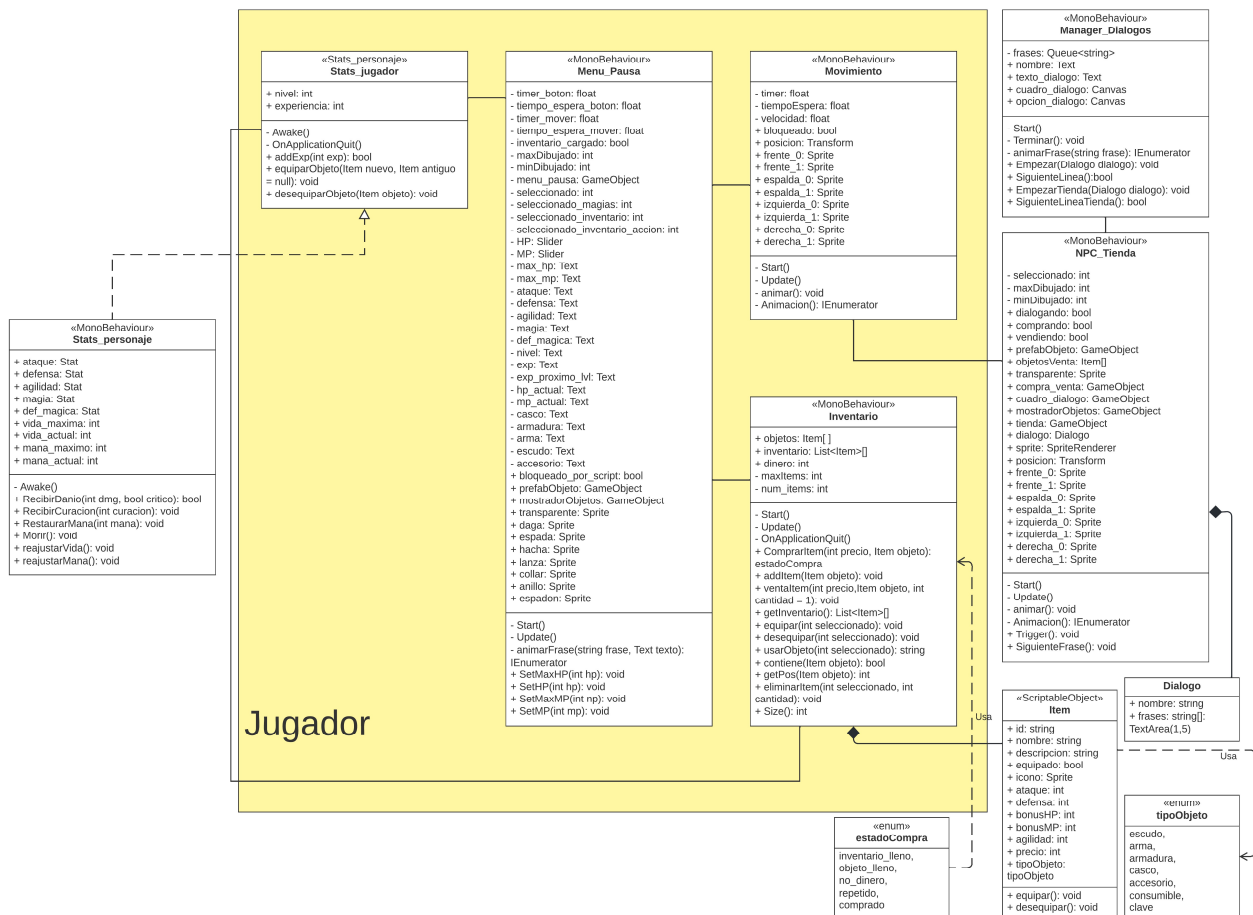


Ilustración 2.3: Diagrama de clases - Tienda

2.2.3 Diseño interfaz

Para diseñar la interfaz nos hemos basado en la interfaz del juego más reciente de la saga DQ, DQ XI. Aquí mostramos algunos ejemplos de la interfaz de dicho juego sacados de la web (Database, s.f.):



Ilustración 2.4: Menú Pausa Dragon Quest XI



Ilustración 2.5: Atributos Dragon Quest XI



Ilustración 2.6: Inventario Dragon Quest XI

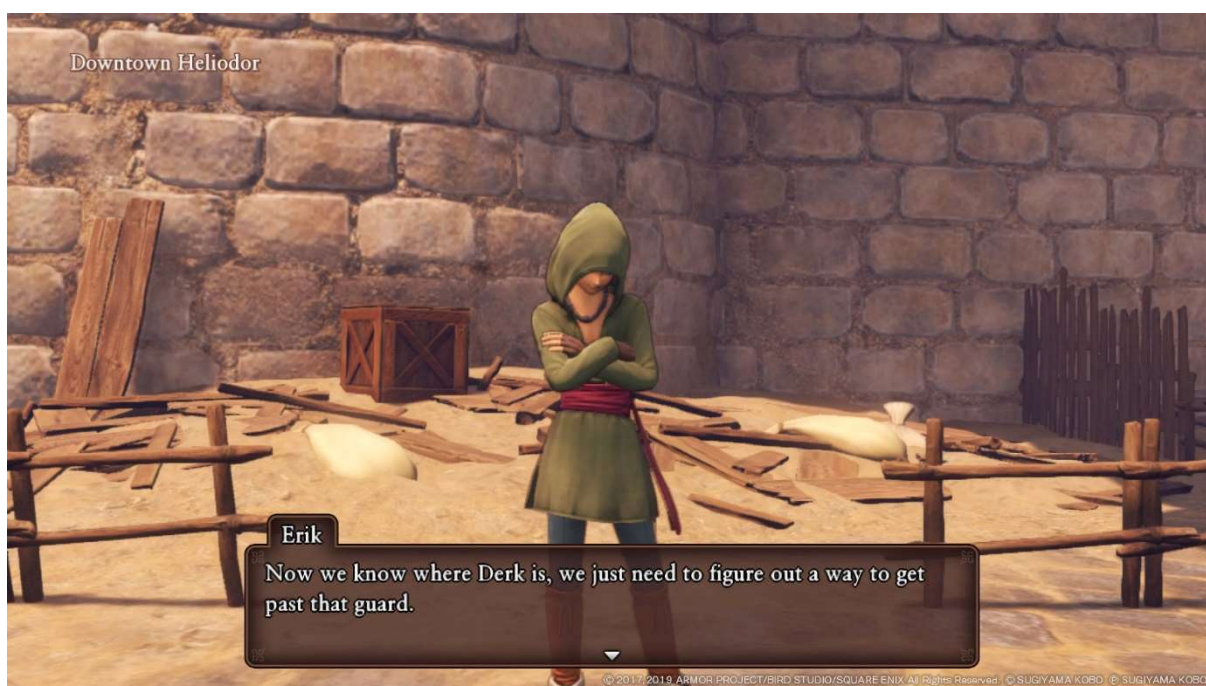


Ilustración 2.7: Diálogo Dragon Quest XI

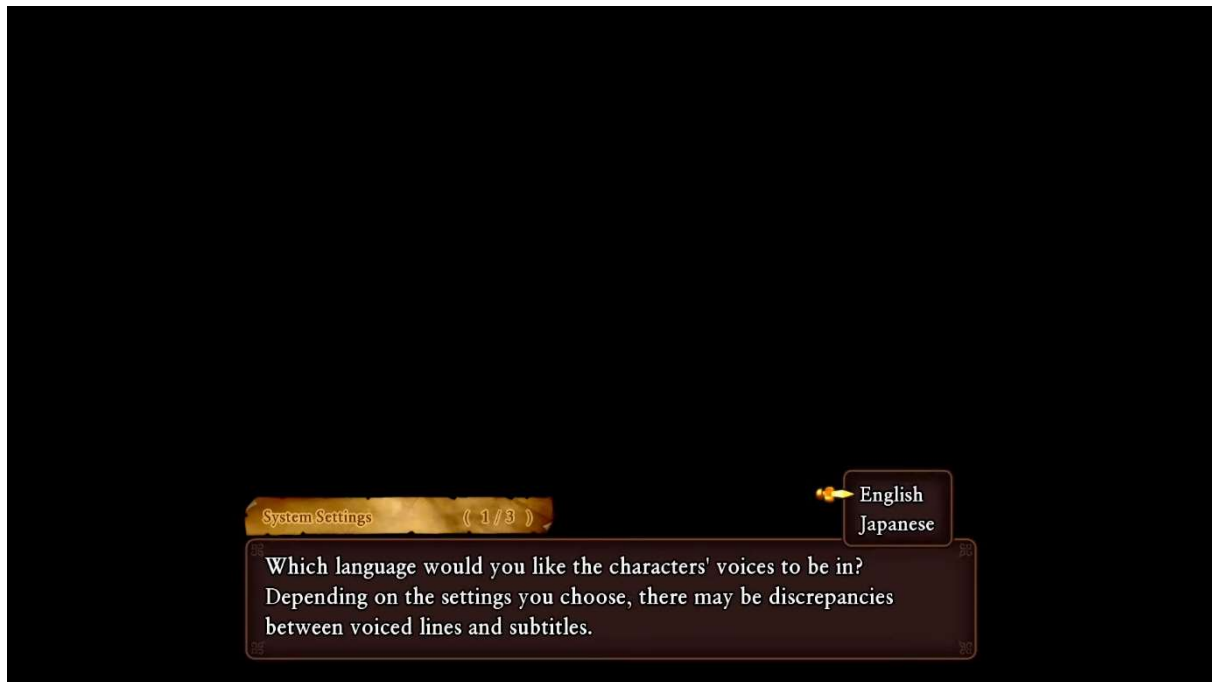


Ilustración 2.8: Selección Dragon Quest XI



Ilustración 2.9: Inicio Dragon Quest XI

En el próximo punto se explicará con más detalle el proceso de creación de la UI.

3 DESARROLLO

En esta sección de la documentación haremos un resumen del trabajo que diseñamos en cada una de los diferentes sprints de nuestro proyecto, explicando en detalle los diferentes elementos y funcionalidades implementadas.

3.1 Primera Iteración – Creación de escenas y mapas

La primera iteración en nuestro proyecto consistió en la creación de las distintas escenas y mapas, así como managers básicos que nos permitiesen la transición entre escenas.

Este sprint, a pesar de ser el más sencillo en cuanto a lo programación se refiere nos llevó una gran cantidad de horas puesto que al tratarse de un juego 2D hay que pintar los escenarios con cuidado en las casillas correspondientes de los TileMaps correspondientes, por ello le fueron asignadas 2 semanas de sprint.

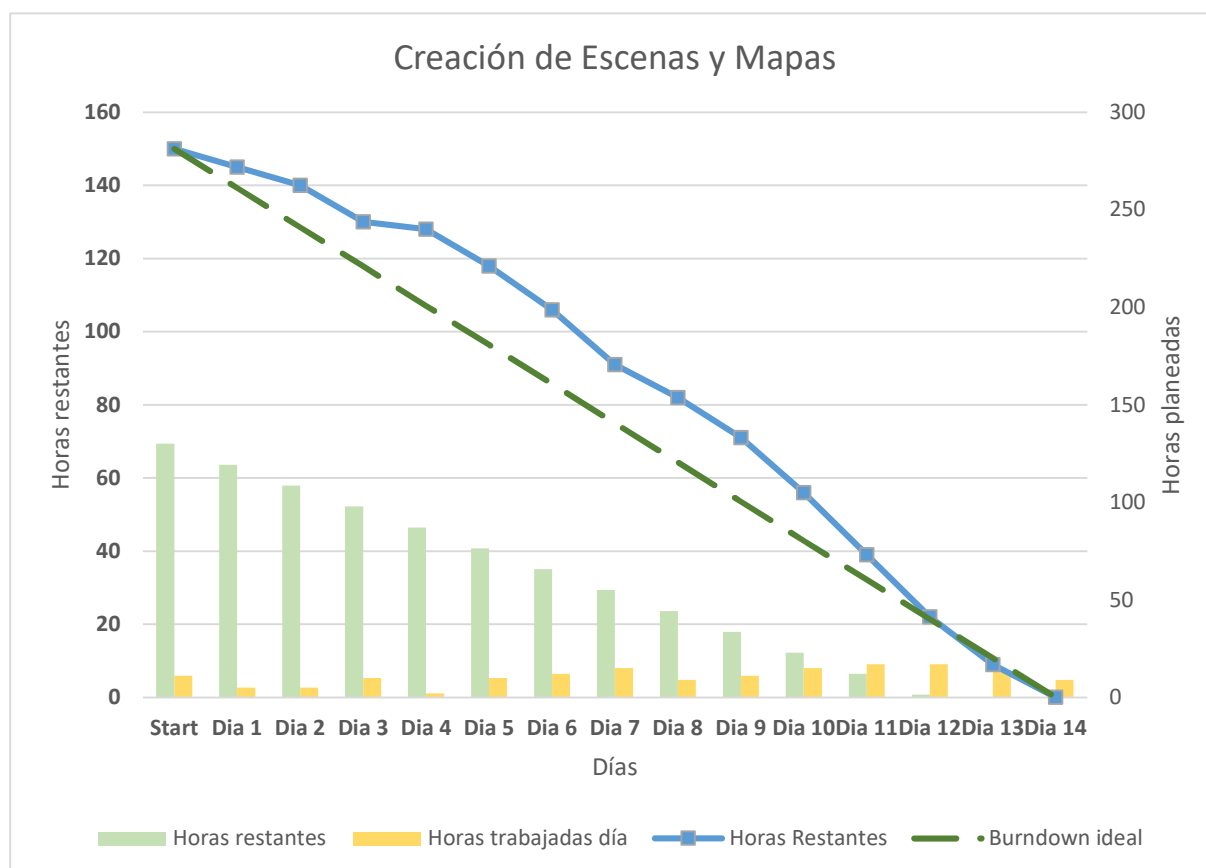


Ilustración 3.1: Diagrama Burndown primera iteración

Como vemos, tardamos unas 150 horas aproximadamente.

Nuestro proyecto tiene 33 escenas diferentes, de las cuales una es el menú de inicio, y el resto son los mapas del juego en los que el jugador se moverá. En este sprint nos dedicamos a hacer los 32 mapas navegables por el jugador. El menú de inicio fue hecho en otro sprint que veremos más adelante.

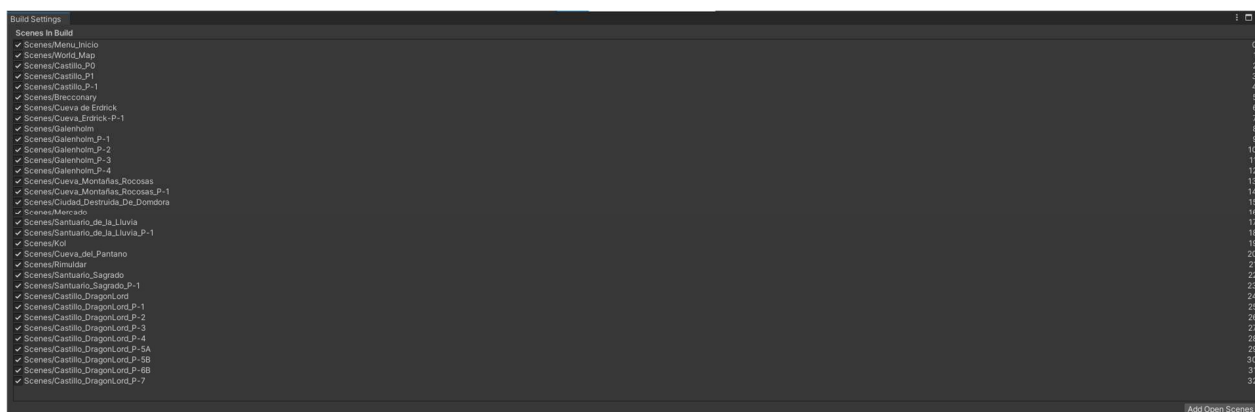


Ilustración 3.2: Build Settings del proyecto en Unity

En las subsecciones que siguen veremos los diferentes mapas que hicimos, comparándolos con los originales del juego para MSX2, ya que no se encuentran imágenes de la versión de NES o MSX, pero son prácticamente idénticos.

3.1.1 Castillo_P1

Este es el primer mapa en el que apareceremos cuando comencemos una nueva partida. Procuraremos tanto en este, como en los siguientes mapas, mostrarlos según el orden lógico en el que iremos visitándolos en el juego.

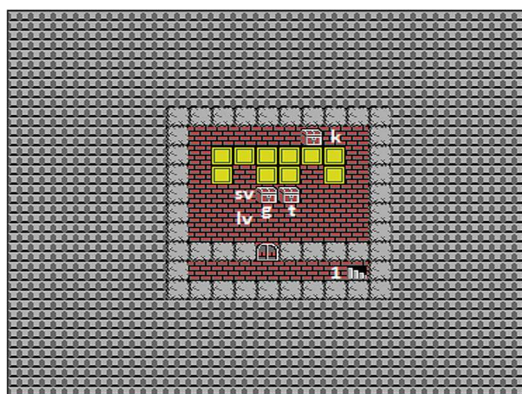


Ilustración 3.3: Castillo P1 MSX2

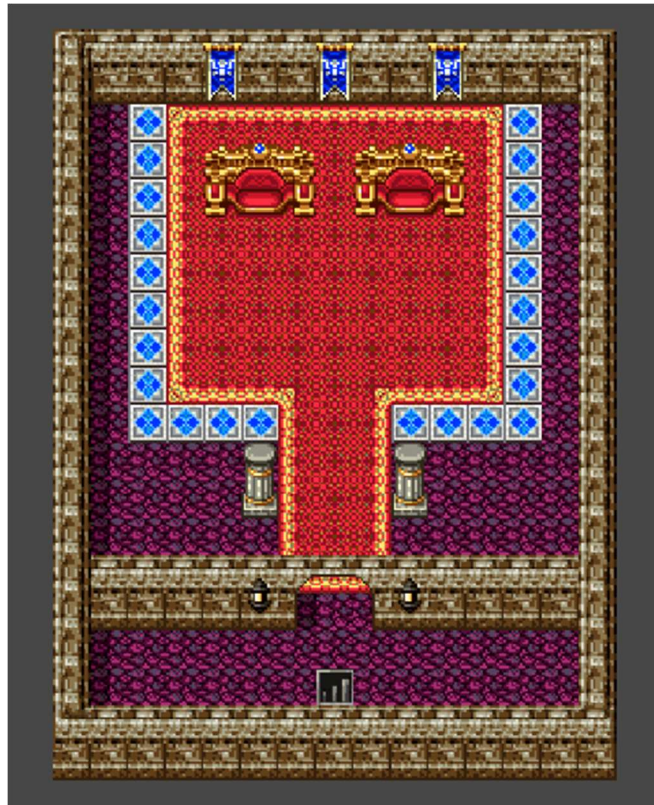


Ilustración 3.4: Castillo P1 de nuestro remake

Para crear nuestros mapas nos hemos basado en los originales y en el remake de Switch, ya que los assets como dijimos son extraídos de dicha versión.

La paleta que hemos usado para la creación de los mapas de este castillo se muestra a continuación, la llamaremos ‘Pueblo1’:

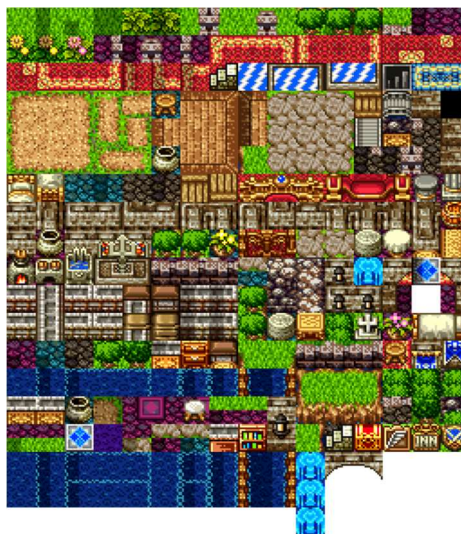


Ilustración 3.5: Paleta Pueblo1

Además de crear el mapa, también hemos incluido la canción “Château Tantagel” del ya mencionado Koichi Sugiyama, e interpretada por la Orquesta Sinfónica Metropolitana de Tokio, que podemos encontrar aquí: (Sugiyama, s.f.) o bien escucharla en Youtube, aquí: (LordRoto, s.f.). Esta la utilizan las 3 escenas dedicadas a este castillo, por lo tanto, solo se mencionará aquí.

En este punto desarrollamos el mánager básico de esta escena. Solo implementaba funcionalidad para permitir al jugador (en un futuro sprint) comprobar si se ha entrado en una escalera para cambiar de escena y bloquearlo para que no siga moviéndose:

```
public static bool entrar_lugar(Vector3 vector)
{
    if (vector == new Vector3(1, -12, 0))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Ilustración 3.6: Castillo_P1_manager - entrar_lugar

Este método será accedido por el jugador y comprobará con su posición actual si se encuentra en una escalera.

```
void SalirDelMapa()
{
    if(Jugador.transform.position == new Vector3(1, -12, 0))
    {
        StartCoroutine(AnimacionCambioEscena("Castillo_P0"));
    }
}
```

Ilustración 3.7: Castillo_P1_manager - SalirDelMapa

Puesto que la escalera no es sino una casilla pintada, comprobaremos si el jugador coincide con la posición de dicha casilla y de ser así, cambiaremos de mapa.

```
1 referencia
IEnumerator AnimacionCambioEscena(string escena)
{
    cambioEscena.SetTrigger("Start");

    yield return new WaitForSeconds(1.1f);

    SistemaGuardado.SetUltimaEscena(SceneManager.GetActiveScene().name);

    SceneManager.LoadScene(escena);
}
```

Ilustración 3.8: Castillo_P1_manager – AnimacionCambioEscena

Aquí, primeramente, ejecutamos una animación que creamos para el cambio de escenas (y que será utilizada en todas las escenas del juego para hacer la transición menos brusca), y después de esperar lo que dura la animación, cambiar la escena. El sistema de guardado es de un sprint posterior, así que, no explicaremos nada sobre ello de momento.

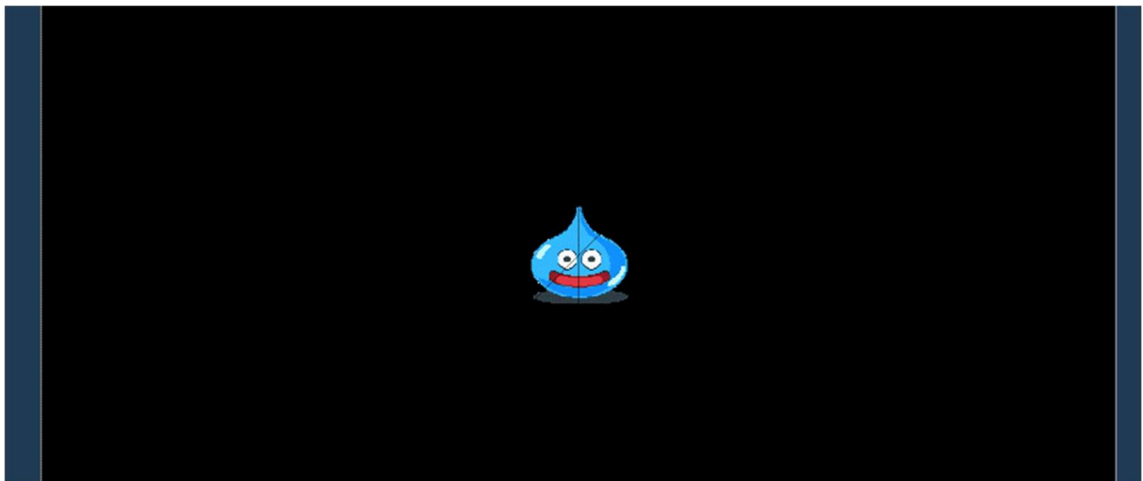


Ilustración 3.9: Animación Cambio de Escena

3.1.2 Castillo_P0

A este mapa podemos acceder desde distintos lugares y llegar a él también desde distintos lugares, ya sea desde el mundo, desde el piso superior o el inferior.



Ilustración 3.10: Castillo P0 MSX2

Este es el mapa original de MSX2, y el nuestro es el siguiente:

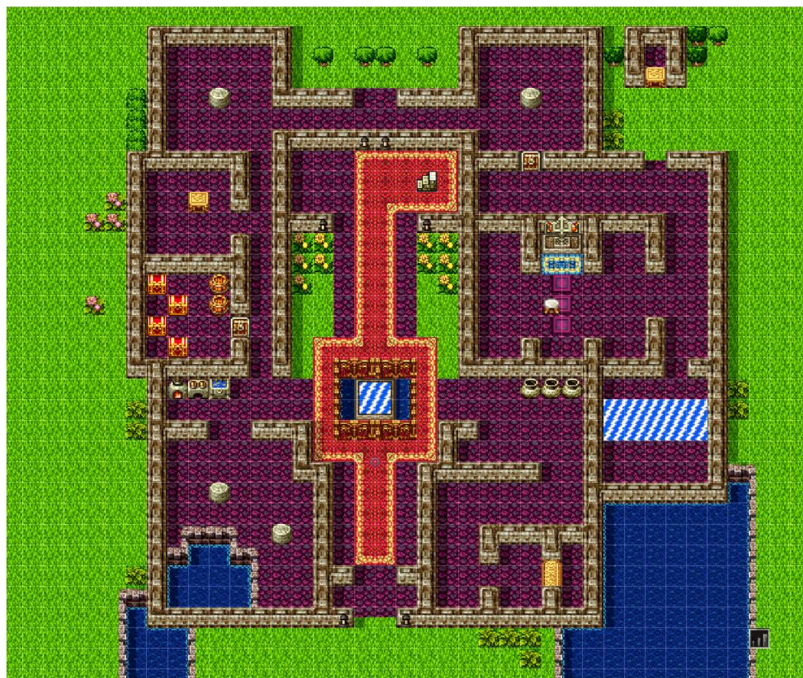


Ilustración 3.11: Castillo P0 de nuestro remake

Al igual que en el anterior, hemos utilizado la paleta 'Pueblo1' para crear el mapa. La única diferencia en cuanto a la forma de hacer este mapa es que para el

agua hemos utilizado un tipo de casilla especial llamada “casilla animada”. Esta nos permite animar las casillas simulando el movimiento del agua.

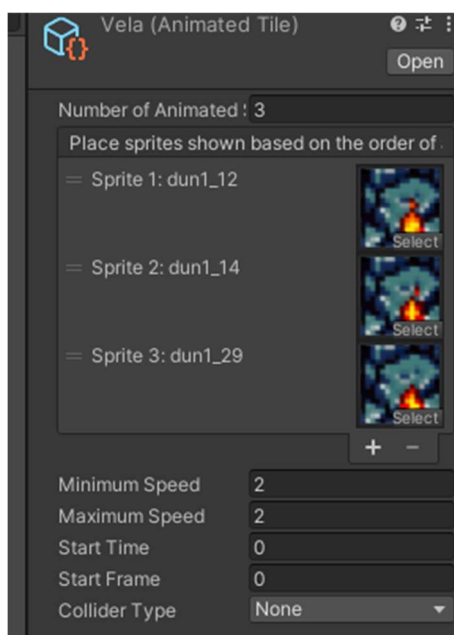


Ilustración 3.12: Ejemplo de casilla animada

En cuanto a código, al igual que en el anterior mapa, tenemos código para que el jugador sepa cuando ha entrado a un lugar u otro. No hace falta mostrar el código en este caso ya que es equivalente al anterior, pero teniendo en cuenta las posibles salidas del mapa como son las dos escaleras que hay y los huecos superiores e inferiores que hay en el mapa. También se tiene en cuenta cuan hacia la derecha te encuentras, de modo que si en vez de ir a la escalera intentas ir hacia la derecha saldrás del mapa hacia el mapa del mundo.

3.1.3 Castillo_P-1

A este mapa llegamos entrando desde la escalera exterior del anterior.

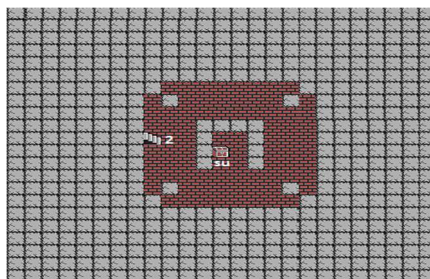


Ilustración 3.13: Castillo P-1 MSX2



Ilustración 3.14: Castillo P-1 de nuestro remake

Este mapa al igual que los anteriores, y a pesar de que se vea muy distinto, también utiliza la paleta ‘Pueblo1’.

En el manager se implementó lo mismo que en el resto de los mapas, por tanto, no es necesario explicarlo.

3.1.4 World_Map

Este mapa es sin duda el que más trabajo ha llevado (en torno a las 55 horas) ya que es el más grande, y el mapa que actúa como nexo para todos los demás mapas del juego. Para que se pueda ver correctamente hemos dividido la imagen en 4 partes, superior izquierda, superior derecha, inferior izquierda e inferior derecha. En dichas imágenes se ven los siguientes lugares:

- **Parte sup. Izquierda:** Castillo de Tantegel (**A**), Breconary (**B**), Cueva de Erdrick (**C**), Galenholm (**D**), Cueva de la montaña (**18**), Castillo DragonLord (**M**)
- **Parte sup. Derecha:** Kol (**E**), Santuario de la Lluvia (**F**), Cueva del Pantano (**G**)
- **Parte inf. Izquierda:** Ciudad Destruída de Domdora (**K**)
- **Parte inf. Derecha:** Mercado/Catlin (**L**), Santuario Sagrado (**I**), Rimuldar (**H**)



Ilustración 3.15: Parte superior izquierda - Mapa Mundo - MSX2



Ilustración 3.16: Parte superior derecha - Mapa Mundo - MSX2

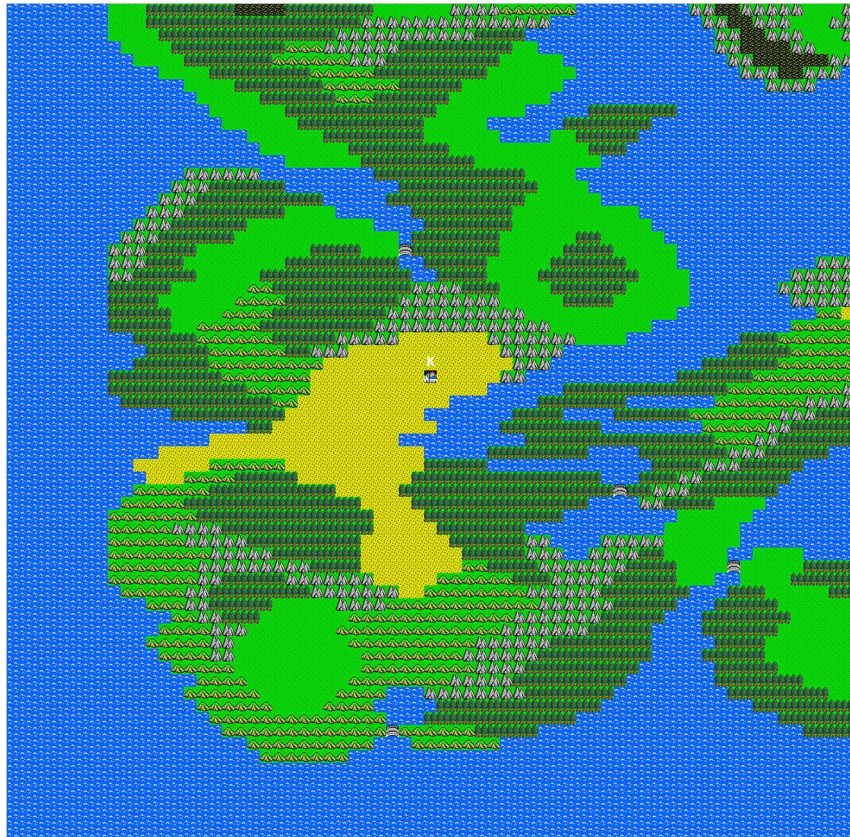


Ilustración 3.17: Parte inferior izquierda - Mapa Mundo - MSX2



Ilustración 3.18: Parte inferior derecha - Mapa Mundo - MSX2

Y en la siguiente imagen podemos ver el mapa del mundo que nosotros hemos creado. Dado que ya hemos visto donde se encuentra cada ciudad y cada cueva no es necesario dividir nuestro mapa en otras 4 partes, hemos seguido la distribución original en la medida de lo posible:

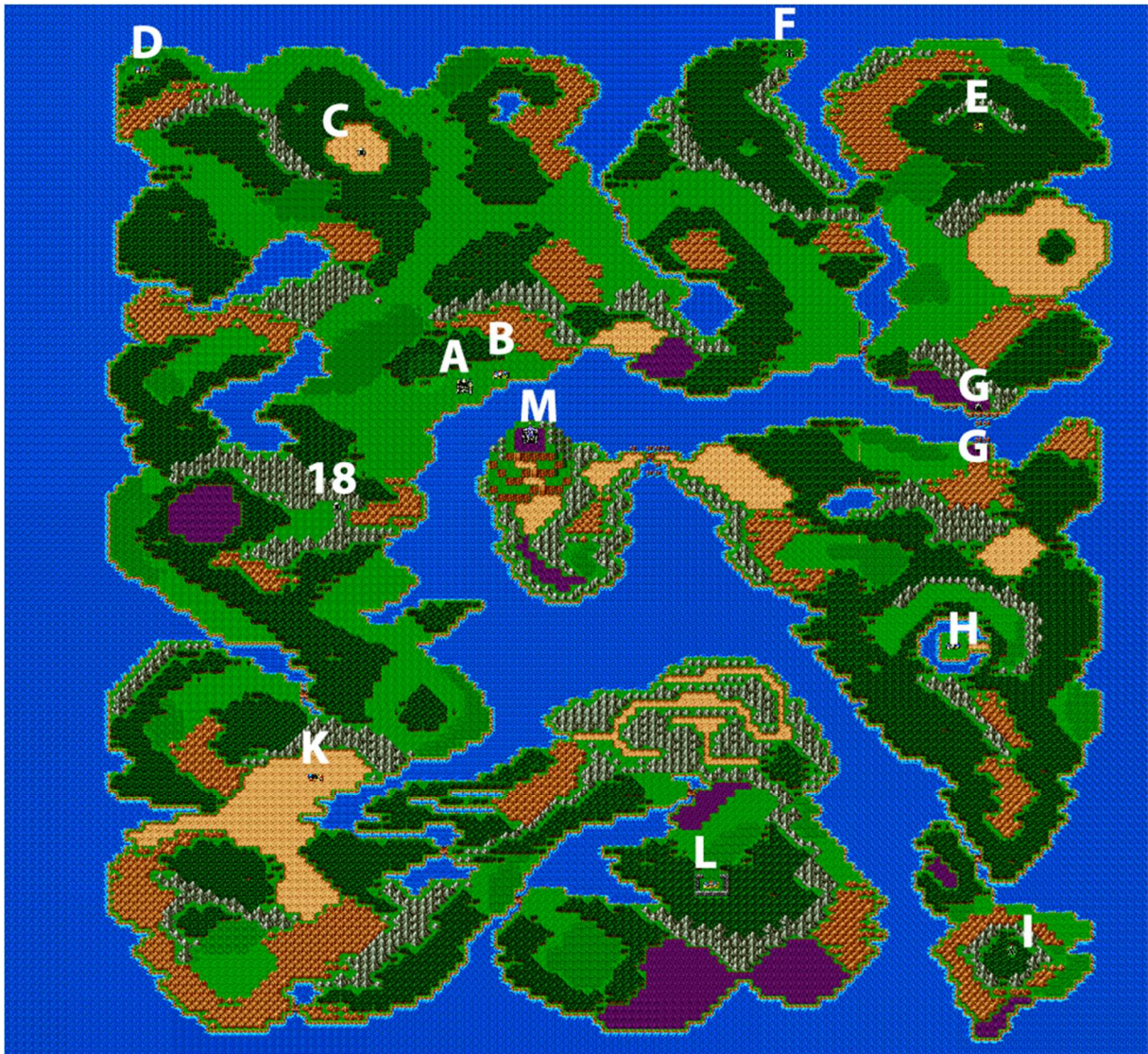


Ilustración 3.19: World_Map de nuestro remake

Para crear este mapa utilizamos una paleta específica que solo se ha utilizado en este mapa, la llamaremos 'WorldMap' y se muestra a continuación:



Ilustración 3.20: Paleta WorldMap

El mánager, al igual que todos los demás, solo se encarga de verificar en qué zona del mapa se encuentra el jugador, para, en caso de ser necesario, moverlo a una nueva escena. También, igual que con el agua de la paleta ‘Pueblo1’ aquí animamos el agua haciendo uso de las casillas animadas.

Algo que también cambia con respecto a los mapas anteriores es la música, ya que en este caso la canción es “Unkown World”. Puede escucharse aquí: (Roto, Youtube, s.f.)

3.1.5 Breconary

Breconary es el primer pueblo que nos encontraremos al salir del castillo inicial.



Ilustración 3.21: Breconary MSX2**Ilustración 3.22: Breconary de nuestro remake**

Para crear este mapa hemos utilizado la paleta 'Pueblo1', y para la música se ha utilizado la canción "People", disponible aquí: (Roto, Youtube, s.f.)

3.1.6 Cueva de Erdrick

La cueva de Erdrick es la tumba del susodicho, tiene dos niveles, aquí se nos presenta el primero.

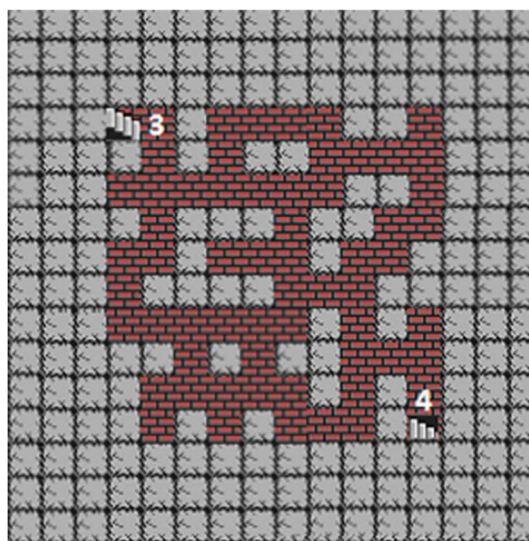
**Ilustración 3.23: Cueva de Erdrick MSX2**



Ilustración 3.24: Cueva de Erdrick de nuestro remake

Para hacer este mapa se ha utilizado una nueva paleta, la llamaremos “Mazmorra1”. Es la siguiente:

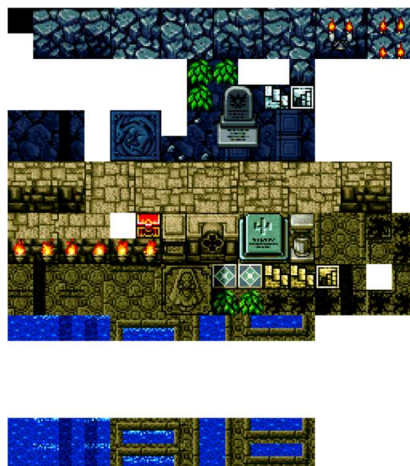


Ilustración 3.25: Paleta Mazmorra1

También, se han animado las velas que se ven en el mapa de igual modo que el agua en otras escenas.

Otra cosa más diferente es que al tratarse de una mazmorra se ha utilizado una canción llamada “Dungeon”: (Roto, Youtube, s.f.)

Además de todo esto, este mapa se trata de la primera cueva en la que entramos, y dado que es subterránea añadimos un Canvas con una imagen que nos hace ver el mapa de la siguiente forma y dar la sensación de estar en una cueva donde la visibilidad es limitada:

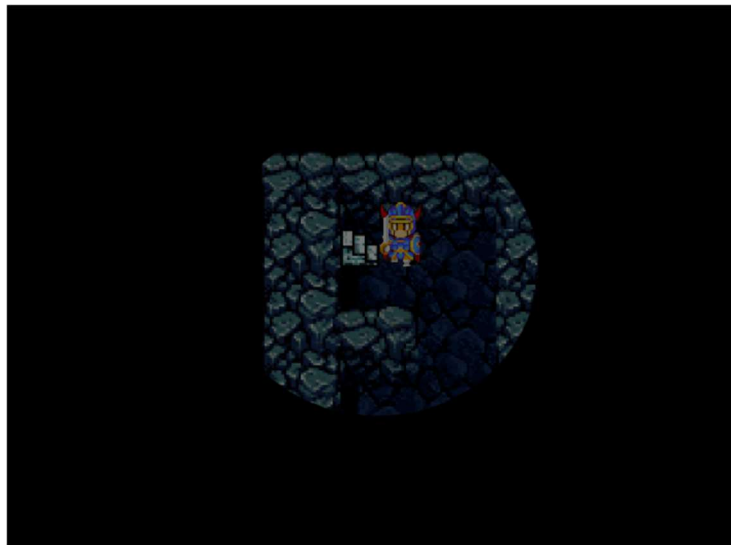


Ilustración 3.26: Modo Cueva

3.1.7 Cueva_Erdrick-P-1

Este es el piso inferior de la cueva de Erdrick. Tiene los mismos elementos que el anterior mapa, está hecho con la paleta “Mazmorra1” y como música utilizamos la canción “Dungeon”.

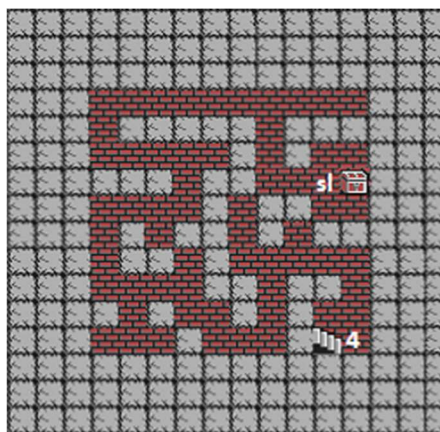


Ilustración 3.27: Cueva de Erdrick P-1 MSX2

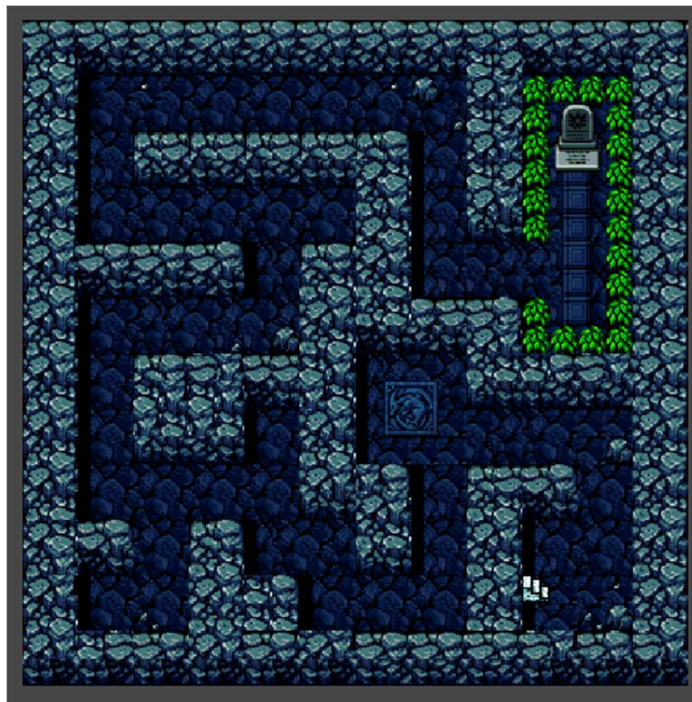


Ilustración 3.28: Cueva de Erdrick P-1 de nuestro remake

3.1.8 Galenholm

Galenholm está en la parte noroeste del mundo.

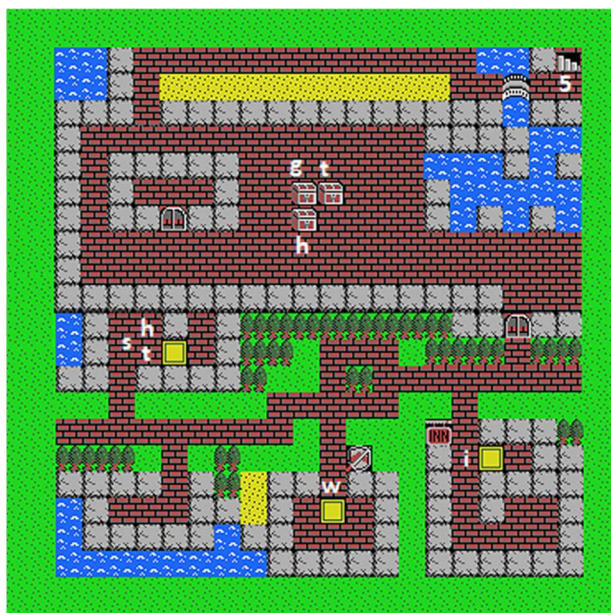


Ilustración 3.29: Galenholm MSX2

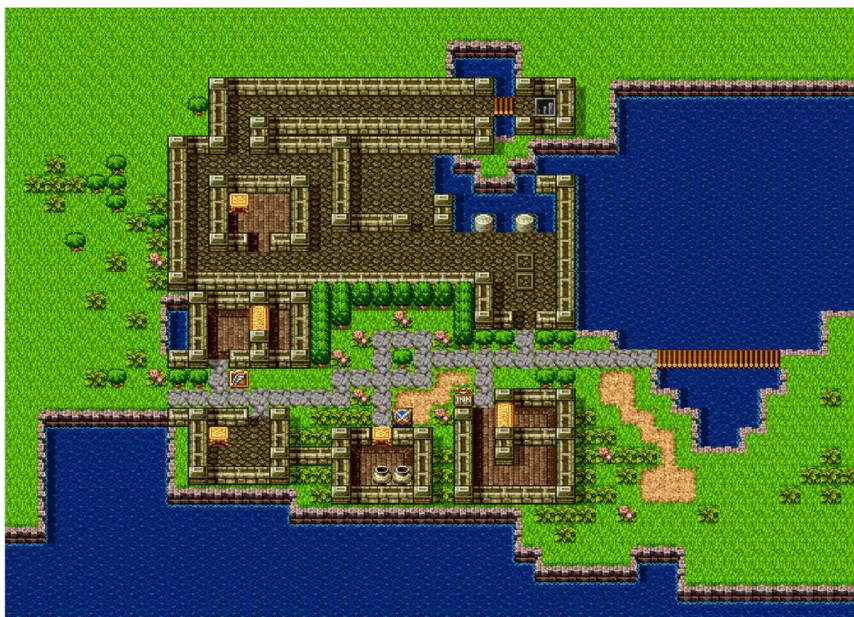


Ilustración 3.30: Galenholm de nuestro remake

Para hacer este mapa hemos utilizado una paleta nueva a la que llamaremos “Pueblo2”, es la siguiente:



Ilustración 3.31: Paleta Pueblo2

En cuanto a la música se ha utilizado “People”.

En las siguientes subsecciones seguiremos viendo el resto de niveles de Galenholm a pesar de ser una zona de más alto nivel y de que dijimos que seguiríamos un orden lógico, ya que pertenecen al mismo pueblo.

3.1.9 Galenholm_P-1

La parte subterránea de Galenholm. Al igual que todas las cuevas o mazmorras subterráneas utilizan el Canvas “Cueva” ya mencionado anteriormente.

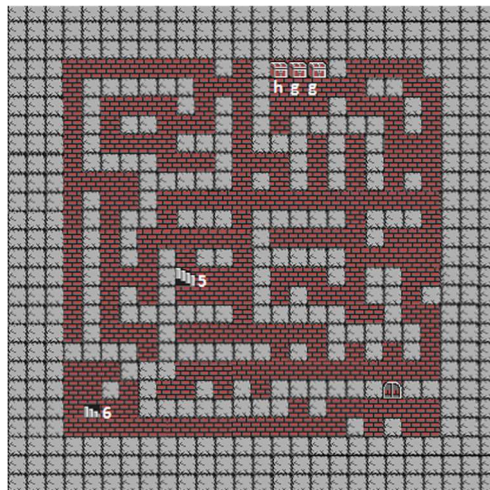


Ilustración 3.32: Galenholm P-1 MSX2



Ilustración 3.33: Galenholm P-1 de nuestro remake

En este mapa hemos utilizado la paleta “Mazmorra1” al igual que en la Cueva de Erdrick. Y la música, al tratarse de una mazmorra es “Dungeon”.

Aquí, y en todos los mapas que contienen múltiples escaleras que van hacia arriba o hacia abajo (a diferencia de por ejemplo Castillo_P0 donde tenemos escaleras que van hacia arriba y hacia abajo, pero solamente 1), tenemos unas variables definidas que utilizaremos una vez tengamos nuestro sistema de guardado, son arrays de float y tienen la siguiente forma:

```
private float[] escalera1 = { 2, 12, 0 };  
private float[] escalera2 = { 10, -10, 0 };  
private float[] escalera3 = { -22, -18, 0 };  
private float[] escalera4 = { 11, 12, 0 };  
private float[] escalera5 = { -14, -6, 0 };
```

Ilustración 3.34: Ejemplo de "variables escalera"

Estas variables nos sirven para conocer el punto de aparición en el piso al que nos dirigimos. Esto lo veremos en profundidad más adelante, en sprints posteriores.

3.1.10 Galenholm_P-2

Segundo piso de la parte subterránea de Galenholm.

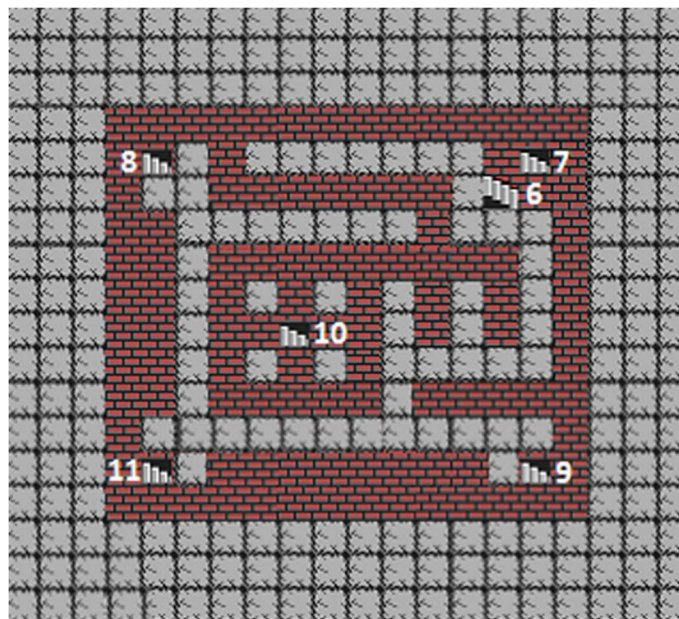


Ilustración 3.35: Galenholm P-2 MSX2

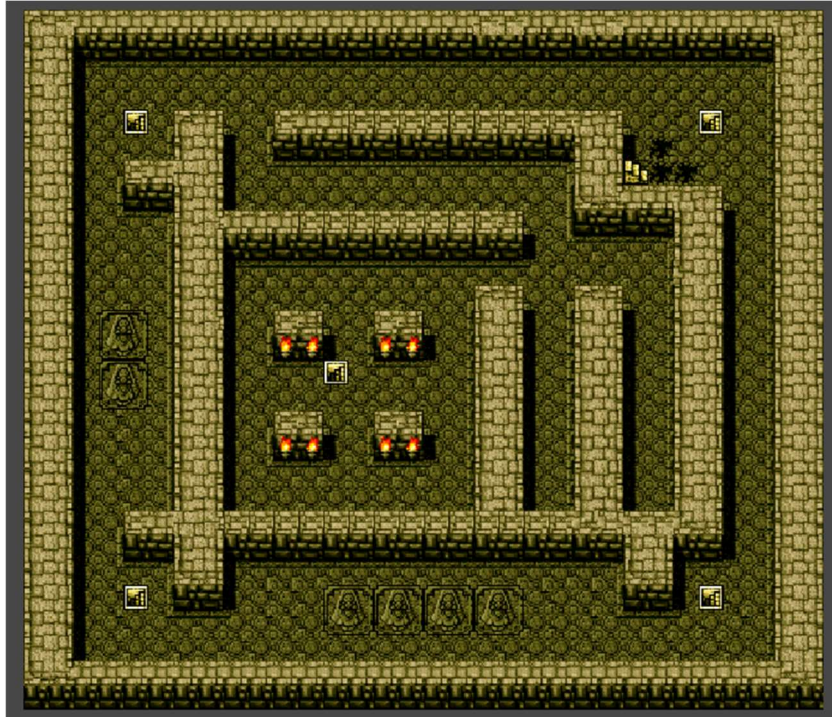


Ilustración 3.36: Galenholm P-2 de nuestro remake

Se utiliza la paleta “Mazmorra1” y la música utilizada es “Dungeon”.

3.1.11 Galenholm_P-3

Tercer piso de la parte subterránea de Galenholm.

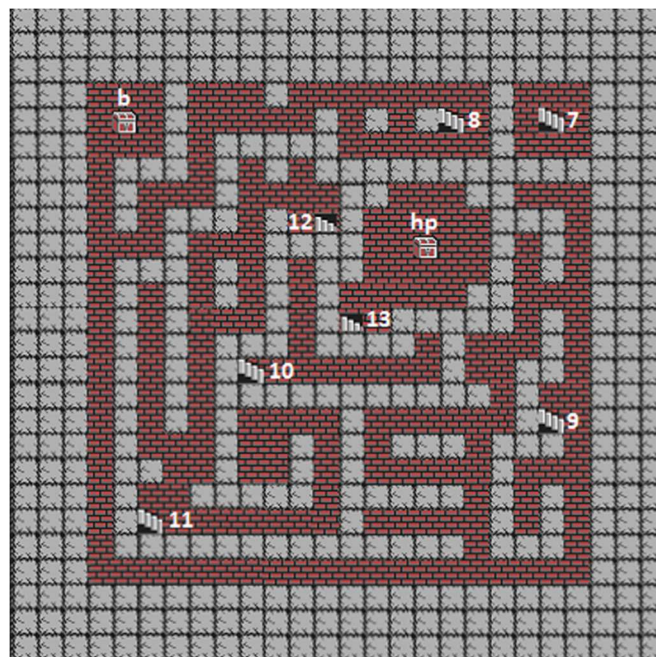


Ilustración 3.37: Galenholm P-3 MSX2



Ilustración 3.38: Galenholm P-3 de nuestro remake

Se utiliza la paleta “Mazmorra1” y la música utilizada es “Dungeon”.

3.1.12 Galenholm_P-4

Cuarta y última planta de la parte subterránea de Galenholm.

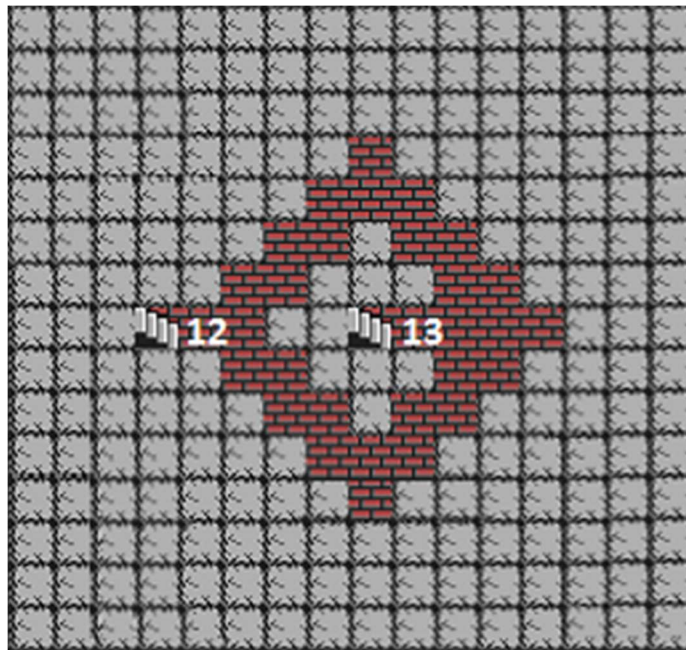


Ilustración 3.39: Galenholm P-4 MSX2

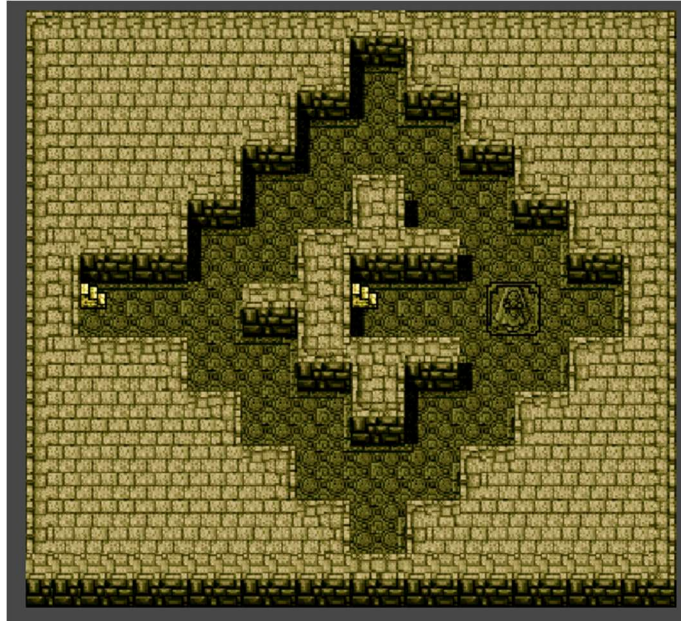


Ilustración 3.40: Galenholm P-4 de nuestro remake

Se utiliza la paleta “Mazmorra1” y la música utilizada es “Dungeon”.

3.1.13 Kol

Ahora que terminamos con Galenholm, nos dirigimos a Kol. Este mapa se sitúa al noreste del mundo.

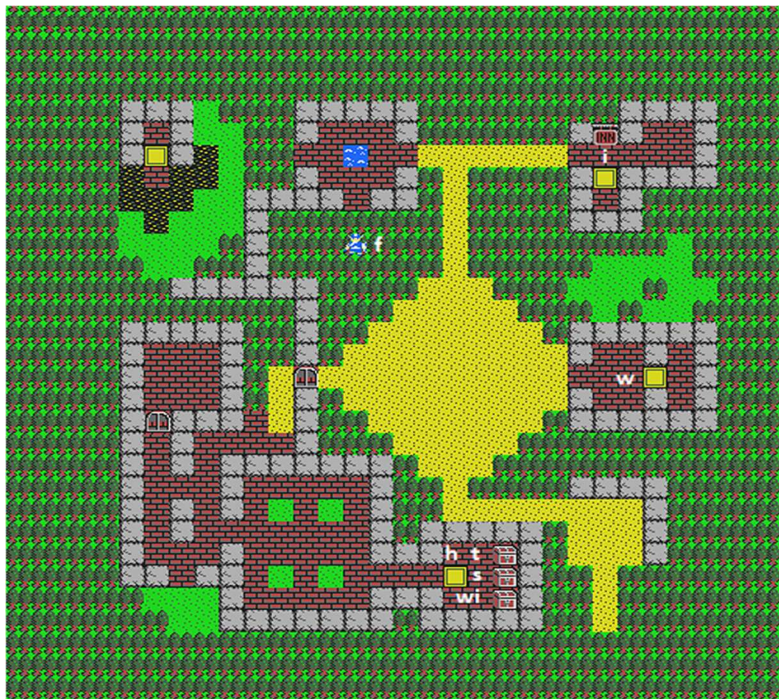


Ilustración 3.41: Kol MSX2

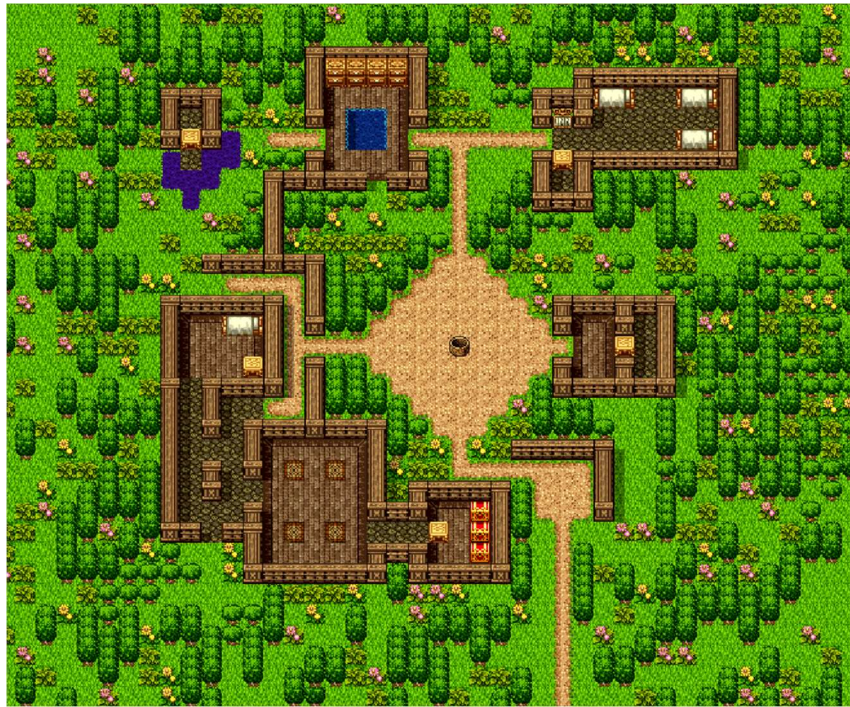


Ilustración 3.42: Kol de nuestro remake

Para hacer este mapa hemos utilizado la paleta “Pueblo2” y como música, al igual que en el resto de los pueblos se ha usado “People”.

3.1.14 Santuario_de_la_Lluvia

Este mapa no es sino la entrada del santuario. En la versión de MSX2 no existe este mapa y por tanto es único de la versión remake de Switch y en este caso, de nuestro remake.



Ilustración 3.43: Santuario de la Lluvia – remake

Para hacer este mapa hemos utilizado la paleta “Pueblo1” y como música hemos utilizado la canción “Holy Shrine” perteneciente a la segunda entrega de DQ, que podemos escuchar aquí: (Roto, Youtube, s.f.)

3.1.15 Santuario_de_la_Lluvia_P-1

Santuario de la Lluvia donde en el juego podremos conseguir el báculo de la lluvia.

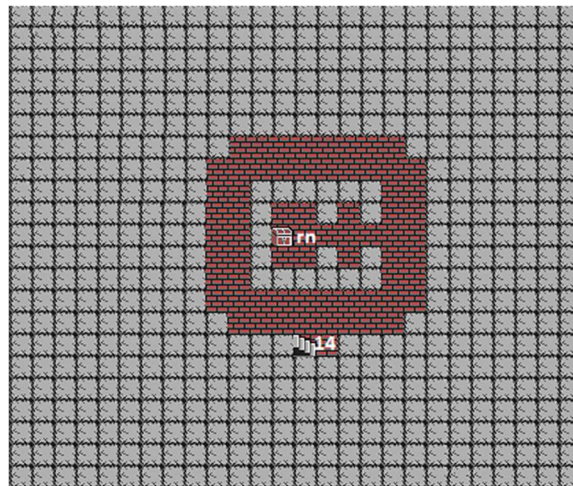


Ilustración 3.44: Santuario de la Lluvia MSX2



Ilustración 3.45: Santuario de la Lluvia P-1 de nuestro remake

Para hacer este mapa se ha utilizado una nueva paleta a la que llamaremos “Mazmorra2”:



Ilustración 3.46: Paleta Mazmorra2

Y al igual que en “Santuario_de_la_Lluvia”, se ha utilizado la canción “Holy Shrine”.

3.1.16 Cueva_del_Pantano

Esta cueva uno dos de los continentes del juego, y es donde nos enfrentaremos a uno de los jefes del juego que está custodiando a la princesa.

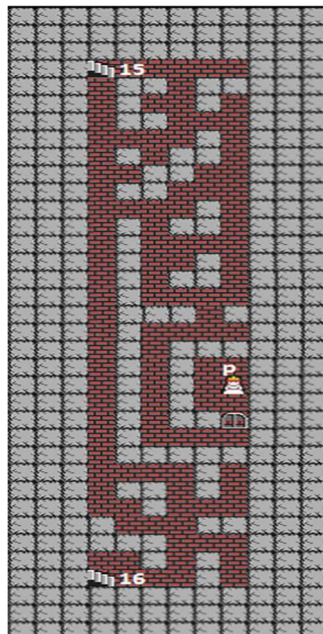


Ilustración 3.47: Cueva del Pantano MSX2

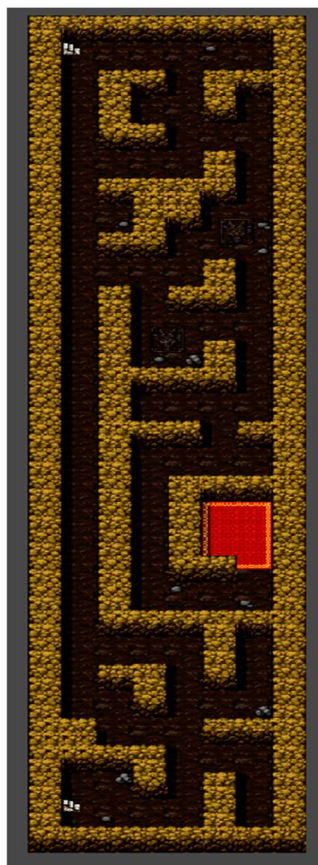


Ilustración 3.48: Cueva del Pantano de nuestro remake

Como paleta hemos utilizado una nueva paleta que llamaremos “Mazmorra2A”, y como música hemos usado “Dungeon”. A continuación, mostramos la paleta:

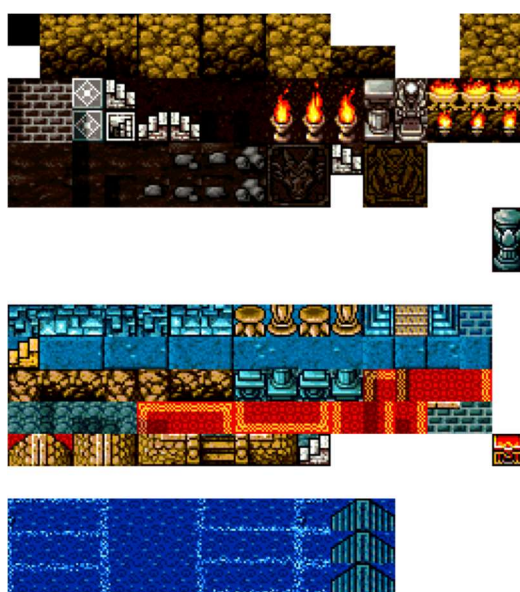


Ilustración 3.49: Paleta Mazmorra2A

3.1.17 Rimuldar

Primera ciudad a la que llegaremos al atravesar la cueva anterior. Se encuentra al sur de esta.



Ilustración 3.50: Rimuldar MSX2



Ilustración 3.51: Rimuldar de nuestro remake

En este mapa se ha usado la paleta “Pueblo1” y como música hemos utilizado “People”.

3.1.18 Santuario_Sagrado

Este santuario está lo más al sureste del mundo. Es el santuario donde una vez poseyendo el báculo de la lluvia y la piedra solar, podremos obtener el poder para enfrentar al DragonLord. Al igual que el santuario de la lluvia, este mapa es la entrada al santuario, y en el original no existe dicha entrada.



Ilustración 3.52: Santuario Sagrado de nuestro remake

3.1.19 Santuario_Sagrado_P-1

Parte interior del Santuario Sagrado. Utiliza la paleta “Mazmorra2”, y al igual que la parte superior, utiliza la canción “Holy Shrine”.

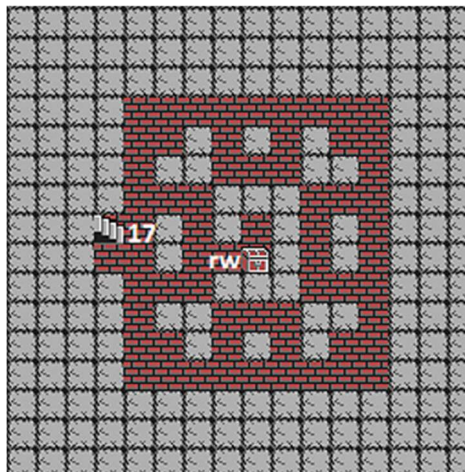


Ilustración 3.53: Santuario Sagrado MSX2

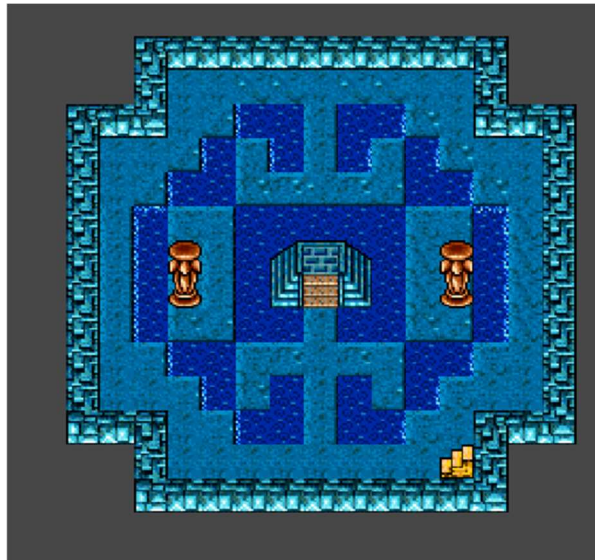


Ilustración 3.54: Santuario Sagrado P-1 de nuestro remake

3.1.20 Cueva_Montañas_Rocosas

Esta cueva se encuentra al sur de Galenholm entre las montañas. Utiliza el Canvas “Cueva”.

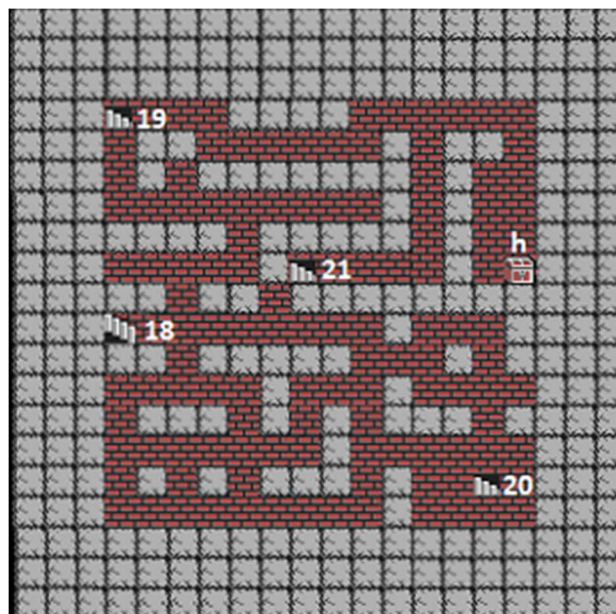


Ilustración 3.55: Cueva Montañas Rocosas MSX2

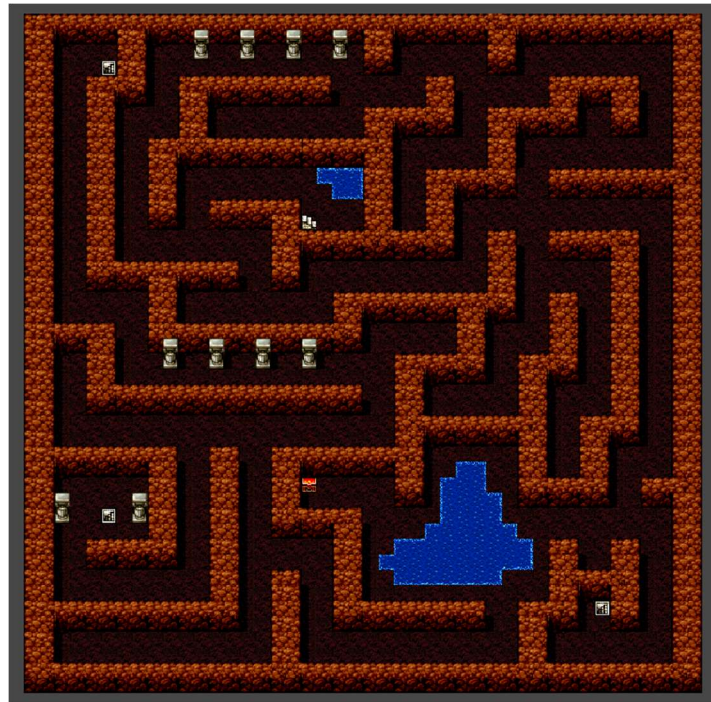


Ilustración 3.56: Cueva Montañas Rocosas de nuestro remake

Para hacer este mapa hemos utilizado la paleta “Mazmorra2” y como música se ha utilizado la canción “Dungeon”.

3.1.21 Cueva_Montañas_Rocosas_P-1

Segunda planta de la cueva de las montañas rocosas. Se utiliza la misma paleta y música.

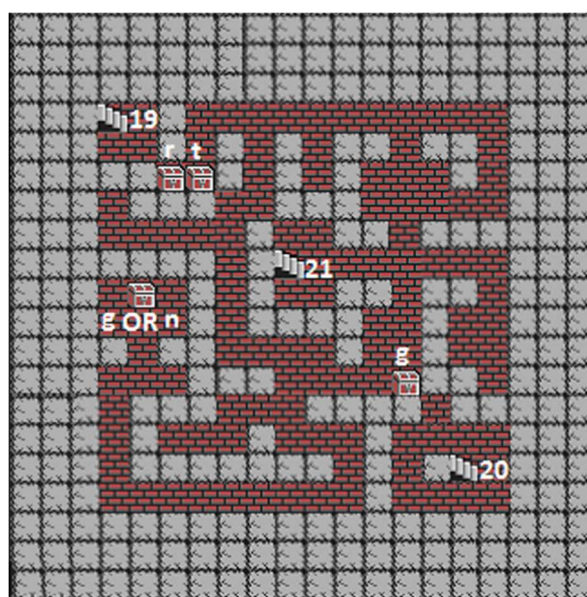


Ilustración 3.57: Cueva Montañas Rocosas P-1 MSX2

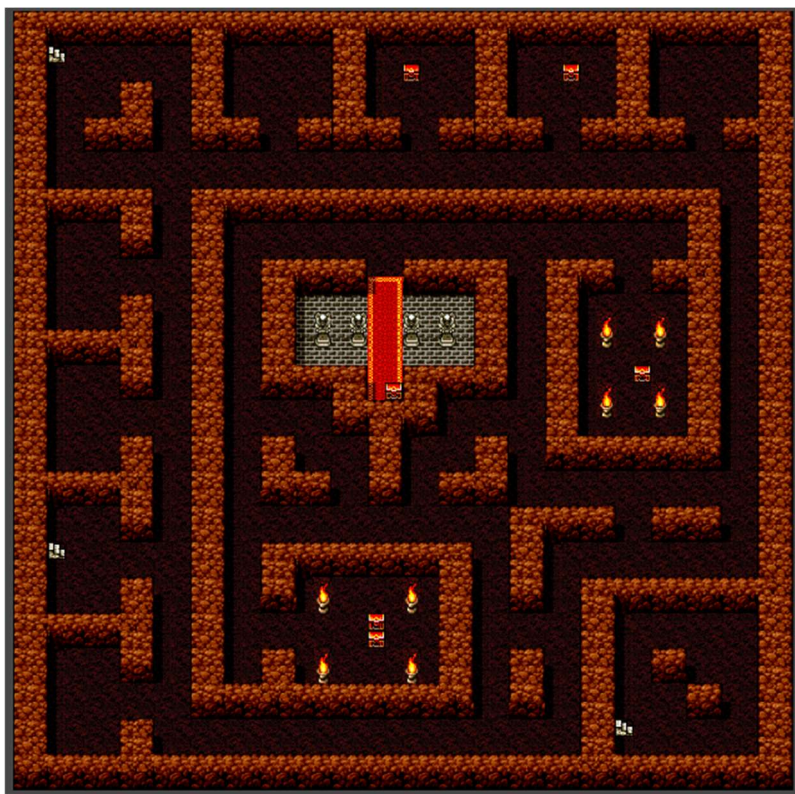


Ilustración 3.58: Cueva Montañas Rocosas P-1 de nuestro remake

3.1.22 Ciudad_Destruida_de_Domdora

Esta ciudad nos la encontramos al sur de la cueva de las montañas rocosas, en el desierto.

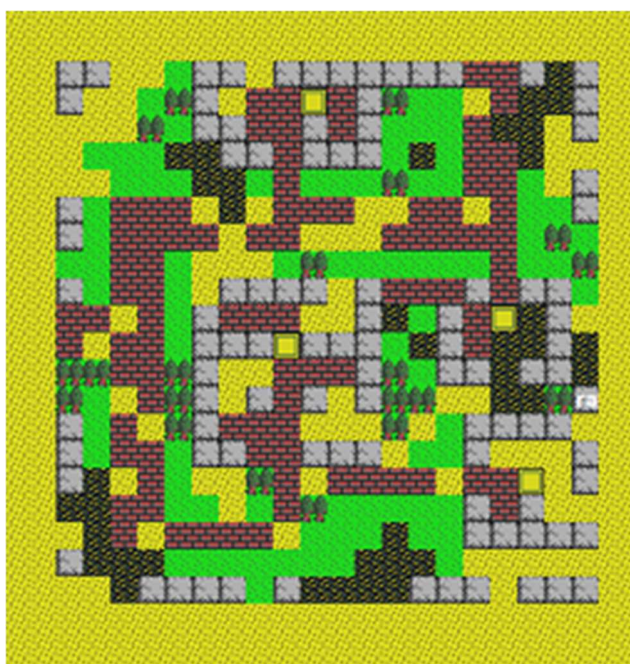


Ilustración 3.59: Ciudad Destruida de Domdora MSX2

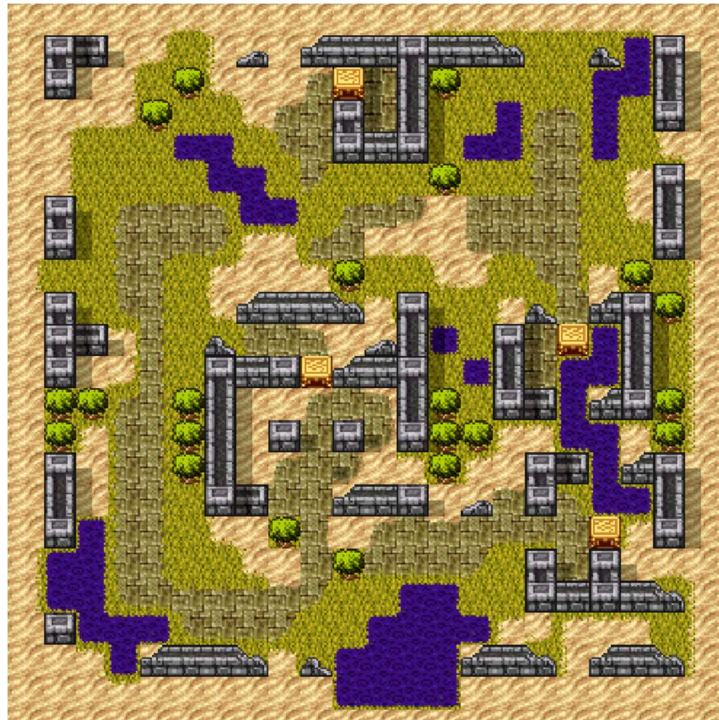


Ilustración 3.60: Ciudad Destruída de Domdora de nuestro remake

Este mapa se ha construido utilizando la paleta “Pueblo2”, y la canción elegida ha sido “Requiem” de la segunda entrega de DQ, que se puede escuchar aquí: (Roto, Youtube, s.f.)

3.1.23 Mercado / Catlin

El fuerte de Catlin (también llamado Mercado), está rodeado de montañas en la parte sur del mundo.

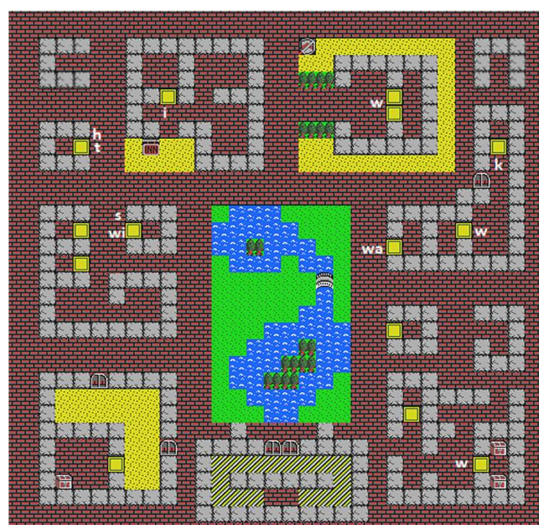


Ilustración 3.61: Mercado MSX2

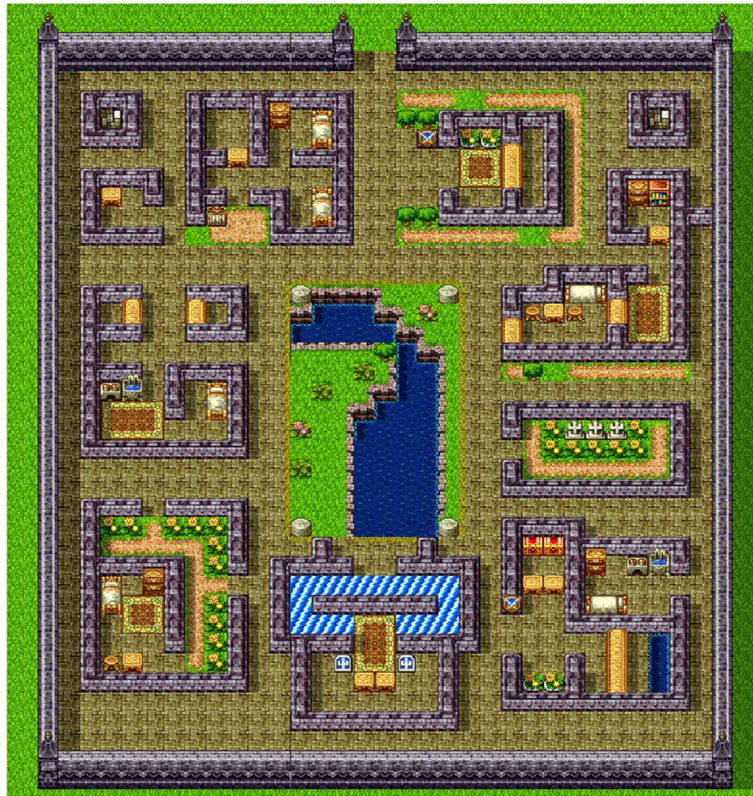


Ilustración 3.62: Mercado de nuestro remake

La paleta utilizada en este caso ha sido “Pueblo2” y la canción elegida ha sido “People”.

3.1.24 Castillo_DragonLord

Esta es la última serie de mapas del juego. Los mapas pertenecientes al castillo del enemigo final, DragonLord.

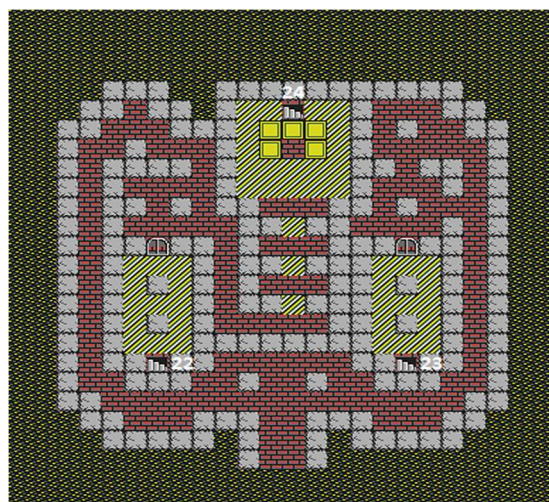


Ilustración 3.63: Castillo DragonLord MSX2

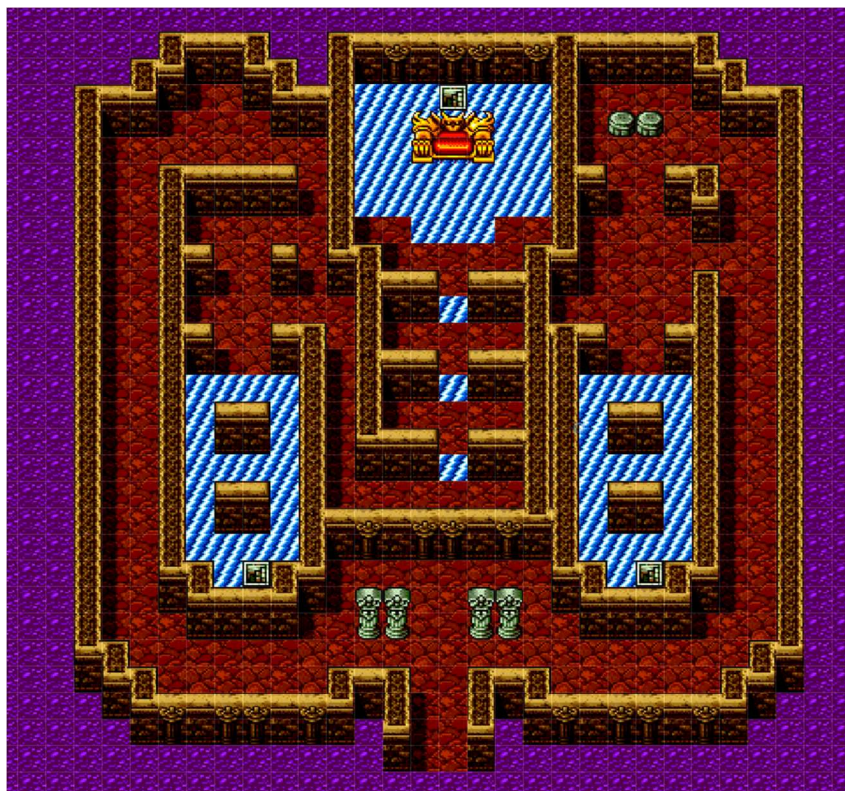


Ilustración 3.64: Castillo DragonLord de nuestro remake

Se ha utilizado una nueva paleta para crear este mapa, la llamaremos “Mazmorra3”, se muestra a continuación:



Ilustración 3.65: Paleta Mazmorra3

En cuanto a música, como en todas las mazmorras se ha optado por la canción “Dungeon”.

3.1.25 Castillo_DragonLord_P-1

Primer piso subterráneo del castillo de DragonLord. Al igual que todas las cuevas se utiliza el Canvas “Cueva”.

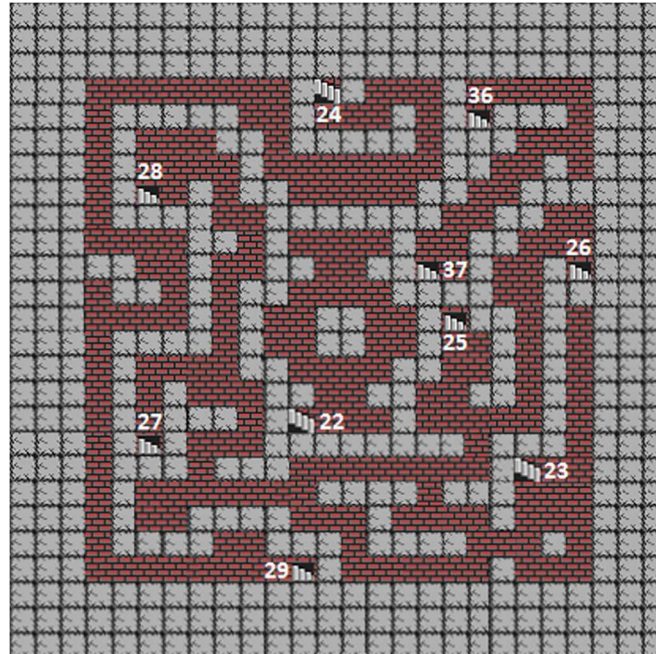


Ilustración 3.66: Castillo DragonLord P-1 MSX2

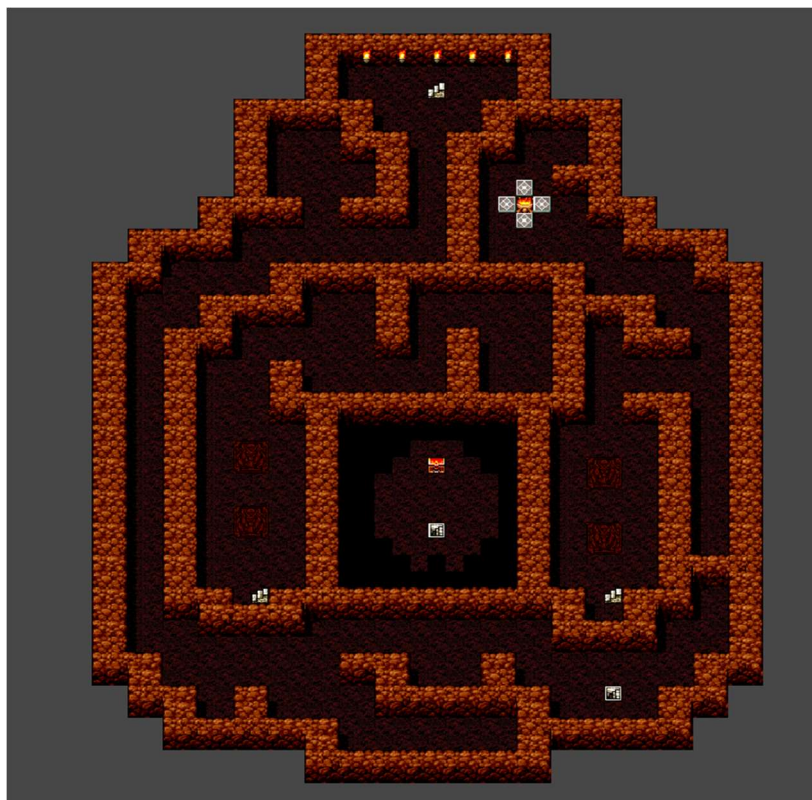


Ilustración 3.67: Castillo DragonLord P-1 de nuestro remake

Se utiliza la paleta “Mazmorra2”, y como canción se utiliza “Dungeon”.

3.1.26 Castillo_DragonLord_P-2

Segundo piso de la parte subterránea del castillo de DragonLord.

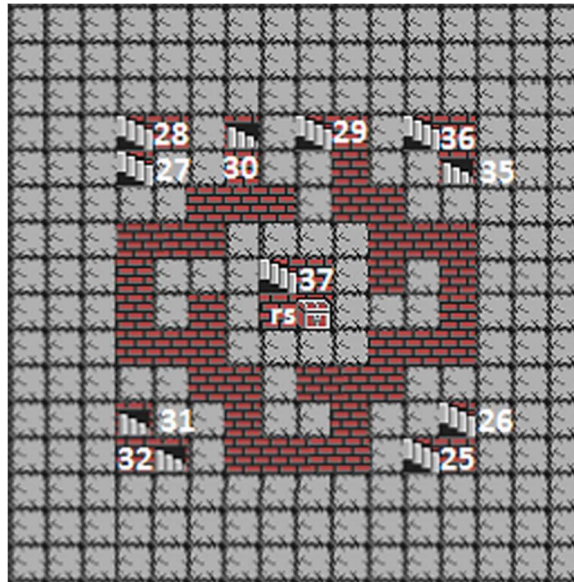


Ilustración 3.68: Castillo DragonLord P-2 MSX2

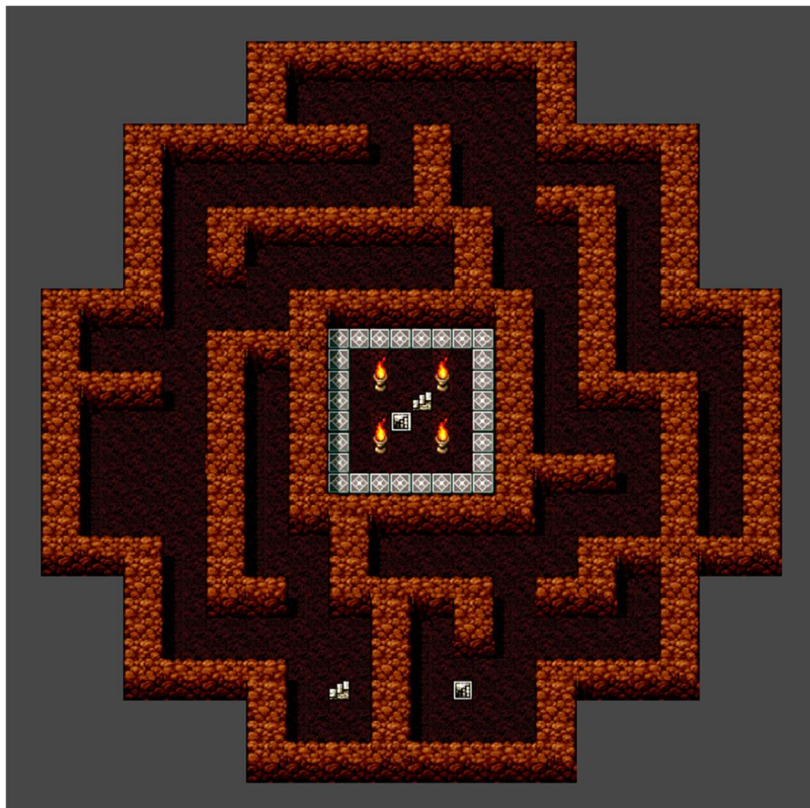


Ilustración 3.69: Castillo DragonLord de nuestro remake

Se utiliza la paleta “Mazmorra2”, y como canción se utiliza “Dungeon”.

3.1.27 Castillo_DragonLord_P-3

Tercer piso de la parte subterránea del castillo de DragonLord. Se utiliza la paleta “Mazmorra2” y la canción “Dungeon”.

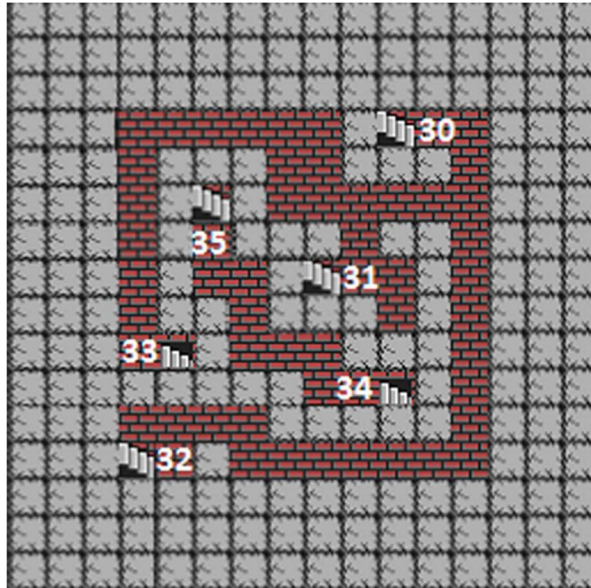


Ilustración 3.70: Castillo DragonLord P-3 MSX2



Ilustración 3.71: Castillo DragonLord P-3 de nuestro remake

3.1.28 Castillo_DragonLord_P-4

Cuarto piso de la parte subterránea del castillo de DragonLord. Se utiliza la paleta “Mazmorra2” y la canción “Dungeon”.

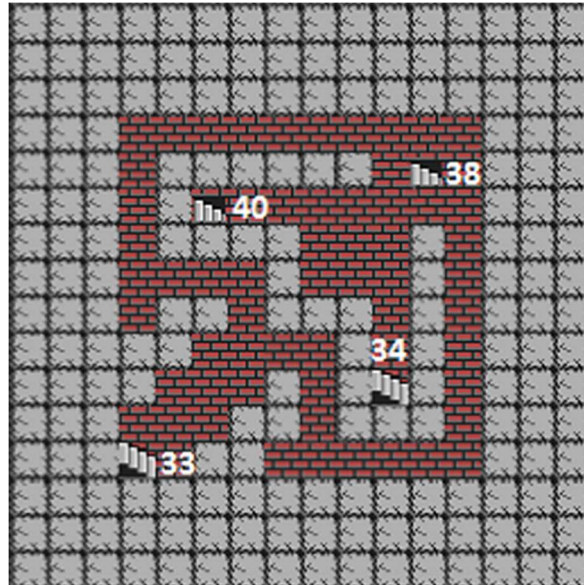


Ilustración 3.72: Castillo DragonLord P-4 MSX2



Ilustración 3.73: Castillo DragonLord P-4 de nuestro remake

3.1.29 Castillo_DragonLord_P-5A

En el quinto piso los escenarios se dividen en dos, 5A y 5B, solo que el piso 5A no es sino una trampa, donde al entrar por las escaleras volverás al mismo lugar una y otra vez hasta que decidas darte la vuelta. La parte 5A se correspondería con la parte central del original.

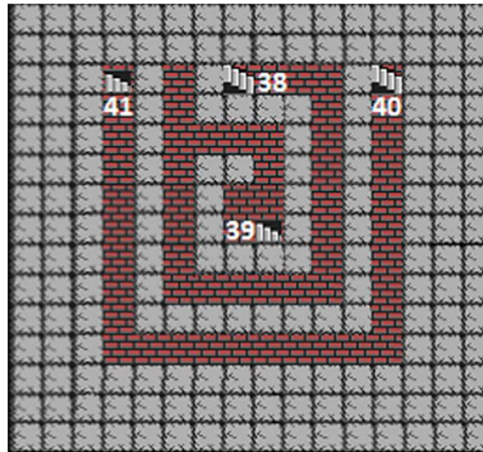


Ilustración 3.74: Castillo DragonLord P-5 MSX2

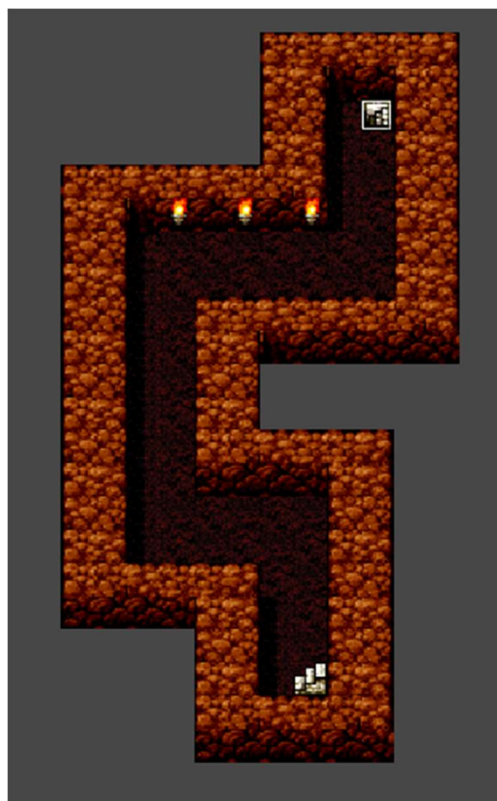


Ilustración 3.75: Castillo DragonLord P-5A de nuestro remake

3.1.30 Castillo_DragonLord_P-5B

Esta es la otra parte del quinto piso del castillo de DragonLord, la que sí nos llevará a encontrarnos con él. Corresponde con la parte de afuera del original.

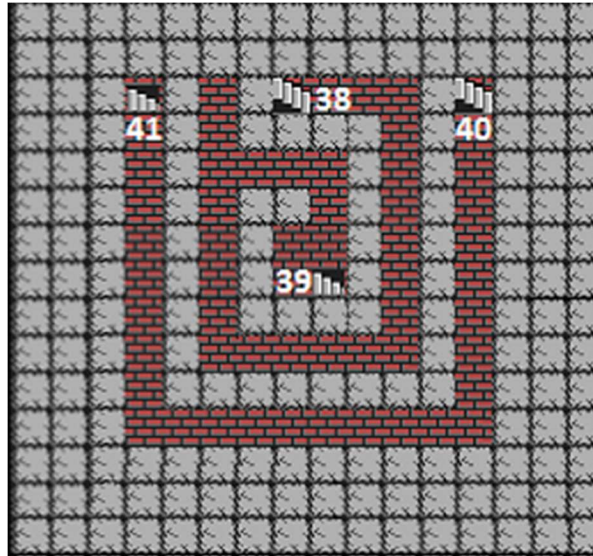


Ilustración 3.76: Castillo DragonLord P-5 MSX2

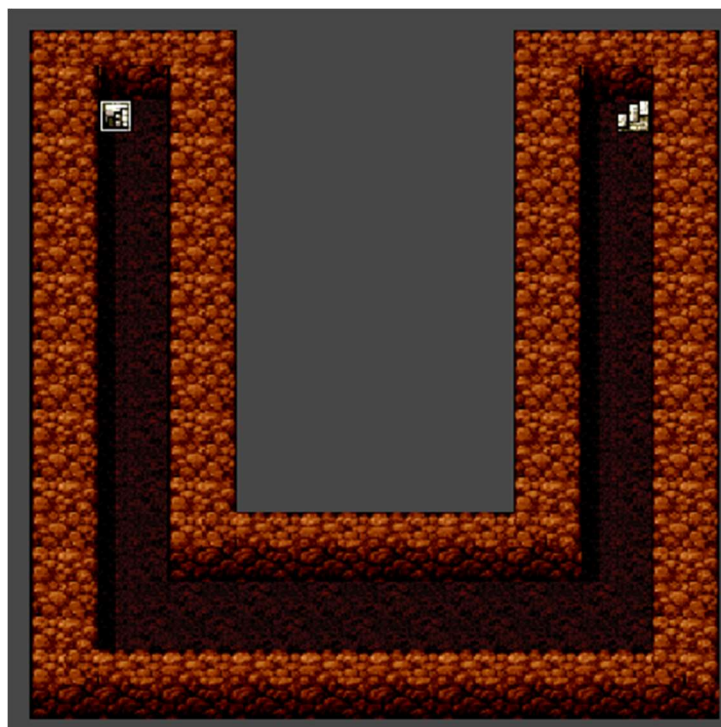


Ilustración 3.77: Castillo DragonLord P-5B de nuestro remake

Se utiliza la paleta “Mazmorra2” y la canción “Dungeon”.

3.1.31 Castillo_DragonLord_P-6B

Este piso es el último antes de llegar al jefe final, se utiliza la paleta “Mazmorra2” y la canción “Dungeon”. La parte superior del original sería equivalente a la continuación de nuestra parte 5A.

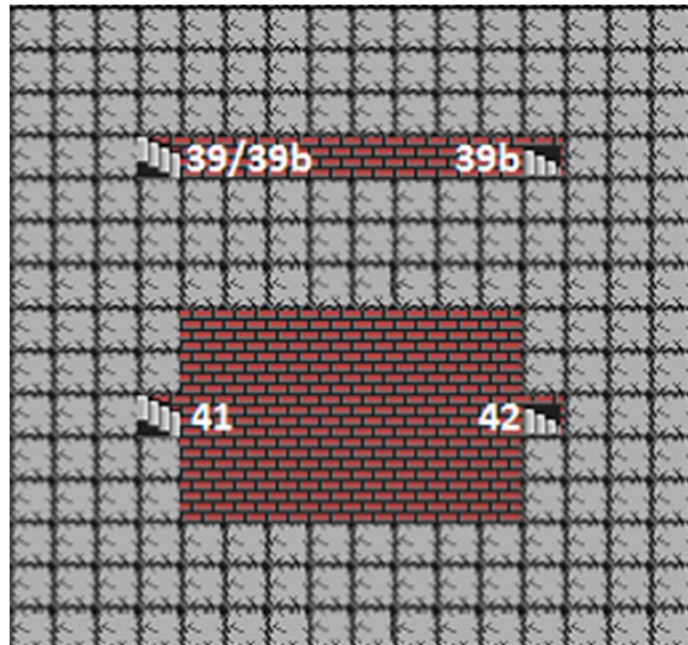


Ilustración 3.78: Castillo DragonLord P-6B MSX2

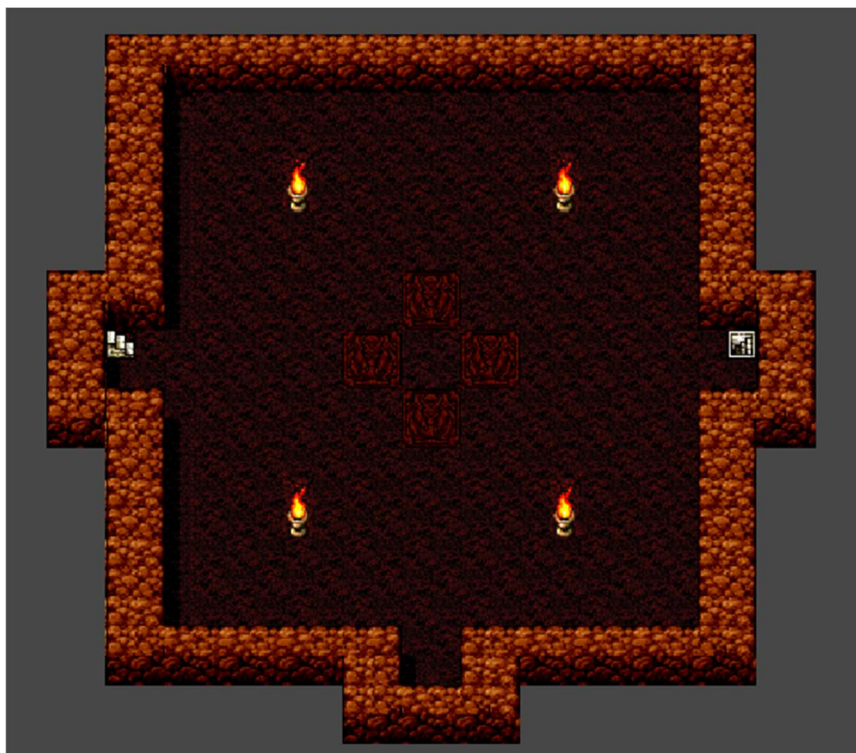


Ilustración 3.79: Castillo DragonLord P-6B de nuestro remake

3.1.32 Castillo_DragonLord_P-7

Este es el último mapa del castillo de DragonLord y del juego. Se utiliza la paleta “Mazmorra3” así como la canción “Dungeon”.

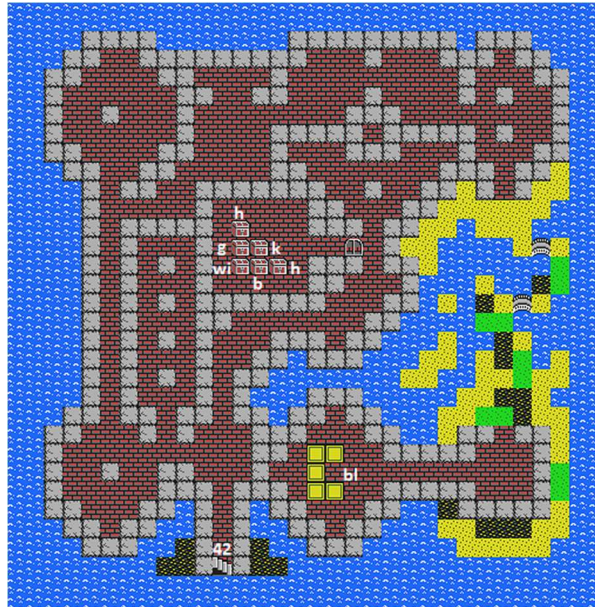


Ilustración 3.80: Castillo DragonLord P-7 MSX2

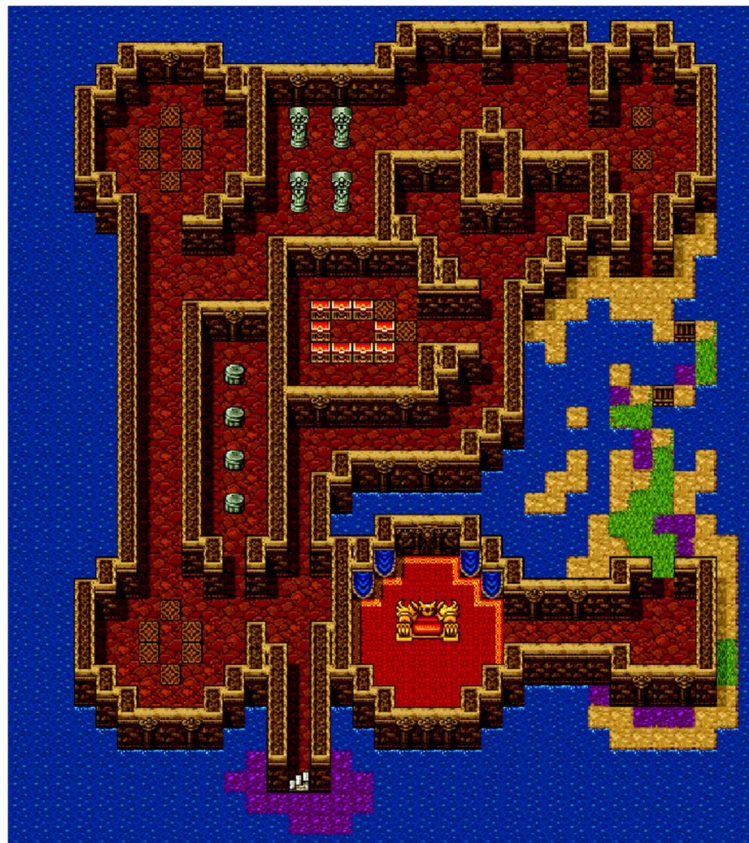


Ilustración 3.81: Castillo DragonLord P-7 de nuestro remake

3.2 Segunda Iteración – Creación del jugador y transición entre escenas.

En la segunda iteración nos encargamos de crear el prefab (una plantilla que representa un objeto y que tiene sus propias características definidas) del jugador con todo lo que conlleva, así como de crear colliders y de tratar la transición de escenas correcta de nuestro jugador, para lo que también tuvimos que empezar a crear nuestro sistema de guardado que posteriormente será ampliado.

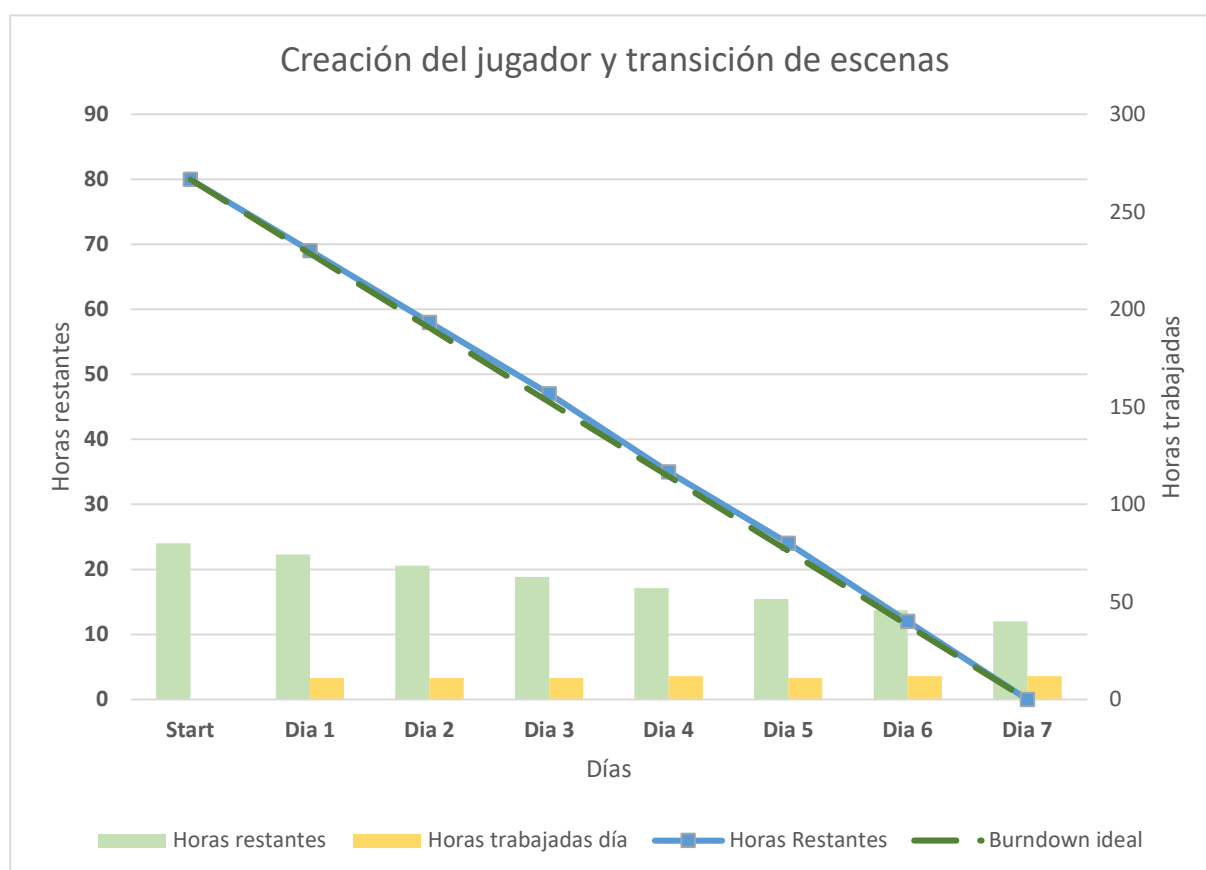


Ilustración 3.82: Diagrama Burndown - Creación del jugador y transición entre escenas

Esta parte del proyecto nos llevó en torno a 75-80 horas, aunque si tuviésemos en cuenta el tiempo dedicado a solventar los problemas que fueron surgiendo quizás llegaríamos a las 85-90.

Primero veremos la creación de nuestro jugador y posteriormente la creación de colliders y transición de escenas.

3.2.1 Creación del Prefab del Jugador

Para crear nuestro prefab de jugador creamos un GameObject vacío, al cual añadimos diferentes componentes que nos servirán tanto para la parte visible de nuestro jugador, como para mover a nuestro jugador.

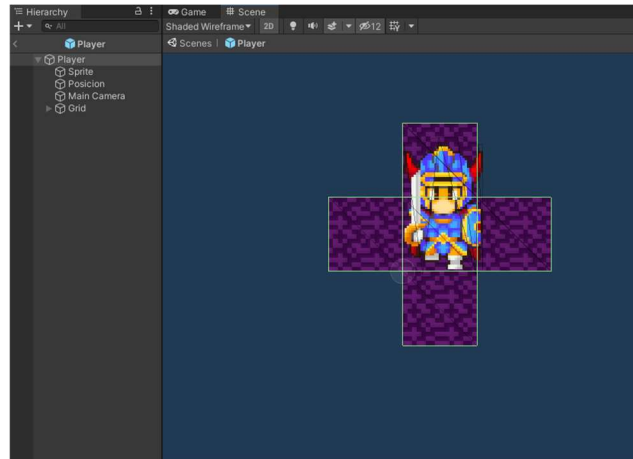


Ilustración 3.83: Jugador

Como se ve en la imagen, nuestro jugador está compuesto por varios elementos:

- **Sprite:** Es, como su nombre indica, donde colocaremos el Sprite inicial del jugador, que en nuestro caso es el que se ve en la imagen.
- **Posición:** Se trata de un objeto vacío cuya funcionalidad radicará (como ya veremos cuando expliquemos los scripts) en moverse por el mapa y realizar las comprobaciones de colisión, así como de mover nuestro Sprite por la escena (el Sprite se moverá hacia la posición).
- **Main Camera:** se trata de la cámara del juego. A parte del menú inicial ninguna escena tiene cámara, ya que la cámara la incorporará el jugador para conseguir que esta lo siga fácilmente, y está preconfigurada para que se vea del modo que queremos.
- **Grid:** El prefab del jugador incorporará un Tilemap (las casillas moradas que vemos a los pies del jugador) que servirán para evitar que los NPCs (cuando los creamos en sprints posteriores) se muevan hacia nuestra misma casilla bloqueándonos el movimiento.



Ilustración 3.84: Sprites jugador

En la imagen de arriba vemos los 8 sprites diferentes con los que cuenta nuestro jugador. Estos, bien manejados en nuestros scripts, nos permitirán animar al jugador.

En este punto necesitamos crear un script para manejar el movimiento de nuestro personaje, así como para animarlo. Para ello creamos el script “Movimiento”.

```
Script de Unity (1 referencia de recurso) | 99+ referencias
public class Movimiento : MonoBehaviour
{
    //Restringimos el movimiento a una casilla cada 0.2 segundos
    private float timer = 0f;
    private const float tiempoEspera= 0.2f;

    //Sirve para bloquear el movimiento una vez llegamos a un sitio
    public static bool bloqueado = false;
    public static bool comprando = false;

    //sirve para comprobar si ya nos hemos movido en este turno
    private bool movido = false;

    private float velocidad = 5f;
    private Gamepad gamepad;
    public static Transform posicion;
```

Ilustración 3.85: Código del Script "Movimiento" 1

Aquí podemos ver que utilizamos un **timer** que contará el tiempo que ha pasado y un **tiempo de espera** que capará nuestro movimiento a una casilla cada **0.2 segundos**, ya que de lo contrario nuestro personaje, en cuestión de menos de un segundo, podría aparecer en la otra punta del mapa ya que obviamente la función **Update** (donde se lleva a cabo todo el código, a excepción de las corrutinas) se

ejecuta muchas veces por segundo. A parte de esto, también tenemos la variable **movido**, para que en caso de que ya nos hayamos movido en ese turno no lo hagamos de nuevo si pulsamos varias teclas a la vez.

La variable **posición** es la que vimos en la imagen de antes donde se mostraba un GameObject llamado posición, junto con la variable **velocidad**, son las variables que harán que nuestro jugador se mueva por la escena sin problemas y sin movimientos bruscos (teletransporte) del jugador.

También vemos que tenemos un par de variables estáticas. En este punto solo nos interesa la primera, bloqueado, ya que la otra fue implementada en un Sprint posterior. **Bloqueado** se encargará de bloquear nuestro movimiento, y es estática puesto que necesitaremos que, por ejemplo, los NPCs bloqueen nuestro movimiento al hablar con ellos, o que cuando entremos en unas escaleras se bloquee nuestro movimiento (aunque en este último caso lo hace el propio script).

En cuanto a la variable de tipo **Gamepad**, es utilizada para leer los movimientos del mando en caso de estar usando uno.

A parte de lo que hemos explicado hasta ahora, también creamos un enum llamado ultimo_mov, que obviamente representa el último movimiento hecho, y nos servirá para animar a nuestro personaje. También tenemos variables para todos los sprites de este.

```
public static ultimo_mov ultimo = ultimo_mov.abajo;
public Sprite frente_0;
public Sprite frente_1;
public Sprite espalda_0;
public Sprite espalda_1;
public Sprite izquierda_0;
public Sprite izquierda_1;
public Sprite derecha_0;
public Sprite derecha_1;
```

Ilustración 3.86: Código del Script "Movimiento" 2

Todas estas variables y algunas de las que hemos mostrado antes necesitan ser públicas para que podamos asignarles desde el editor de Unity, por ejemplo, en el caso de esta última imagen, los diferentes sprites de nuestro personaje.

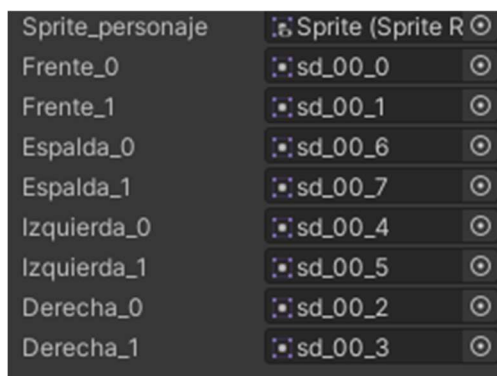


Ilustración 3.87: Código del Script "Movimiento" 3

Lo primero que hacemos cuando inicia una escena es lo siguiente:

```

void Start()
{
    if (SistemaGuardado.ExisteArchivo() is false)
    {
        ultimo = ultimo_mov.arriba;
    }
    posicion = this.gameObject.transform.Find("Posicion");

    posicion.parent = null;
    StartCoroutine(Animacion());
    gamepad = Gamepad.current;
}

```

Ilustración 3.88: Código del Script "Movimiento" 4

En el método **Start**, el cual se ejecuta antes del primer frame, asignamos a nuestro script la posición, y posteriormente indicamos que esta salga de nuestro jugador, es decir, que deje de tener como padre al GameObject Player que vimos antes. Esto debe de hacerse así para que el objeto Player pueda seguir al objeto posición, de otro modo la posición de este último sería relativa a Player y no global.

Por otro lado, asignamos a la variable gamepad el controlador actual, haya o no haya uno. Posteriormente se comprobará si es nulo y se actuará en consecuencia.

Además de eso, iniciamos la corrutina **Animación**, que se sobreentiende será la que anime a nuestro jugador.

```
1 referencia
IEnumerator Animacion()
{
    while (true)
    {
        animar();
        yield return new WaitForSeconds(0.4f);
    }
}
```

Ilustración 3.89: Código del Script "Movimiento" 5

Esta a su vez se encarga de, en un bucle infinito, llamar a la función **animar** que se encarga de cambiar los sprites.

```
void animar()
{
    switch (ultimo)
    {
        case ultimo_mov.arriba:
            if (sprite_personaje.sprite == espalda_1)
            {
                sprite_personaje.sprite = espalda_0;
            }
            else
            {
                sprite_personaje.sprite = espalda_1;
            }
            break;

        case ultimo_mov.abajo:
            if (sprite_personaje.sprite == frente_1)
            {
                sprite_personaje.sprite = frente_0;
            }
            else
            {
                sprite_personaje.sprite = frente_1;
            }
            break;

        case ultimo_mov.derecha:
            if (sprite_personaje.sprite == derecha_1)
            {
                sprite_personaje.sprite = derecha_0;
            }
            else
            {
                sprite_personaje.sprite = derecha_1;
            }
            break;

        case ultimo_mov.izquierda:
            if (sprite_personaje.sprite == izquierda_1)
            {
                sprite_personaje.sprite = izquierda_0;
            }
            else
            {
                sprite_personaje.sprite = izquierda_1;
            }
            break;
    }
}
```

Ilustración 3.90: Código del Script "Movimiento" 6

Según el último movimiento, que se actualizará obviamente cada vez que nos movamos, comprobará si corresponde cambiar a uno u otro Sprite para dar el efecto animado.



Ilustración 3.91: Animación jugador

Luego tenemos el método **Update** que se ejecuta en cada frame. Lo primero que nos encontramos es aumentar el valor del timer según el tiempo transcurrido, así como, movernos hacia la posición de nuestro objeto posición.

```
void Update()
{
    timer += Time.deltaTime;

    transform.position = Vector3.MoveTowards(transform.position, posicion.position, velocidad * Time.deltaTime);
}
```

Ilustración 3.92: Código del Script "Movimiento" 7

A continuación de este trozo de código nos encontramos con la parte que comprueba si estamos intentando movernos en una dirección o en otra.

```
// W
//Movimiento pulsando las teclas A S D o bien utilizando un mando.
if (!bloqueado)
{
    if (timer > tiempoEspera)
    {
        if (Gamepad.current != null)
        {
            gamepad = Gamepad.current;
            if (Input.GetKey(KeyCode.W) || gamepad.dpad.up.IsPressed() || (gamepad.leftStick.ReadValue().y >= .5f && Mathf.Abs(gamepad.leftStick.ReadValue().x) < Mathf.Abs(gamepad.leftStick.ReadValue().y)))
            if ((Input.GetKey(KeyCode.A) || gamepad.dpad.left.IsPressed() || (gamepad.leftStick.ReadValue().x <= -.5f && Mathf.Abs(gamepad.leftStick.ReadValue().x) > Mathf.Abs(gamepad.leftStick.ReadValue().y))) &&
            if ((Input.GetKey(KeyCode.S) || gamepad.dpad.down.IsPressed() || (gamepad.leftStick.ReadValue().y <= -.5f && Mathf.Abs(gamepad.leftStick.ReadValue().x) < Mathf.Abs(gamepad.leftStick.ReadValue().y))) &&
            if ((Input.GetKey(KeyCode.D) || gamepad.dpad.right.IsPressed() || (gamepad.leftStick.ReadValue().x >= .5f && Mathf.Abs(gamepad.leftStick.ReadValue().x) > Mathf.Abs(gamepad.leftStick.ReadValue().y))) &&
            else
            {
                if (Input.GetKey(KeyCode.W))
                if (Input.GetKey(KeyCode.A) && !movido)
                if (Input.GetKey(KeyCode.S) && !movido)
                if (Input.GetKey(KeyCode.D) && !movido)
            }
            movido = false;
        }
    }
}
```

Ilustración 3.93: Código del Script "Movimiento" 8

Aquí lo que hacemos primero es comprobar si ha pasado el tiempo suficiente como para poder movernos, es decir, si ya han pasado los 0.2 segundos que tenemos que esperar entre movimiento y movimiento. Posteriormente, en caso de poder movernos comprobamos si Gamepad.current es nulo; Esto lo hacemos para asegurarnos de no intentar leer los datos de nuestra variable gamepad anteriormente asignada y que en su momento era válida si ahora se ha desconectado el mando, se ha cambiado, etc. Si hay un gamepad conectado comprobaremos tanto las posibles acciones del teclado como las del mando, si por el contrario no tenemos un mando conectado, utilizaremos solo las del teclado.

```

gamepad = Gamepad.current;
if (Input.GetKey(KeyCode.W) || gamepad.dpad.up.IsPressed() || (gamepad.leftStick.ReadValue().y >= .5f && Mathf.Abs(gamepad.leftStick.ReadValue().x) < .5f))
{
    if (ultimo != ultimo_mov.arriba)
    {
        sprite_personaje.sprite = espalda_0;
        ultimo = ultimo_mov.arriba;
    }
    if (!Physics2D.OverlapCircle(posicion.position + new Vector3(0.5f, 1, 0), .2f, colliders))
    {
        posicion.position += new Vector3(0, 1, 0);
        timer = 0;
        movido = true;
        ultimo = ultimo_mov.arriba;
        Bloqueado();
        if (SceneManager.GetActiveScene().name == "Ciudad_Destruida_De_Domdora" && posicion.position == new Vector3(11, -17, 0))
        {
            bloqueado = true;
            StartCoroutine(BatallaDomdora());
        }
        else
        {
            if (5 > UnityEngine.Random.Range(0, 100) && bloqueado == false)
            {
                print("Inicio batalla");
                bloqueado = true;
                StartCoroutine(spawnBatalla());
            }
        }
    }
}

```

Ilustración 3.94: Código del Script "Movimiento" 9

En este código vemos como, primeramente, cambiamos el Sprite del personaje, así como la dirección de último movimiento, ya que podamos o no movernos, queremos que se refleje la intención de movimiento en el juego.

En el siguiente if, comprobamos si en caso de movernos en esa dirección chocaríamos con un collider, o, mejor dicho, contra un objeto de la capa indicada, en este caso la capa Collider.

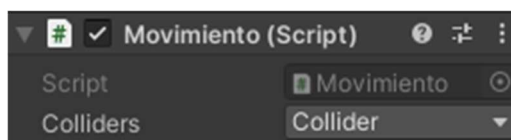


Ilustración 3.95: Capa para colliders del jugador

Para ello hemos hecho uso de **Physics2D**, que nos proporciona opciones y métodos para tratar con físicas 2D. En caso de no colisionar movemos la posición en la dirección correspondiente, reiniciamos el timer para empezar la espera de 0.2 segundos, y marcamos movido como true, para evitar, como ya dijimos antes, que podamos movernos 2 o más casillas en un turno. Luego llamamos a la función **Bloqueado**, que lo que hace es acceder a las funciones que explicamos en la sección 3.1 de la documentación. Las funciones estáticas que poseían los managers y que devolvían true cuando se entrase en una escalera o cuando se cambiase de mapa, y con ello el jugador bloqueará su movimiento. El resto de código es perteneciente a sprints posteriores, cuando hagamos la parte de batalla.

```
8 referencias
public void Bloqueado()
{
    switch (SceneManager.GetActiveScene().name)
    {
        default:
            break;
        case "World_Map":
            bloqueado = Manager_World_Map.entrar_lugar(posicion.position);
            break;
        case "Breconary":
            bloqueado = Manager_Breconary.entrar_lugar(posicion.position);
            break;
        case "Castillo_P0":
            bloqueado = Castillo_P0_manager.entrar_lugar(posicion.position);
            break;
        case "Castillo_P1":
            bloqueado = Castillo_P1_manager.entrar_lugar(posicion.position);
            break;
        case "Castillo_P-1":
            bloqueado = Manager_Castillo_P_menos1.entrar_lugar(posicion.position);
            break;
        case "Cueva de Erdrick":
            bloqueado = Manager_Cueva_Erdrick.entrar_lugar(posicion.position);
            break;
    }
}
```

Ilustración 3.96: Código del Script "Movimiento" 10

El método no aparece completo, pero nos podemos hacer a la idea ya que es lo mismo para cada una de las escenas.

Con esto hecho, una vez colocados los colliders ya tendremos todo listo para que nuestro jugador se mueva por los mapas.

3.2.2 Creación de colliders y transición de escenas

En esta parte del sprint primero definimos todos los colliders de los mapas y posteriormente nos dedicamos a testear la transición de escenas.

Para hacer los colliders primeramente creamos una nueva capa llamada “Collider”, como ya dijimos en la anterior subsección. Una vez hecha la capa, creamos un nuevo Tilemap (igual que el que utilizamos para pintar los mapas) y pintamos sobre él las zonas que queremos sean colliders.

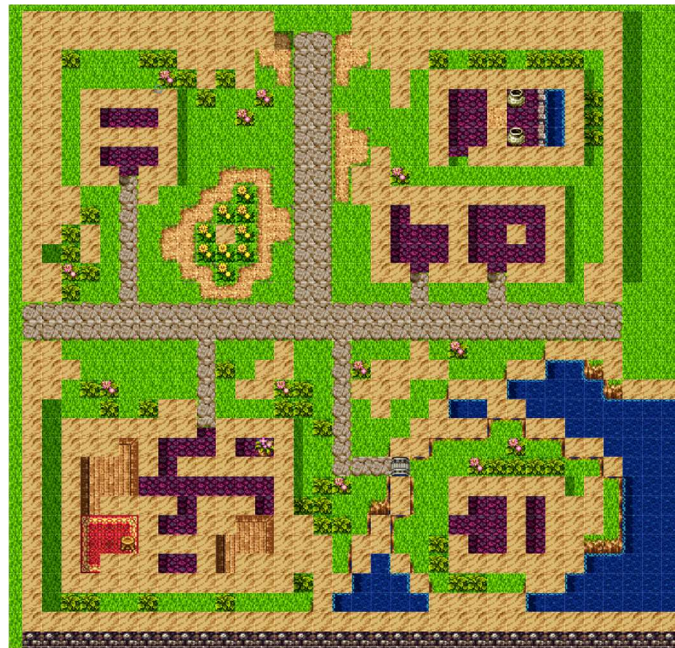


Ilustración 3.97: Colliders Breconary

Una vez pintadas las zonas que nos interesa no poder atravesar, añadimos a nuestro Tilemap los componentes:

- **Tilemap Collider 2D:** Básicamente es lo que le da la propiedad de actuar como un collider a nuestro Tilemap, es el que hace que al comprobar con Physics2D si colisiona o no, diga que sí cuando ocurra.
- **Rigidbody 2D:** En este caso debemos marcarlo como “estático”, ya que, si no, al estar emulando 2D en un espacio 3D, este caería hasta el infinito.
- **Composite Collider 2D:** Sirve para hacer nuestros colliders muchísimo, muchísimo más eficientes, ya que en vez de hacer de cada casilla un collider con sus 4 vértices, coge los vértices exteriores para toda la figura formada.

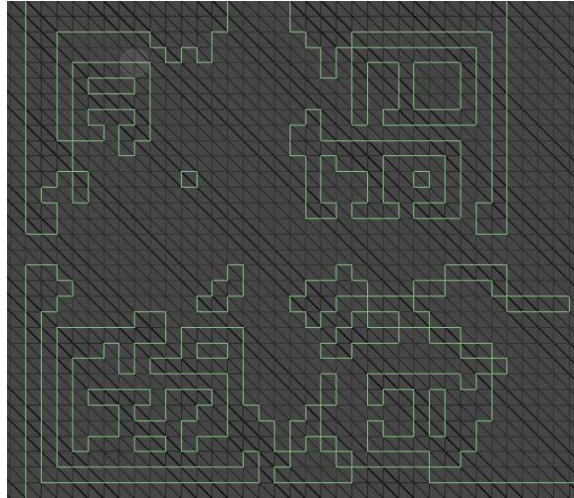


Ilustración 3.98: Colliders Breconary usando Composite

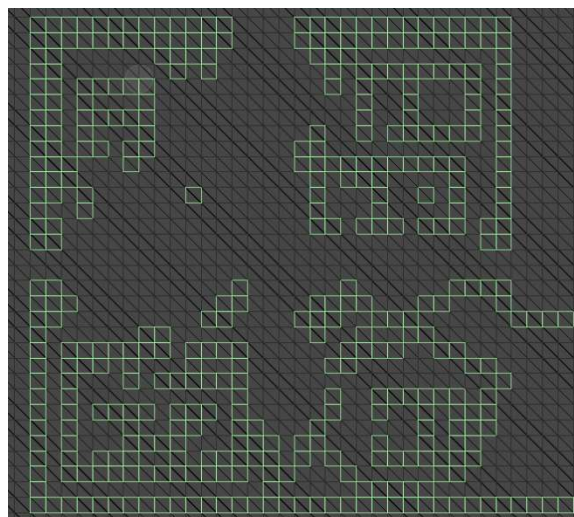


Ilustración 3.99: Colliders Breconary sin usar Composite

A parte de estos elementos, tenemos que asignar a nuestro Tilemap la capa que queramos actúe como collider, en nuestro caso “Collider”.

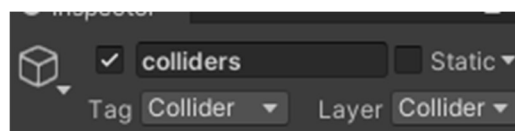


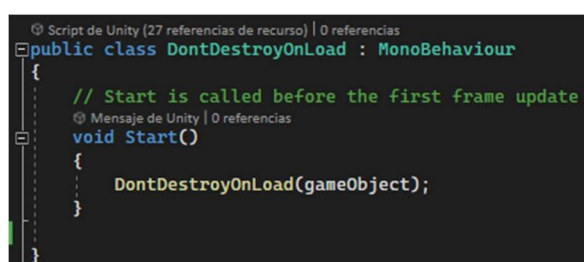
Ilustración 3.100: Asignación de capa Collider

En este punto nuestro jugador ya es capaz de moverse por el mapa y chocar con los colliders evitando que pase a través de ellos, no mostraremos todos los

colliders que hemos puesto en los mapas, pero obviamente han sido las zonas de agua, las paredes, estructuras y demás.

En cuanto a los managers mostraremos ejemplos de las funcionalidades nuevas implementadas (solo un ejemplo por caso, porque en la mayoría de las ocasiones son iguales), así como de los cambios a antiguos.

Lo primero que tuvimos que tener en cuenta era que al realizar cambios de escena entre escenas que pertenecen al mismo conjunto, como puede ser la parte subterránea de Galenholm, o los 3 pisos del castillo inicial; Tuvimos que tener en cuenta que si cargamos una partida existente, en algunos casos apareceremos en un piso, otros en otro, no podemos asumir que siempre aparezcamos en el mismo lugar, por tanto, necesitamos tener un AudioSource con la canción que queramos que se escuche en cada escena. Todas las escenas del mismo conjunto comparten una misma canción, es decir, no queremos que la música comience desde el principio cada vez que cambiemos entre escenas que sean del mismo conjunto, y para ello necesitamos hacer que los objetos no se destruyan al cambiar de escena, para hacer esto posible utilizamos un nuevo script que lo añadiremos a aquellos objetos que no deseemos se destruyan al cambiar de escena. Lo llamamos **DontDestroyOnLoad**, y su única función es darle la propiedad de no destruirse al cambiar de escena a un objeto.

A screenshot of the Unity Inspector showing the DontDestroyOnLoad script. The script is a C# class that inherits from MonoBehaviour. It has a Start method that calls DontDestroyOnLoad(gameObject). The script is attached to a GameObject, and the DontDestroyOnLoad method is highlighted in yellow.

```
Script de Unity (27 referencias de recurso) | 0 referencias
public class DontDestroyOnLoad : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        DontDestroyOnLoad(gameObject);
    }
}
```

Ilustración 3.101: Script DontDestroyOnLoad

Al añadir este script a un objeto hace que la canción continúe sin parar al cambiar a una escena del mismo conjunto, pero en dicha escena también hay una canción igual esperando a ser reproducida. Para evitar que dos canciones iguales suenen a la vez, tenemos que destruir la de la escena a la que nos movemos, y aquí es donde entra en juego el sistema de guardado.

La utilidad que tendrá el sistema de guardado en este caso es que podemos guardar cada vez que cambiemos de escena de qué escena venimos, ya que podría ser que vengamos del menú de inicio y acabemos de cargar la partida, en cuyo caso no queremos que se elimine la canción que existe en la escena, pero si venimos de otra escena del mismo conjunto, sí. Para esto empezamos a crear nuestro sistema de guardado y añadimos las funciones **LoadUltimaEscena** y **SetUltimaEscena**, para cargar y modificar los datos de esta respectivamente.

Al cambiar de escena, cuando llamamos a la función **AnimacionCambioEscena**, guardamos la escena que estamos abandonando:

```
2 referencias
IEnumerator AnimacionCambioEscena(string escena)
{
    cambioEscena.SetTrigger("Start");

    yield return new WaitForSeconds(1.1f);

    SistemaGuardado.SetUltimaEscena(SceneManager.GetActiveScene().name);

    float[] aux = new float[3];
    aux[0] = posicion.position.x;
    aux[1] = posicion.position.y;
    aux[2] = posicion.position.z;
    SistemaGuardado.SetUltimaPos(aux);

    SceneManager.LoadScene(escena);
}
```

Ilustración 3.102: AnimacionCambioEscena

También guardamos la posición desde la que cambiamos de escena en caso de estar en una escena donde tengamos que hacer uso de los atributos mencionados en el anterior sprint, “escalera”.

Y, al cargar la siguiente escena, utilizamos estos datos para eliminar, o no, la canción.

```
10 Mensaje de Unity | 0 referencias
void Start()
{
    Movimiento.bloqueado = false;
    StartCoroutine(InicioJuego());
    if (SistemaGuardado.LoadUltimaEscena() == "Castillo_P0")
    {
        Destroy(cancion.gameObject);
    }
    else
    {
        lugarAparicion();
    }
}
```

Ilustración 3.103: Código Script "Castillo_P1_manager"

Como también mencionamos arriba, puede ser que estemos en un mapa donde haya varias escaleras hacia un mismo mapa. En estos casos queremos que cada escalera nos sitúe en un punto distinto de aparición, y es donde serán de utilidad las variables “escalera”. Estas guardan las posiciones desde las que podemos entrar a dicho mapa desde otros mapas, y como guardamos la posición desde la que entremos (mirar ilustración 3.102) en otra función **SetUltimaPos**, podemos comprobar desde qué escalera entramos con **LoadUltimaPos** y colocar al jugador en consonancia.

```

if (SistemaGuardado.LoadUltimaEscena() == "Galenholm_P-3")
{
    Destroy(cancion.gameObject);

    if (SistemaGuardado.LoadUltimaPos()[0] == escalera1[0] && SistemaGuardado.LoadUltimaPos()[1] == escalera1[1])
    {
        posicion.position = new Vector3(-49, 34, 0);
        Jugador.transform.position = posicion.position;
    }
    else if (SistemaGuardado.LoadUltimaPos()[0] == escalera2[0] && SistemaGuardado.LoadUltimaPos()[1] == escalera2[1])
    {
        posicion.position = new Vector3(escalera2[0], escalera2[1], 0);
        Jugador.transform.position = posicion.position;
    }
    else if (SistemaGuardado.LoadUltimaPos()[0] == escalera3[0] && SistemaGuardado.LoadUltimaPos()[1] == escalera3[1])
    {
        posicion.position = new Vector3(escalera3[0], escalera3[1], 0);
        Jugador.transform.position = posicion.position;
    }
    else if (SistemaGuardado.LoadUltimaPos()[0] == escalera4[0] && SistemaGuardado.LoadUltimaPos()[1] == escalera4[1])
    {
        posicion.position = new Vector3(escalera4[0], escalera4[1], 0);
        Jugador.transform.position = posicion.position;
    }
    else if (SistemaGuardado.LoadUltimaPos()[0] == escalera5[0] && SistemaGuardado.LoadUltimaPos()[1] == escalera5[1])
    {
        posicion.position = new Vector3(escalera5[0], escalera5[1], 0);
        Jugador.transform.position = posicion.position;
    }
    else
    {
        lugarAparicion();
    }
}

```

Ilustración 3.104: Comprobar escalera de entrada Galenholm P-2

También vemos en este ejemplo, que en caso de no entrar desde Galenholm_P-3 llamamos a una función llamada **lugarAparicion**. Esto lo hacemos en todas las escenas, y es una función que comprueba si hay datos guardados sobre la posición si venimos del menú de inicio, dícese, si estamos cargando la partida, en cuyo caso queremos que se nos coloque en la posición que estábamos al abandonar el juego.

```

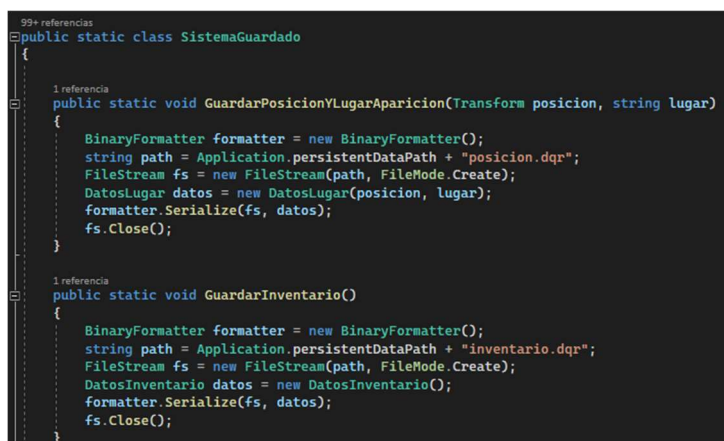
1 referencia
void lugarAparicion()
{
    if (SistemaGuardado.LoadUltimaEscena() == "Menu_Inicio")
    {
        float[] posicionGuardada = SistemaGuardado.LoadPosicion();
        if (posicionGuardada != null)
        {
            posicion.transform.position = new Vector3(posicionGuardada[0], posicionGuardada[1], posicionGuardada[2]);
            Jugador.transform.position = new Vector3(posicionGuardada[0], posicionGuardada[1], posicionGuardada[2]);
        }
    }
    else
    {
        Destroy(cancion);
    }
}

```

Ilustración 3.105: lugarAparicion

Esa es toda la información que implementarán la mayoría de los managers del juego, salvo excepciones como sería iniciar una nueva partida, o managers de escenas donde ocurran ciertos eventos que se verán en sprints posteriores.

Dado que hemos hablado del sistema de guardado vamos a explicar cómo funciona.

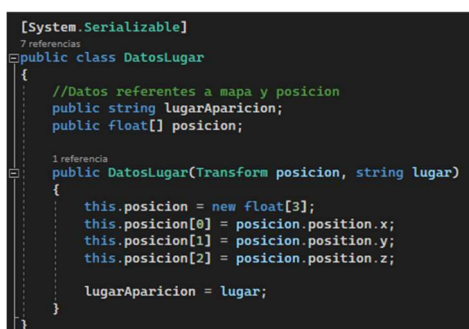


```
99+ referencias
public static class SistemaGuardado
{
    1 referencia
    public static void GuardarPosicionYLugarAparicion(Transform posicion, string lugar)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "posicion.dqr";
        FileStream fs = new FileStream(path, FileMode.Create);
        DatosLugar datos = new DatosLugar(posicion, lugar);
        formatter.Serialize(fs, datos);
        fs.Close();
    }

    1 referencia
    public static void GuardarInventario()
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "inventario.dqr";
        FileStream fs = new FileStream(path, FileMode.Create);
        DatosInventario datos = new DatosInventario();
        formatter.Serialize(fs, datos);
        fs.Close();
    }
}
```

Ilustración 3.106: SistemaGuardado

Se trata de una clase estática a la que todos los demás scripts pueden acceder. En nuestro proyecto hemos decidido crear distintos archivos de guardado dependiendo de la información que queramos guardar. En la imagen podemos ver dos ejemplos, uno guardando **DatosLugar**, que hace referencia a la escena y posición donde se ha guardado, y otro guardando **DatosInventario**, que guarda el contenido del inventario, que aún no hemos desarrollado, pero que se hará en sprints posteriores. Hay algo que recalcar, y es que, para poder guardar los datos es necesario marcarlos como Serializables poniendo antes de la definición de la clase **[System.Serializable]**.



```
[System.Serializable]
7 referencias
public class DatosLugar
{
    //Datos referentes a mapa y posicion
    public string lugarAparicion;
    public float[] posicion;

    1 referencia
    public DatosLugar(Transform posicion, string lugar)
    {
        this.posicion = new float[3];
        this.posicion[0] = posicion.position.x;
        this.posicion[1] = posicion.position.y;
        this.posicion[2] = posicion.position.z;

        lugarAparicion = lugar;
    }
}
```

Ilustración 3.107: DatosLugar

Lo que hacemos es abrir o crear un archivo (para cargar o guardar, respectivamente) en el lugar especificado, en nuestro caso hemos usado **Application.persistentDataPath**, que dependiendo del sistema operativo se guardará en uno u otro lugar. Es útil definir así el path para que no haya problemas si se decidiese hacer una versión para distintos sistemas.

3.3 Tercera Iteración – Creación del menú de inicio y de la UI

En este tercer sprint nos dedicamos a hacer toda la interfaz del juego y el menú de inicio, para el cual también necesitábamos de la interfaz y por lo que aún no lo habíamos hecho. En las subsecciones que seguirán veremos la escena menú inicial, su interfaz y el mánager que se encarga de manejarlo. En el resto, simplemente hablaremos de la interfaz, ya que la funcionalidad se implementó en otro sprint.

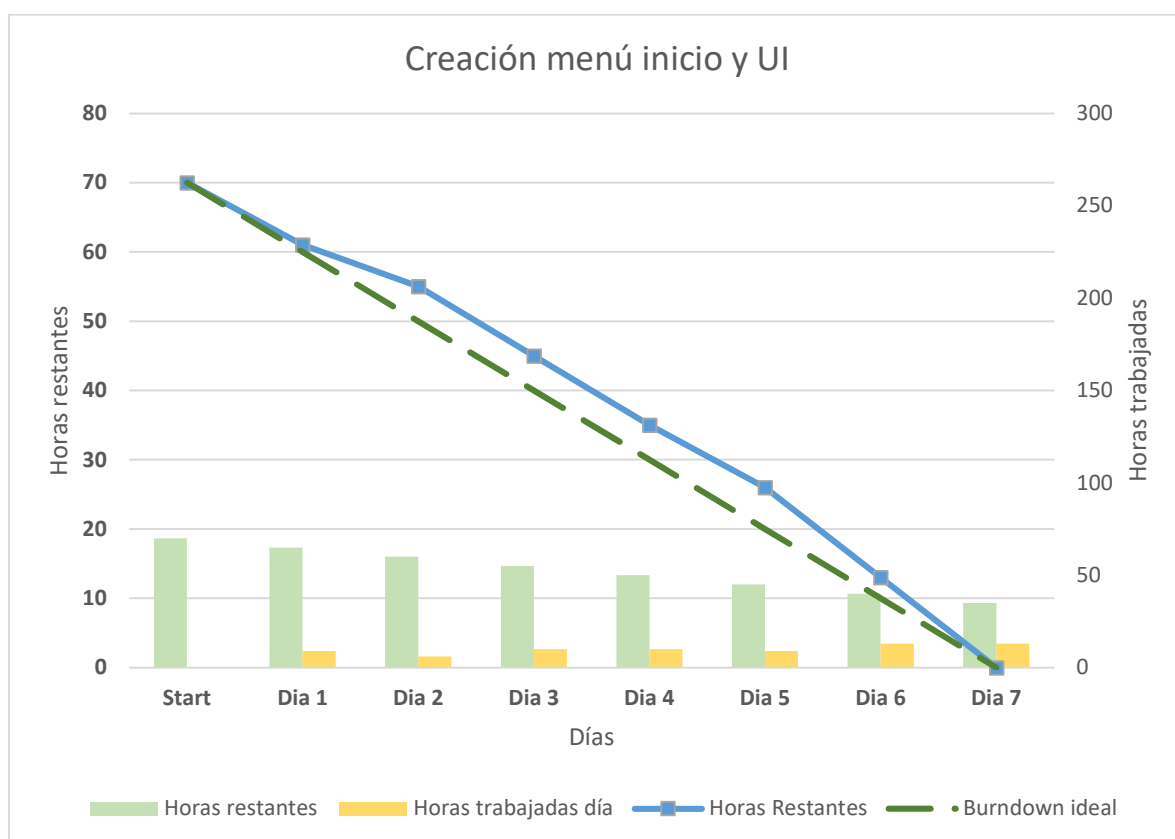


Ilustración 3.108: Diagrama Burndown - Creación del menú inicial y la UI

Le dedicamos un tiempo aproximado de 70 horas entre la creación del menú y su interfaz, y la creación del resto de interfaces. Parece mucho tiempo para solo hacer

la interfaz, pero hay que tener en cuenta que la mayoría de las imágenes utilizadas para hacer nuestra interfaz, si bien son inspiradas en la interfaz ya mencionada en apartado 2.2.3 Diseño interfaz, todas las hemos hecho nosotros usando la aplicación de Photoshop, a excepción del logo del juego que estaba incluido en los assets extraídos de la versión de Nintendo Switch.

3.3.1 Menu_Inicio

El menú de inicio consta de una cámara (la que viene por defecto al crear una escena, ya que no necesitaremos nada más) y un Canvas. Este Canvas será el que incluya dentro todos los elementos de la UI del menú.

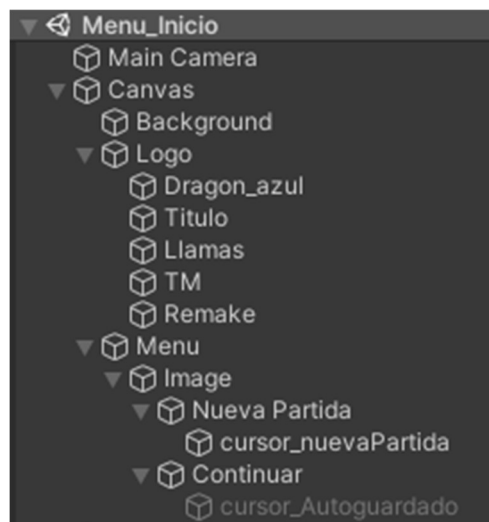


Ilustración 3.109: Elementos UI Menú Inicio

Como podemos ver, tenemos 3 elementos:

- **Background:** Simplemente se trata de una imagen por defecto en negro.



Ilustración 3.110: Background Menu_Inicio

- **Logo:** Es un conjunto de distintas imágenes colocadas para formar el logo del juego:

- Dragon_azul:



Ilustración 3.111: Dragon_azul

- Titulo:



Ilustración 3.112: Titulo

- Llamas:



Ilustración 3.113: Llamas

- TM:



Ilustración 3.114: TM

A parte de estas 4 imágenes también contiene un Text, en el que se indica que el remake está siendo realizado por mí.

- **Menú:** Esta parte es con la que interactuaremos para decidir si queremos crear una nueva partida. Está formada por una imagen con 2 textos dentro y cada uno con un cursor (otra imagen que representa un cursor como se ve aquí abajo) dentro.



Ilustración 3.115: Dragon-Quest-Menu_Background



Ilustración 3.116: Cursor

Cada uno de los textos son “Nueva Aventura” o bien “Continuar Partida”.

En el mánager nos encargaremos de hacer que todo se muestre de forma correcta, pero en el editor se ve de este modo:



Ilustración 3.117: Menu_Inicio editor

Como podemos ver, tenemos el cursor de continuar partida desmarcado por defecto, para siempre empezar con “Nueva Aventura” seleccionada. En el mánager nos encargamos de que al empezar el juego y entrar al menú solo se nos muestre el logo con su música de fondo, para la cual se ha utilizado la canción “Overture” que se puede escuchar aquí: (Roto, Youtube, s.f.); Y posteriormente al pulsar ‘Enter’ o bien ‘A’ con el mando, mostrar el menú de selección de modo.

```

Script de Unity (1 referencia de recurso) | 0 referencias
public class Gestor_Menu : MonoBehaviour
{
    public RectTransform continuar;
    private float timer_boton_A = 0f;
    private float timer_mover = 0f;
    public static string nombre = "";
    public GameObject Logo;
    private Gamepad gamepad;
    public Canvas menu;
    public GameObject cursor_NuevaPartida;
    public GameObject cursor_Autoguardado;
    //bool activo = true;
    Mensaje de Unity | 0 referencias
    void Start()
    {
        menu.gameObject.SetActive(false);
        gamepad = Gamepad.current;
    }
    Mensaje de Unity | 0 referencias
    void Update()
    {
        8 referencias
        void CambioEscena(string escena)
        {
            SistemaGuardado.SetUltimaEscena(SceneManager.GetActiveScene().name);
            SceneManager.LoadScene(escena);
        }
    }
}

```

Ilustración 3.118: Código Script "Gestor_Menu" 1

En el método **Start** desactivamos el menú para que no aparezca al inicio. También vemos que tenemos acceso ambos cursores, el de “Nueva Aventura” y el de “Continuar Partida” para que en caso de seleccionar una u otra opción se pueda habilitar la seleccionada y deshabilitar la otra.

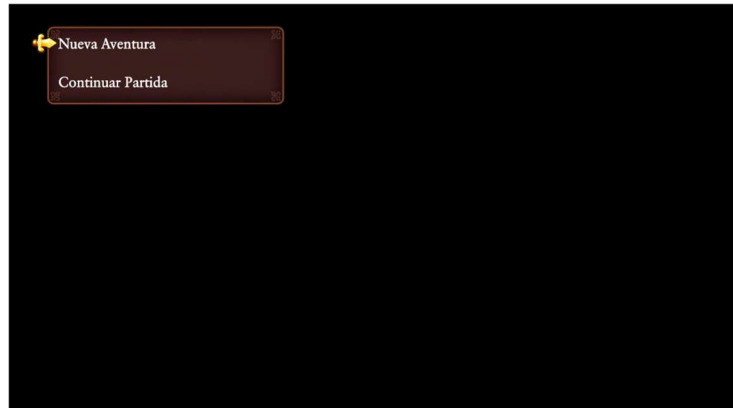
```

if (!menu.gameObject.activeInHierarchy)
{
    if (Gamepad.current != null)
    {
        gamepad = Gamepad.current;
        if (Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed())
        {
            menu.gameObject.SetActive(true);
            Logo.gameObject.SetActive(false);
        }
    }
    else
    {
        if (Input.GetKeyDown(KeyCode.Return))
        {
            menu.gameObject.SetActive(true);
            Logo.gameObject.SetActive(false);
        }
    }
}
}

```

Ilustración 3.119: Código Script "Gestor_Menu" 2

En esta primera parte del método **Update** (que solo se ejecutará mientras el menú esté desactivado) hacemos, ya sea con o sin mando, que al pulsar el botón/tecla correspondiente se deshabilite el logo y se nos muestro solo el menú.

*Ilustración 3.120: Menú al pulsar*

Una vez llegamos aquí, es decir, hemos activado el menú, en nuestro mánager se empezará a ejecutar otra parte del código:

```
if (menu.gameObject.activeInHierarchy)
{
    timer_boton_A += Time.deltaTime;

    if(SistemaGuardado.ExisteArchivo() is false)
    {
        cursor_Autoguardado.transform.parent.gameObject.SetActive(false);
        if (timer_boton_A > .5f)
        {
            if (Gamepad.current != null)
            {
                gamepad = Gamepad.current;
                if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()) && cursor_NuevaPartida.activeInHierarchy)
                {
                    CambioEscena("Castillo_P1");
                }
            }
            else
            {
                if (Input.GetKeyDown(KeyCode.Return) && cursor_NuevaPartida.activeInHierarchy)
                {
                    CambioEscena("Castillo_P1");
                }
            }
        }
    }
}
else
```

Ilustración 3.121: Código Script "Gestor_Menu" 3

En caso de que no exista el archivo de guardado se bloqueará la opción de Continuar Partida y solo se permitirá comenzar una Nueva Aventura. Si, por el contrario, sí que existe un archivo de guardado, tenemos la opción de elegir ya sea

Nueva Aventura o Continuar Partida. En esta parte usamos 2 timers distintos para movernos y para pulsar, de 0.1 y 0.5 segundos respectivamente. Debemos hacerlo así, ya que en caso contrario simplemente con pulsar un botón al principio se iniciaría una nueva partida, puesto que, en el caso del mando, este manda señales constantes de estar siendo pulsado, a diferencia del teclado que manda una única señal cuando es presionado.

```
if (timer_boton_A > .5f)
{
    if (Gamepad.current != null)
    {
        gamepad = Gamepad.current;
        if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()) && cursor_NuevaPartida.activeInHierarchy)
        {
            SistemaGuardado.NuevaPartida();
            CambioEscena("Castillo_P1");
        }
        else if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()) && cursor_Autoguardado.activeInHierarchy)
        {
            string lugarAparicion = SistemaGuardado.LoadLugarAparición();
            if (lugarAparicion != null)
                CambioEscena(lugarAparicion);
            else
                CambioEscena("Castillo_P1");
        }
    }
    else
    {
        if (Input.GetKeyDown(KeyCode.Return) && cursor_NuevaPartida.activeInHierarchy)
        {
            SistemaGuardado.NuevaPartida();
            CambioEscena("Castillo_P1");
        }
        else if (Input.GetKeyDown(KeyCode.Return) && cursor_Autoguardado.activeInHierarchy)
        {
            string lugarAparicion = SistemaGuardado.LoadLugarAparición();
            if (lugarAparicion != null)
                CambioEscena(lugarAparicion);
            else
                CambioEscena("Castillo_P1");
        }
    }
}
timer_mover += Time.deltaTime;
```

Ilustración 3.122: Código del Script "Gestor_Menu" 4

Aquí se controla la parte de pulsar botones, y como vemos, en caso de pulsar en continuar partida cargamos el mapa en el que nos encontrábamos desde los archivos de guardado. Nunca deberían de ser nulos, pero por si hubiese algún tipo de error en la lectura, en caso de ser nulos, cargaremos desde un mapa por defecto.

```

if (timer_mover > 0.1f)
{
    if (Gamepad.current != null)
    {
        gamepad = Gamepad.current;
        if (Input.GetKeyDown(KeyCode.W) || gamepad.dpad.up.IsPressed())
        {
            if (cursor_Autoguardado.activeInHierarchy)
            {
                cursor_NuevaPartida.SetActive(true);
                cursor_Autoguardado.SetActive(false);
                timer_mover = 0;
            }
        }
        else if (Input.GetKeyDown(KeyCode.S) || gamepad.dpad.down.IsPressed())
        {
            if (cursor_NuevaPartida.activeInHierarchy)
            {
                cursor_NuevaPartida.SetActive(false);
                cursor_Autoguardado.SetActive(true);
                timer_mover = 0;
            }
        }
    }
    else
    {
        if (Input.GetKeyDown(KeyCode.W))...
        else if (Input.GetKeyDown(KeyCode.S))...
    }
}

```

Ilustración 3.123: Código del Script "Gestor_Menu" 5

Y en esta otra imagen vemos la parte de movernos entre las opciones de Nueva Aventura y de Continuar Partida.

3.3.2 Interfaz para diálogos y mánager de diálogos

Nuestra interfaz para diálogos, al igual que menú de inicio está formado por un Canvas con distintos objetos dentro. En este caso dentro de nuestra interfaz para diálogos nos encontramos con 3 tipos de diálogos que son:

- **OpcionDialogo:** Se utilizará cuando en un diálogo tengamos que elegir entre una u otra opción. Está compuesta de una imagen que, al igual que en el menú, tendrá dentro 2 textos con sus cursores.



Ilustración 3.124: Dragon-Quest-Dialogue_Opcion

- **CuadroDialogo:** Será la que utilizarán los NPCs para hablar con nosotros una vez los implementemos. Es una imagen con dos Textos dentro, uno será el nombre del NPC y otro lo que tenga que decirnos.

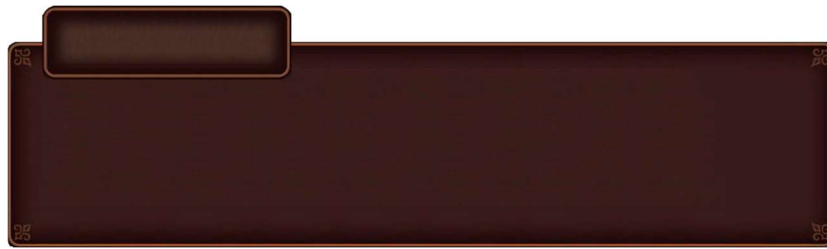


Ilustración 3.125: Dragon-Quest-Dialogue_Background

- **Tienda:** Será la que utilizaremos para comprar objetos. También la manejarán los NPCs que sean mercaderes. La tienda tiene bastantes objetos dentro de ella que los NPCs deberán de manejar, son los siguientes:

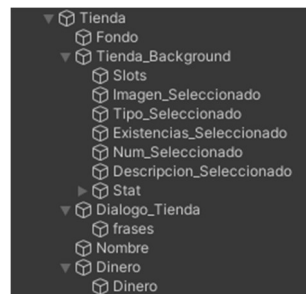


Ilustración 3.126: Elementos Tienda



Ilustración 3.127: Tienda

A la izquierda de la imagen en blanco y debajo de tienda hay un Objeto llamado Slots, que lo que hará será encargarse de almacenar y poner de forma ordenada, haciendo uso de **Vertical Layout Group**, todos los objetos que haya en la tienda. Aunque solo pintando un máximo de 5 de ellos a la vez. Para esto creamos un prefab con los datos que contendrán dichos objetos al que hemos llamado "Item".

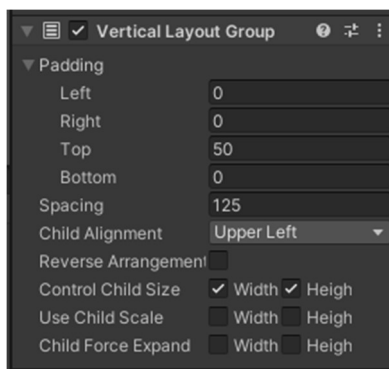


Ilustración 3.128: Vertical Layout – Slots



Ilustración 3.129: Prefab Item

Estos prefabs contendrán un icono que representará al objeto, un nombre, un precio, así como tres variables, seleccionado, cursor, equipado, que se encargarán de mostrar si está seleccionado (ya sea en la tienda o en el inventario, cuya interfaz es muy similar) o bien si, en caso de ser un objeto equipable, se encuentra equipado.

En cuanto al mánager de diálogos, este será el encargado de escribir los diferentes mensajes que ya sea la tienda, ya sean otros NPCs, se nos transmiten. Primeramente, creamos una clase **Dialogo** que tendrá la información de quién nos transmite un mensaje y el contenido de este.

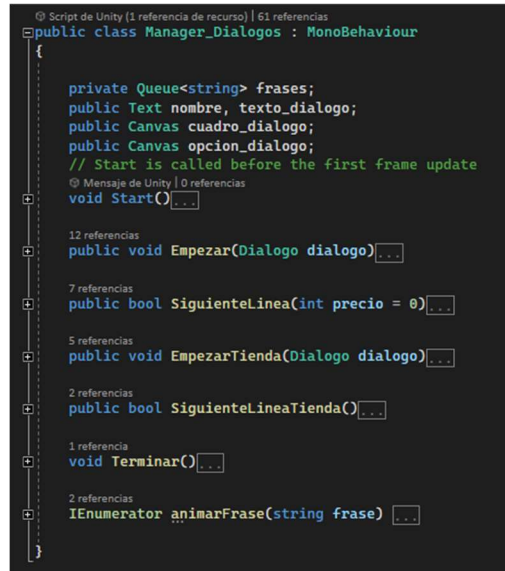
```
[System.Serializable]
39 referencias
public class Dialogo
{
    public string nombre;

    [TextArea(1,5)]
    public string[] frases;
}
```

Ilustración 3.130: Clase Dialogo

Como vemos es una clase muy sencilla, pero servirá para nuestro propósito.

En el mánager tenemos las siguientes variables y métodos:



```
Script de Unity (1 referencia de recurso) | 61 referencias
public class Manager_Dialogos : MonoBehaviour
{
    private Queue<string> frases;
    public Text nombre, texto_dialogo;
    public Canvas cuadro_dialogo;
    public Canvas opcion_dialogo;
    // Start is called before the first frame update
    [SerializeField]
    void Start()
    {
    }

    12 referencias
    public void Empezar(Dialogo dialogo)
    {
    }

    7 referencias
    public bool SiguienteLinea(int precio = 0)
    {
    }

    5 referencias
    public void EmpezarTienda(Dialogo dialogo)
    {
    }

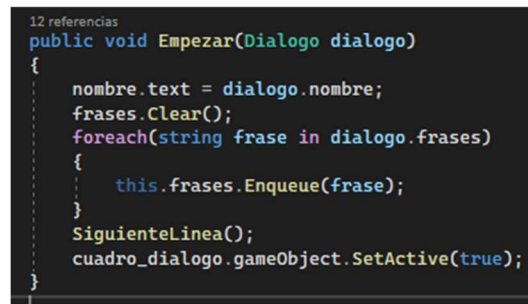
    2 referencias
    public bool SiguienteLineaTienda()
    {
    }

    1 referencia
    void Terminar()
    {
    }

    2 referencias
    IEnumerator animarFrase(string frase)
    {
    }
}
```

Ilustración 3.131: Clase Manager_Dialogos

En el método **Empezar**, lo que hacemos es, a partir del diálogo introducido, asignar el nombre y el mensaje que se nos da.



```
12 referencias
public void Empezar(Dialogo dialogo)
{
    nombre.text = dialogo.nombre;
    frases.Clear();
    foreach(string frase in dialogo.frases)
    {
        this.frases.Enqueue(frase);
    }
    SiguienteLinea();
    cuadro_dialogo.gameObject.SetActive(true);
}
```

Ilustración 3.132: Código del Script "Manager_Dialogos" 1

A su vez dentro del método Empezar llamamos a **SiguienteLinea**, lo que hace que se nos muestre la primera línea de todas las que contenga el mensaje.


```

7 referencias
public bool SiguienteLinea(int precio = 0)
{
    if(frases.Count == 0)
    {
        Terminar();
        return false;
    }

    string frase = frases.Dequeue();
    StopAllCoroutines();
    StartCoroutine(animarFrase(frase));
    return true;
}

```

Ilustración 3.133: Código del Script "Manager_Dialogos" 2

En el método **SiguienteLinea** comprobamos, y si ya no tenemos más frases devolvemos false. Esto sirve para que los NPCs sepan cuando han terminado de conversar con nosotros. El método **Terminar** lo único que hace es poner inactivo el cuadro de diálogo (quitarlo de la pantalla). Si aún quedan líneas, se llamará a la corrutina **animarFrase**, que se dedicará a presentar nuestra frase letra por letra, como es habitual en videojuegos.

```

2 referencias
IEnumerator animarFrase(string frase)
{
    texto_dialogo.text = "";
    foreach(char letra in frase.ToCharArray())
    {
        texto_dialogo.text += letra;
        yield return new WaitForSeconds(0.01f);
    }
}

```

Ilustración 3.134; Código del Script "Manager_Dialogos" 3

3.3.3 Interfaz Menú de Pausa y Manager Menú de Pausa

Nuestro menú de Pausa consta de 5 apartados diferenciados:

- **menú_inicial**: Será el que se muestre cuando pulsemos esc. o bien start en el mando, y desde el cual podremos elegir si queremos acceder al inventario, a las estadísticas, a las magias o miscelánea.



Ilustración 3.135: Menu_inicial del Menú de Pausa



Ilustración 3.136: Dragon-Quest-Pause-Menu

- **Inventario:** Es la interfaz del inventario de nuestro juego, es muy parecida a la de la tienda. Cuando tengamos el inventario y demás implementado nos permitirá usar objetos, equiparlos, desequiparlos o tirarlos.

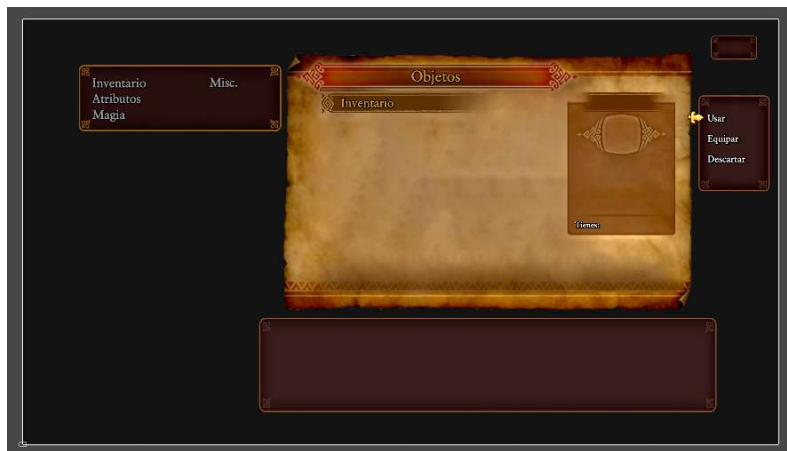


Ilustración 3.137: Interfaz Inventario

Para hacer esta interfaz se han utilizado las siguientes imágenes creadas en Photoshop:



Ilustración 3.138: Dragon-Quest-Eleccion-Inventario



Ilustración 3.139: Dragon-Quest-Inventario-Dinero



Ilustración 3.140: Dragon-Quest-Dialogo-Inventario



Ilustración 3.141: Dragon-Quest-Inventario

- **Estadísticas:** En esta parte se nos mostrarán las estadísticas de nuestro jugador una vez las implementemos.



Ilustración 3.142: Estadísticas Jugador

Todos los números que aparecen son números por defecto. Aunque el nombre y el título (parte superior derecha) son así durante todo el juego. Para hacer este menú hemos utilizado las siguientes imágenes:

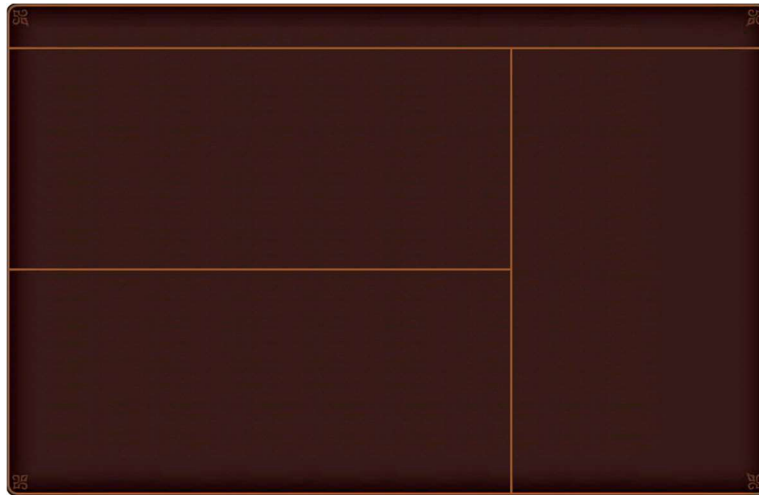


Ilustración 3.143: Dragon-Quest-Estadísticas



Ilustración 3.144: Main_Character

Esta imagen del personaje principal es sino un recorte con algunos retoques de la imagen que podemos encontrar aquí: (Enix, s.f.)

Además de estas, también hemos utilizamos los iconos de daga, espada, espadón, lanza, hacha, armadura, casco, escudo, anillo y collar extraídos del juego DQ XI, y que cambiarán dependiendo de lo que llevemos equipado:



Ilustración 3.145: Icono Espada



Ilustración 3.146: Icono Espadón



Ilustración 3.147: Icono Hacha



Ilustración 3.148: Icono Lanza



Ilustración 3.149: Icono Daga



Ilustración 3.150: Icono Casco



Ilustración 3.151: Icono Armadura



Ilustración 3.152: Icono Escudo



Ilustración 3.153: Icono Anillo



Ilustración 3.154: Icono Collar

- **Magias:** Se nos muestran los distintos conjuros que podemos utilizar y nos dejará utilizar aquellos que se puedan usar fuera del combate, como las curaciones.



Ilustración 3.155: Menú Magias

- **Miscelánea:** Este apartado nos permitirá guardar la partida y salir del juego, es el más sencillo de todos.



Ilustración 3.156: Menú Miscelánea

En cuanto al mánager que se encargará de controlar el menú de pausa, es un script que portará el jugador.

```

// Script de Unity (1 referencia de recurso) | 7 referencias
public class Menu_Pausa : MonoBehaviour
{
    private float timer_boton = 0f;
    private float tiempo_espera_boton = 0.5f;
    private float timer_mover = 0f;
    private float tiempo_espera_mover = 0.3f;
    public static bool bloqueado_por_script = false;
    private bool inventario_cargado = false;
    public GameObject prefabObjeto;
    private int maxDibujado = 4;
    private int minDibujado = 0;
    public GameObject mostradorObjetos;
    private GameObject menu_pausa;
    private int seleccionado = 0;
    private int seleccionado_magias = 0;
    private int seleccionado_inventario = 0;
    private int seleccionado_inventario_accion = 0;
    private Gamepad gamepad;
    public Sprite transparente;
    public Sprite daga;
    public Sprite espada;
    public Sprite espadon;
    public Sprite hacha;
    public Sprite lanza;
    public Sprite collar;
    public Sprite anillo;
    private Slider HP;
    private Slider MP;
    private Text max_hp, max_mp, ataque, defensa, agilidad, magia, def_magica, nivel, exp, exp_proximo_lvl, hp_actual, mp_actual, casco, armadura, arma, accesorio, escudo;
}

```

Ilustración 3.157: Código del Script "Menu_Pausa" 1

Como se puede ver tenemos una cantidad de variables bastante grande, desde timers y sprites para las armas, armaduras, etc. Hasta variables de texto que recogerán los valores de las estadísticas del jugador (aunque aún no se han implementado).

```

// Menú de pausa (1 referencia)
void Start()
{
    gamepad = Gamepad.current;
    menu_pausa = GameObject.Find("MenuPausa");
    mostradorObjetos = menu_pausa.transform.Find("Canvas").transform.Find("Inventario").transform.Find("Inventario_Background").transform.Find("Slots").gameObject;
    HP = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Barra_HP").GetComponent<Slider>();
    MP = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Barra_MP").GetComponent<Slider>();
    max_hp = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Max_HP").transform.Find("MP").GetComponent<Text>();
    hp_actual = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("HP_Actual").GetComponent<Text>();
    max_mp = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Max_MP").transform.Find("MP").GetComponent<Text>();
    mp_actual = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("MP_Actual").GetComponent<Text>();
    ataque = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Ataque").transform.Find("Atq").GetComponent<Text>();
    defensa = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Defensa").transform.Find("Def").GetComponent<Text>();
    agilidad = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Agilidad").transform.Find("SPD").GetComponent<Text>();
    magia = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Poder_Magico").transform.Find("MAG").GetComponent<Text>();
    def_magica = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Defensa_Magica").transform.Find("DEF_MAG").GetComponent<Text>();
    nivel = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Nivel").GetComponent<Text>();
    exp = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Experiencia_Actual").transform.Find("Experiencia_Actual_Num").GetComponent<Text>();
    exp_proximo_lvl = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Experiencia_Restante").transform.Find("Experiencia_Restante_Num").GetComponent<Text>();
    casco = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Casco").transform.Find("Text").GetComponent<Text>();
    arma = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Arma").transform.Find("Text").GetComponent<Text>();
    armadura = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Armadura").transform.Find("Text").GetComponent<Text>();
    accesorio = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Accesorio").transform.Find("Text").GetComponent<Text>();
    escudo = menu_pausa.transform.Find("Canvas").transform.Find("Atributos").transform.Find("Atributos_Background").transform.Find("Escudo").transform.Find("Text").GetComponent<Text>();
}

```

Ilustración 3.158: Código del Script "Menu_Pausa" 2

El método **Start** está dedicado a asignar a cada variable su GameObject o su componente en la escena. Normalmente nos basta con poner públicas las variables y asignarlas desde el editor, de modo que nos ahorramos todo este código, pero en este caso no es posible, ya que toda la interfaz pertenece a la escena Menu_Inicio, solo que tienen el script DontDestroyOnLoad para que no se destruyan al cambiar de escena.

```

void Update()
{
    timer_boton += Time.deltaTime;

    if (SceneManager.GetActiveScene().name != "Menu_Inicio" && (Movimiento.bloqueado != true || bloqueado_por_script == true))
    {
        if (menu_pausa.transform.Find("Canvas").transform.Find("Menu_Inicial").gameObject.activeInHierarchy)
        {
            else
            {
                if (Gamepad.current != null)
                {
                    gamepad = Gamepad.current;
                    if ((Input.GetKeyDown(KeyCode.Escape) || gamepad.startButton.IsPressed()) && timer_boton > tiempo_espera_boton)
                    {
                        menu_pausa.transform.Find("Canvas").transform.Find("Fondo").gameObject.SetActive(true);
                        menu_pausa.transform.Find("Canvas").transform.Find("Menu_Inicial").gameObject.SetActive(true);
                        bloqueado_por_script = true;
                        Movimiento.bloqueado = true;
                        timer_boton = 0;
                    }
                }
            }
        }
        else
        {
            if (Input.GetKeyDown(KeyCode.Escape) && timer_boton > tiempo_espera_boton)
            {
                menu_pausa.transform.Find("Canvas").transform.Find("Fondo").gameObject.SetActive(true);
                menu_pausa.transform.Find("Canvas").transform.Find("Menu_Inicial").gameObject.SetActive(true);
                bloqueado_por_script = true;
                Movimiento.bloqueado = true;
                timer_boton = 0;
            }
        }
    }
}

```

Ilustración 3.159: Código del Script "Menu_Pausa" 2

En el método **Update** de nuestro mánager lo primero que hacemos es comprobar que el movimiento no esté bloqueado y que no estemos en el menú inicial, condiciones que evitan que lo usemos en esas situaciones. Como ya se ha mencionado a lo largo de esta documentación, siempre que abramos el menú, hablemos con un NPC, etc. Se bloqueará el movimiento. También para eso sirve la variable "bloqueado_por_script", que significa que ha sido bloqueado por el mismo script, es decir, sí podrá ejecutarse siempre y cuando el movimiento lo haya bloqueado él mismo. Luego, mientras el menú no esté activo comprobamos si se pulsa esc. o Start, y en el momento en que se haga, bloqueamos el movimiento y activamos el menú, dándonos acceso a las diferentes posibilidades de este. Hay mucho código que garantiza la correcta visualización de la interfaz, pero nos centraremos en la parte más importante.

```

if (menu_pausa.transform.Find("Canvas").transform.Find("Inventario").gameObject.activeInHierarchy)
else if (menu_pausa.transform.Find("Canvas").transform.Find("Atributos").gameObject.activeInHierarchy)
else if (menu_pausa.transform.Find("Canvas").transform.Find("Magia").gameObject.activeInHierarchy)
else if (menu_pausa.transform.Find("Canvas").transform.Find("Misc").gameObject.activeInHierarchy)
else
{
    timer_mover += Time.deltaTime;
    if (Gamepad.current != null)
    {
        gamepad = Gamepad.current;
        if ((Input.GetKeyDown(KeyCode.W) || gamepad.dpad.up.IsPressed()) && timer_mover > tiempo_espera_mover)
        else if ((Input.GetKeyDown(KeyCode.S) || gamepad.dpad.down.IsPressed()) && timer_mover > tiempo_espera_mover)
        else if ((Input.GetKeyDown(KeyCode.D) || gamepad.dpad.right.IsPressed()) && timer_mover > tiempo_espera_mover)
        else if ((Input.GetKeyDown(KeyCode.A) || gamepad.dpad.left.IsPressed()) && timer_mover > tiempo_espera_mover)
        else if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()) && timer_boton > tiempo_espera_boton)
        {
            switch (seleccionado)
            {
                case 0:
                    menu_pausa.transform.Find("Canvas").transform.Find("Inventario").gameObject.SetActive(true);
                    break;
                case 1:
                    menu_pausa.transform.Find("Canvas").transform.Find("Atributos").gameObject.SetActive(true);
                    break;
                case 2:
                    menu_pausa.transform.Find("Canvas").transform.Find("Magia").gameObject.SetActive(true);
                    break;
                case 3:
                    menu_pausa.transform.Find("Canvas").transform.Find("Misc").gameObject.SetActive(true);
                    break;
            }
            timer_boton = 0;
        }
        else if ((Input.GetKeyDown(KeyCode.Escape) || gamepad.buttonEast.IsPressed()) && timer_boton > tiempo_espera_boton)
    }
}

```

Ilustración 3.160: Código del Script "Menu_Pausa" 3

Cuando elegimos una de las opciones de menú activamos la parte correspondiente, ya sea inventario, atributos, magia o misc. En este sprint solo hemos implementado la parte de miscelánea, que como ya vimos solo consiste en la opción para guardar y salir del juego. El resto de las opciones se implementarán más adelante cuando existan los objetos (para el inventario) y las estadísticas del jugador (para los atributos y las magias), por ahora implementamos su interfaz.

```
else if (menu_pausa.transform.Find("Canvas").transform.Find("Misc").gameObject.activeInHierarchy)
{
    if (Gamepad.current != null)
    {
        gamepad = Gamepad.current;
        if ((Input.GetKeyDown(KeyCode.Escape) || gamepad.buttonEast.IsPressed()) && timer_boton > tiempo_espera_boton)
        {
            menu_pausa.transform.Find("Canvas").transform.Find("Misc").gameObject.SetActive(false);
            timer_boton = 0;
        }
        if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()) && timer_boton > tiempo_espera_boton)
        {
            Application.Quit();
        }
    }
    else
    {
        if (Input.GetKeyDown(KeyCode.Escape) && timer_boton > tiempo_espera_boton)
        {
            menu_pausa.transform.Find("Canvas").transform.Find("Misc").gameObject.SetActive(false);
            timer_boton = 0;
        }
        if (Input.GetKeyDown(KeyCode.Return) && timer_boton > tiempo_espera_boton)
        {
            Application.Quit();
        }
    }
}
else...
```

Ilustración 3.161: Código del Script "Menu_Pausa" 4

Como dijimos, la parte de miscelánea es la más sencilla. Solo cuenta con una única opción, por ello solo tenemos que registrar si se pulsa el botón de atrás o el de aceptar.

3.3.4 Interfaz de combate

Esta es la única interfaz que nos queda por explicar. En este caso no solo está formada por objetos pertenecientes a UI (tipo de objeto), sino que también tiene un AudioSource con la canción "Battle" que se puede escuchar aquí: (Roto, Youtube , s.f.), y que será utilizado por el mánager de batalla que NO implementaremos aquí.

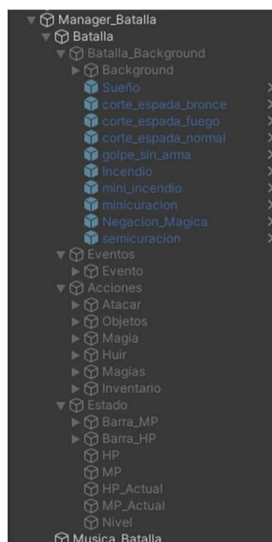


Ilustración 3.162: Objetos de Manager_Batalla

Tenemos muchos objetos diferentes, empezaremos explicando desde lo más sencillo, cuando hagamos el mánager se explicará en mayor profundidad:

- **Estado**, y todo lo que tiene dentro sirven para mostrar la situación actual del jugador en cuanto a vida, mana, nivel, etc. Se refiere.



Ilustración 3.163: Estado - Manager_Batalla

- **Acciones**, hace referencia a todas las acciones diferentes que podremos hacer en la pelea, pueden ser Atacar, usar objetos, usar magia o huir.



Ilustración 3.164: Acciones Batalla

- **Eventos**, es el cuadro de diálogo donde se mostrarán los mensajes de batalla, por ejemplo, “El enemigo ha recibido 20 de daño”.



Ilustración 3.165: Diálogo batalla

- **Batalla_Background**, engloba tanto al escenario de batalla como a los prefabs de animaciones de todos los tipos de ataques y curaciones que se pueden hacer en el juego y a los cuales se llamará cuando sea necesario en el combate. El escenario de combate es el siguiente:

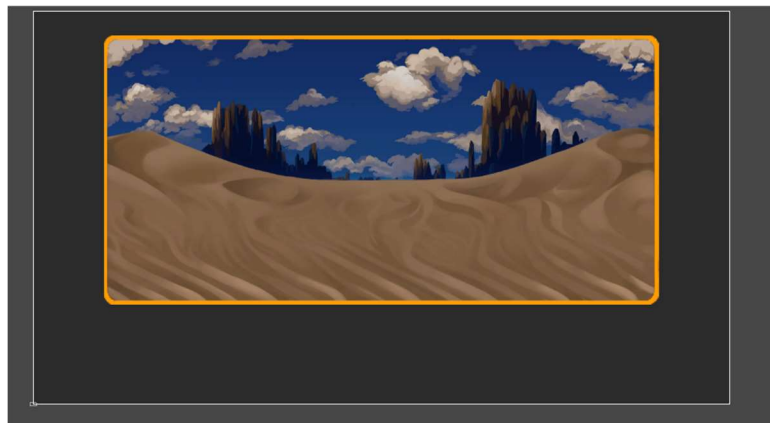


Ilustración 3.166: Escenario Combate

La imagen es la que tenemos por defecto, pero como ya explicaremos, cambiará según el lugar en que nos encontremos.

También hemos hablado de todas las animaciones de tipos de ataques y las mostraremos a continuación, aunque no entraremos en detalle de como las hemos hecho, ya que simplemente hay que usar el animador de Unity e ir cambiando las imágenes cada 'x' fotogramas.

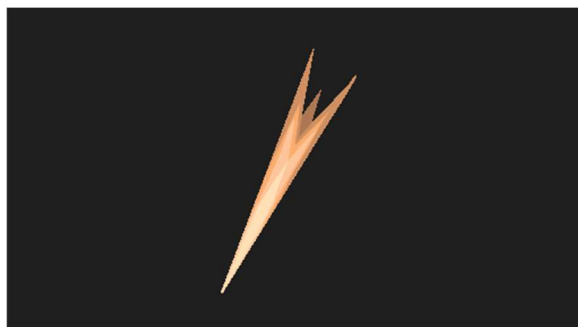


Ilustración 3.167: Animación corte_espada_bronce

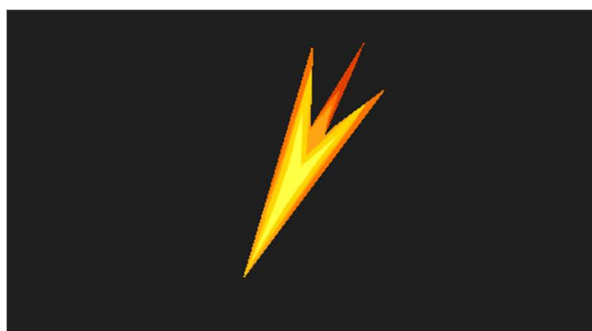


Ilustración 3.168: Animación corte_espada_fuego



Ilustración 3.169: Animación corte_espada_normal

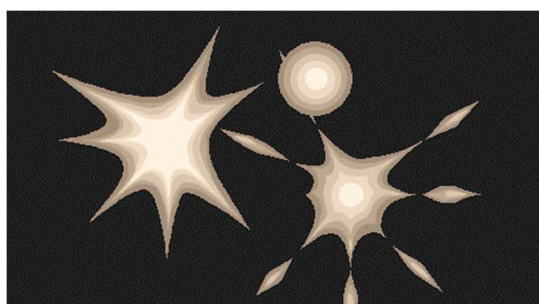


Ilustración 3.170: Animación golpe_sin_arma



Ilustración 3.171: Animación Incendio



Ilustración 3.172: Animación Mini-Incendio

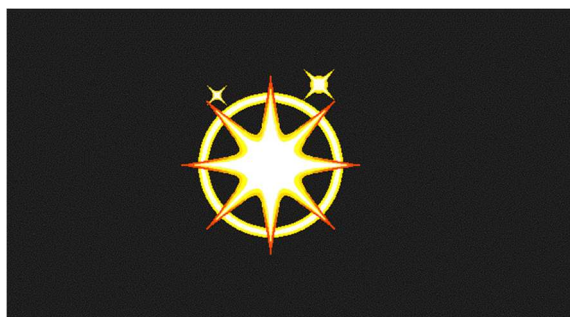


Ilustración 3.173: Animación Mini-Curación

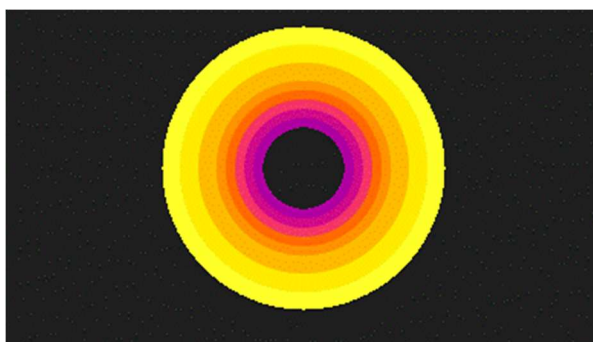


Ilustración 3.174: Animación Negación Mágica

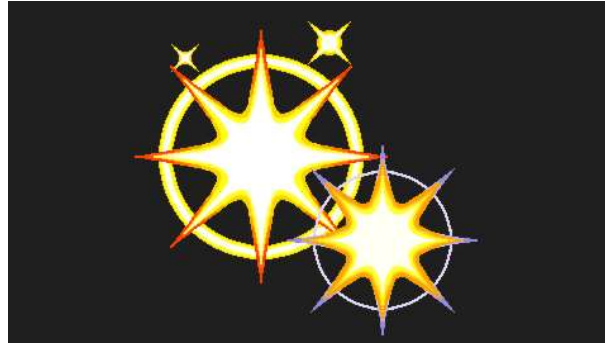


Ilustración 3.175: Animación Semi-Curación



Ilustración 3.176: Animación Sueño

Esto es todo lo que tiene nuestra interfaz de combate. A parte de esto, cuando realicemos la implementación del mánager veremos que hay alguna animación más, así como, se instanciarán los enemigos dentro de la zona de batalla y se interactuará con ellos allí, pero esa parte pertenece a otro sprint.

3.4 Cuarta Iteración – Creación de NPCs, Manager de Batalla, manejo del menú de pausa, etc.

En esta cuarta iteración creamos todos los NPCs que usaremos en el juego, así como los distintos scripts para controlarlos, todo el sistema de combate incluyendo las estadísticas, magias y objetos (estos últimos son necesario además de para el combate, para la tienda, y para NPCs que entreguen objetos) y el resto de las funcionalidades que faltaban dado que estos no existían.

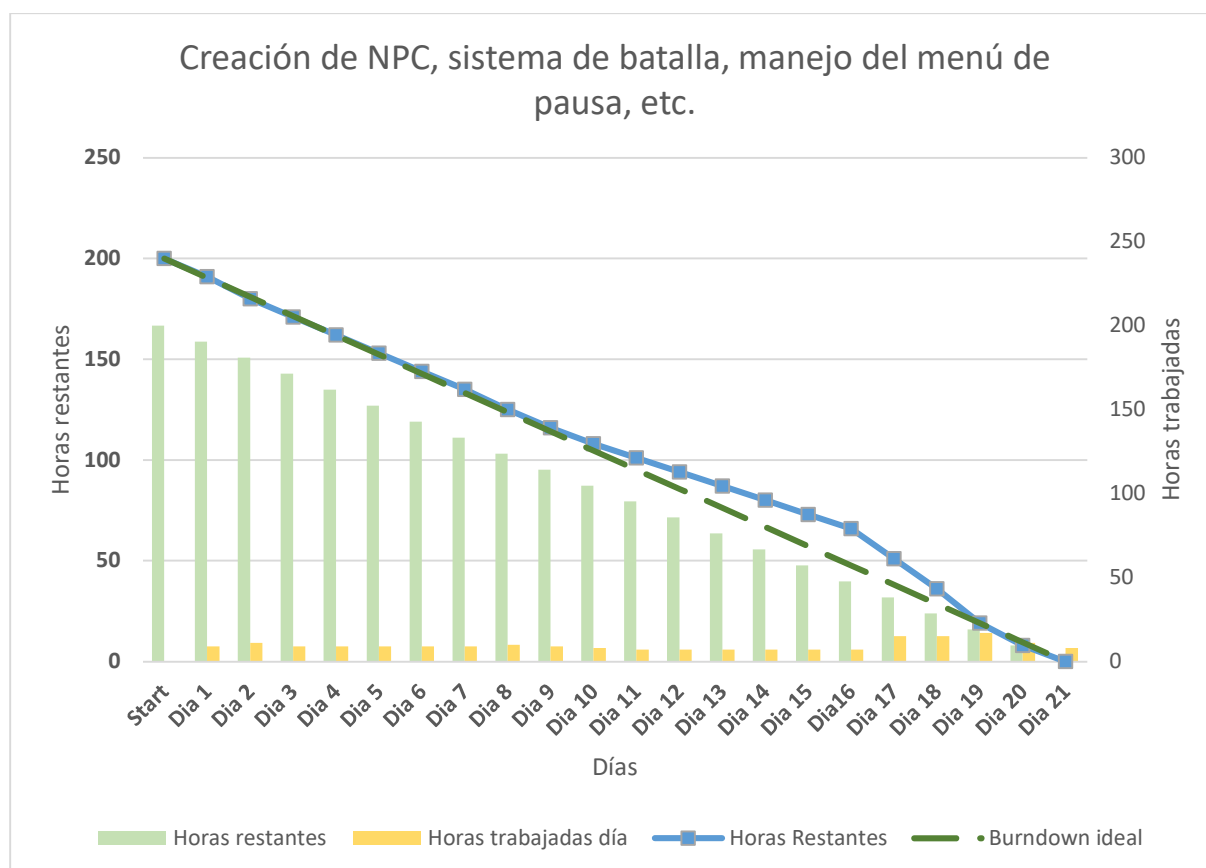


Ilustración 3.177: Diagrama Burndown - Creación de NPCs, Manager de batalla, manejo del menú de pausa, etc.

Esta iteración al ser que más funcionalidad implementa es también a la que más tiempo le hemos dedicado. Se le han asignado 3 semanas de sprint y un aproximado de 200 horas de trabajo.

3.4.1 Creación de NPCs

Los NPCs son bastante similares al jugador que ya vimos anteriormente. Están formados por un Sprite, una posición y un Grid con un Tilemap cuya capa es Collider, para evitar que nuestro jugador pueda entrar a la misma casilla de un NPC.

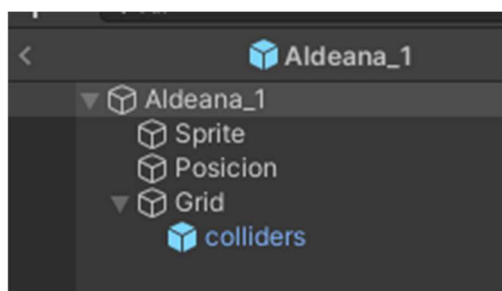
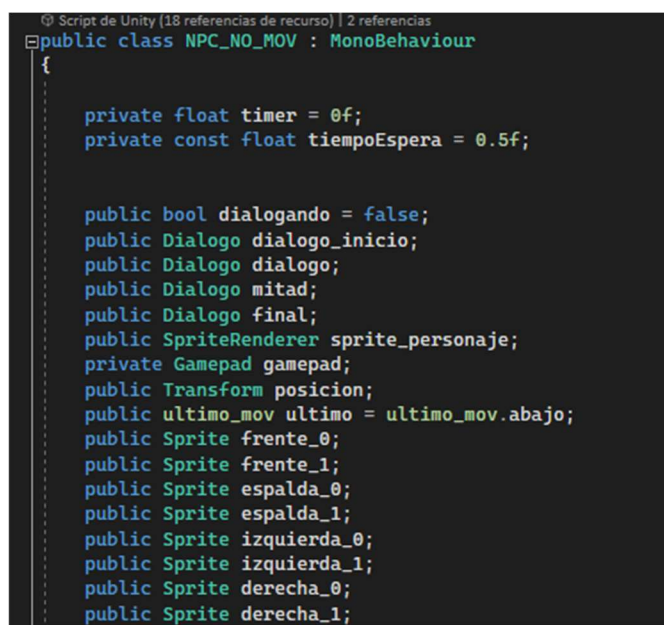


Ilustración 3.178: Composición NPC

Primero vamos a explicar los diferentes Scripts que pueden llevar nuestros NPCs para posteriormente mostrar todos los NPCs distintos que hemos utilizado.

3.4.1.1 NPC_NO_MOV y NPC_NO_MOV_2

Este tipo de NPC será el que se encuentra en un lugar quieto esperando a que vayas a hablar con él.

A screenshot of the Unity Inspector window showing the code for the NPC_NO_MOV script. The script is a C# class that inherits from MonoBehaviour. It contains several private and public variables. The private variables are timer (float, 0f) and tiempoEspera (const float, 0.5f). The public variables are dialogando (bool, false), dialogo_inicio (Dialogo), dialogo (Dialogo), mitad (Dialogo), final (Dialogo), sprite_personaje (SpriteRenderer), gamepad (Gamepad), posicion (Transform), ultimo_mov (ultimo_mov.abajo), frente_0 (Sprite), frente_1 (Sprite), espalda_0 (Sprite), espalda_1 (Sprite), izquierda_0 (Sprite), izquierda_1 (Sprite), derecha_0 (Sprite), and derecha_1 (Sprite).

```
Script de Unity (18 referencias de recurso) | 2 referencias
public class NPC_NO_MOV : MonoBehaviour
{
    private float timer = 0f;
    private const float tiempoEspera = 0.5f;

    public bool dialogando = false;
    public Dialogo dialogo_inicio;
    public Dialogo dialogo;
    public Dialogo mitad;
    public Dialogo final;
    public SpriteRenderer sprite_personaje;
    private Gamepad gamepad;
    public Transform posicion;
    public ultimo_mov ultimo = ultimo_mov.abajo;
    public Sprite frente_0;
    public Sprite frente_1;
    public Sprite espalda_0;
    public Sprite espalda_1;
    public Sprite izquierda_0;
    public Sprite izquierda_1;
    public Sprite derecha_0;
    public Sprite derecha_1;
}
```

Ilustración 3.179: Código del Script "NPC_NO_NOV" 1

Tenemos las variables que se ven en la imagen. Como vemos el NPC tiene una variable que al igual que el menú de pausa comprueba si está hablando con nosotros o no, con comportamientos diferenciados. Además de esto, tenemos la posición, el último movimiento, sprites para todas las posiciones, etc. Al igual que lo tenía nuestro jugador. Pero más importante, tiene Diálogos; Los diálogos se utilizarán cuando vayamos a hablar con el personaje y serán los que se pasen al mánager de diálogos que vimos en la sección anterior. Tiene diferentes Diálogos para adecuarse a la parte del juego en la que nos encontremos. En el método **Start** simplemente iniciamos la animación del NPC, así que pasaremos directamente a ver el método **Update** que es donde ocurre todo.

```

void Update()
{
    if (Menu_Pausa.bloqueado_por_script == false)
    {
        if (!dialogando)
        {
            timer += Time.deltaTime;
            if (timer > tiempo)
            {
                if (Gamepad.current != null)
                {
                    gamepad = Gamepad.current;
                    if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()))
                    {
                        && !FindObjectOfType<Manager_Dialogos>().transform.Find("Cuadro_Dialogo").gameObject.activeInHierarchy
                        && ((Movimiento.posicion.position == (posicion.position + new Vector3(1, 0, 0)) && Movimiento.ultimo == ultimo_mov.izquierda)
                        || (Movimiento.posicion.position == (posicion.position + new Vector3(-1, 0, 0)) && Movimiento.ultimo == ultimo_mov.derecha)
                        || (Movimiento.posicion.position == (posicion.position + new Vector3(0, -1, 0)) && Movimiento.ultimo == ultimo_mov.arriba)
                        || (Movimiento.posicion.position == (posicion.position + new Vector3(0, 1, 0)) && Movimiento.ultimo == ultimo_mov.abajo)
                    ))
                    {
                        // ...
                    }
                }
                else
                {
                    if (Input.GetKeyDown(KeyCode.Return) && !FindObjectOfType<Manager_Dialogos>().transform.Find("Cuadro_Dialogo").gameObject.activeInHierarchy
                    && ((Movimiento.posicion.position == (posicion.position + new Vector3(1, 0, 0)) && Movimiento.ultimo == ultimo_mov.izquierda)
                    || (Movimiento.posicion.position == (posicion.position + new Vector3(-1, 0, 0)) && Movimiento.ultimo == ultimo_mov.derecha)
                    || (Movimiento.posicion.position == (posicion.position + new Vector3(0, -1, 0)) && Movimiento.ultimo == ultimo_mov.arriba)
                    || (Movimiento.posicion.position == (posicion.position + new Vector3(0, 1, 0)) && Movimiento.ultimo == ultimo_mov.abajo)
                    ))
                    {
                        // ...
                    }
                }
            }
        }
    }
}

```

Ilustración 3.180: Código del Script "NPC_NO_NOV" 2

En la imagen podemos ver que lo primero que comprobamos es que no tengamos el menú de pausa abierto. Esto lo conseguimos accediendo a la variable estática **bloqueado_por_script** del susodicho script. Es necesario dado que, si no lo hiciésemos, al pulsar el botón A (o la tecla Enter) se iniciaría una conversación con el NPC a la vez que tendríamos abierto el menú de pausa. También vemos que, al igual que en todos los scripts que leen la información ya sea del teclado o del mando, tenemos un timer y la ya clásica comprobación de si estamos usando un Gamepad o no. A parte de esto, tenemos un if donde comprobamos si el jugador (accediendo a la posición de este, que es un atributo estático) está pulsando la tecla o botón adecuado mientras mira al NPC en cuestión y está situado a 1 casilla de distancia, en cuyo caso, iniciará la conversación (a través de la función **Trigger**), hará que el NPC mire a nuestro jugador, se bloqueará el movimiento de este nuestro jugador y se pondrá a true el booleano que comprueba si se está ya entablando una conversación o no.

```

2 Referencias
public void Trigger()
{
    // ...
    FindObjectOfType<Manager_Dialogos>().Empezar(dialogo);
}

```

Ilustración 3.181: Código del Script "NPC_NO_MOV" 3

La función **Trigger** se encarga de pasar a Manager_Dialogos el diálogo que quiere que se reproduzca llamando a su función **Empezar**.

```

else
{
    timer += Time.deltaTime;
    if (timer > tiempoEspera)
    {
        if (Gamepad.current != null)
        {
            gamepad = Gamepad.current;
            if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()))
            {
                bool aux = SiguienteFrase();
                if (!aux)
                {
                    dialogando = false;
                    Movimiento.bloqueado = false;
                }
                timer = 0;
            }
        }
        else
        {
            if (Input.GetKeyDown(KeyCode.Return))
            {
                bool aux = SiguienteFrase();
                if (!aux)
                {
                    dialogando = false;
                    Movimiento.bloqueado = false;
                }
                timer = 0;
            }
        }
    }
}
}

```

Ilustración 3.182: Código del Script "NPC_NO_MOV" 4

En la siguiente parte del Script, la parte que se ejecutará mientras se está sosteniendo una conversación, es la que vemos en la imagen superior. Dado que ya mandamos el diálogo al mánager ya se debería ver en pantalla la primera frase con el NPC. A partir de ahí, al ir pulsando el botón A, o Enter, se irán mostrando el resto de las frases hasta que ya no quede ninguna, en cuyo caso se cerrará automáticamente (lo controla el Script Manager_Dialogo) el diálogo, y el NPC nos devolverá el movimiento. A su vez este, marcará como falso el booleano dialogando, puesto que ya hemos terminado.



Ilustración 3.183: Ejemplo de Diálogo con NPC del tipo NPC_NO_MOV

A parte de lo ya explicado, tenemos un método llamado **TriggerInicio**, aunque este solo será utilizado por el rey en la primera conversación del juego. La única diferencia es que se pasa un diálogo diferente.

También tenemos una versión de este mismo script que es para personajes que se encuentran detrás de un mostrador pero que no venden nada. La diferencia entre ambos scripts es, a parte de la evidente (que el jugador se encuentre a dos casillas de distancia), que se le hable mirando en la posición contraria a la que mira el NPC. Esto sirve para evitar que hablemos con un NPC a través de una pared, solo podremos hablar con él frente al mostrador, y hemos llamado a este Script “**NPC_NO_MOV_2**”.



Ilustración 3.184: Ejemplo de Diálogo con NPC del tipo NPC_NO_MOV_2

3.4.1.2 NPC_MOV

Como podemos suponer después de ver que NPC_NO_MOV es un NPC que no se mueve, en este caso NPC_MOV es lo contrario, es un NPC que se mueve por la escena de forma aleatoria cada 2 segundos como veremos a continuación.

```

Script de Unity (11 referencias de recurso) | 0 referencias
public class NPC_MOV : MonoBehaviour
{
    private float timer = 0f;
    private float velocidad = 5f;
    private float timer_mover = 0f;
    private const float tiempoEspera = 0.5f;
    private const float tiempoEspera_mover = 2f;

    public bool dialogando = false;
    public Dialogo dialogo;
    public SpriteRenderer sprite_personaje;
    private Gamepad gamepad;
    public LayerMask colliders;
    public Transform posicion;
    public ultimo_mov ultimo = ultimo_mov.abajo;
    public Sprite frente_0;
    public Sprite frente_1;
    public Sprite espalda_0;
    public Sprite espalda_1;
    public Sprite izquierda_0;
    public Sprite izquierda_1;
    public Sprite derecha_0;
    public Sprite derecha_1;
}

```

Ilustración 3.185: Código del Script "NPC_MOV" 1

Las variables son muy parecidas a aquellas que teníamos en NPC_NO_MOV, esto se debe a que el funcionamiento básico es el mismo, solo que se mueve por la escena. Para controlar el área en la que se mueve, cada cuanto tiempo se mueve y demás, hemos hecho uso de una capa de colliders, igual que hacíamos en nuestro jugador, solo que en este caso la capa no será la capa "Collider", sino la capa "Collider_NPC".

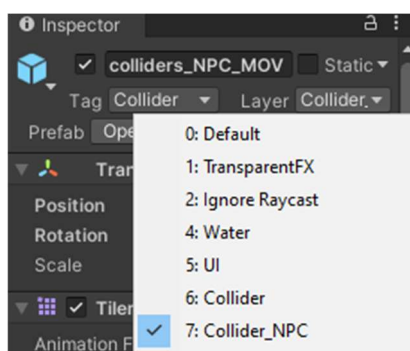


Ilustración 3.186: Colliders NPC

Hacemos este tipo de colliders para evitar que los NPC acaben entrando donde no deben, atravesando paredes, o yéndose fuera de los pueblos y ciudades. Como mencionamos en la parte 3.2.1 de la documentación, el jugador lleva un Grid con un Tilemap con capa "Collider_NPC" en forma de cruz, para evitar que los NPCs se muevan a las direcciones donde podría moverse el jugador. Ya que estamos en un programa con ejecución concurrente de tareas, y, a pesar de comprobar si el hueco está libre en un momento determinado, al no tener una variable atómica para poder

comprobarlo con certeza, siempre puede producirse el caso de que un NPC y nuestro jugador se muevan a la misma casilla a la vez causando un bloqueo irreparable. Por esto es por lo que decidimos poner colliders en forma de cruz en vez de solo en la posición de nuestro jugador. Esto causará que los NPC no se muevan por sí mismos a nuestro lado, pero esto no es algo problemático.

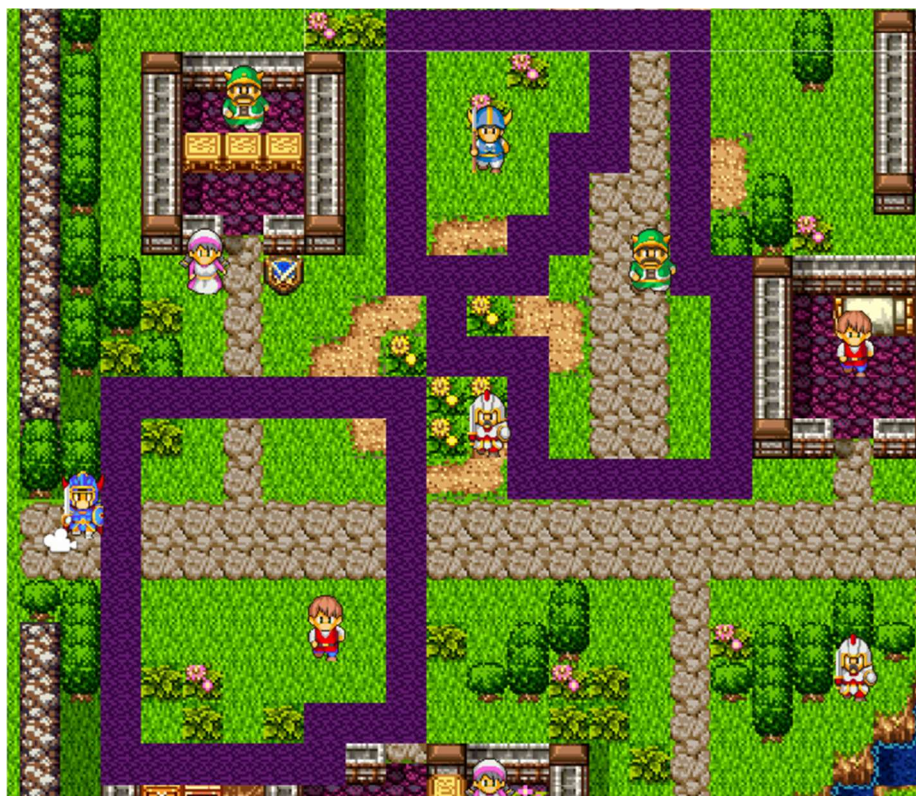


Ilustración 3.187: Colliders NPCs en Breconary

En el método Update de este Script hacemos cosas similares tanto a lo que hace el Script de NPC_NO_MOV, como lo que hace el Script Movimiento de nuestro personaje, ya que primeramente comprobamos si el jugador ha intentado hablar con nosotros (en cuyo caso hacemos lo mismo que hacíamos en NPC_NO_MOV) o si no, donde, si ha pasado el tiempo de espera suficiente, que como ya dijimos, son 2 segundos, nos intentaremos mover en una dirección aleatoria siempre que no choquemos con algún collider que nos lo impida. En la imagen inferior podemos observar cómo efectivamente, hacemos las mismas comprobaciones que para nuestro jugador, pero en este caso para una capa diferente (aunque en el código compartan el nombre, ya que obviamente, reutilizamos el código que se pudo), la capa "Collider_NPC".

```

if(timer_mover > tiempoEspera_mover && !dialogando)
{
    //posibles direcciones de mov
    int random = Random.Range(0, 4);
    if(Movimiento.comprando == false && Movimiento.bloqueado == false)
    {
        Movimiento.bloqueado = true;
        switch (random)
        {
            case 0:
                if (ultimo != ultimo_mov.arriba)
                if (!Physics2D.OverlapCircle(posicion.position + new Vector3(0.5f, 1, 0), .2f, colliders))
                timer_mover = 0;
                break;
            case 1:
                if (ultimo != ultimo_mov.izquierda)
                if (!Physics2D.OverlapCircle(posicion.position + new Vector3(0, 0.5f, 0), .2f, colliders))
                timer_mover = 0;
                break;
            case 2:
                if (ultimo != ultimo_mov.abajo)
                if (!Physics2D.OverlapCircle(posicion.position + new Vector3(0.5f, 0, 0), .2f, colliders))
                timer_mover = 0;
                break;
            case 3:
                if (ultimo != ultimo_mov.derecha)
                if (!Physics2D.OverlapCircle(posicion.position + new Vector3(1, 0.5f, 0), .2f, colliders))
                timer_mover = 0;
                break;
        }
        Movimiento.bloqueado = false;
    }
}

```

Ilustración 3.188: Código del Script "NPC_MOV" 2

El resto del Script no es necesario mostrarlo, ya que como dijimos es igual que el de NPC_NO_MOV, comprobamos si se nos habla, y en ese caso iniciamos una conversación.

3.4.1.3 NPC_OBJETO

Este script es el que utilizamos en los distintos cofres, armarios y jarrones del juego cuando queremos poner objetos que se puedan obtener. Son simples y solo tienen un diálogo que te dicen el objeto que te entregan.

```

private float timer = 0f;
private const float tiempoEspera = 0.5f;

public bool dialogando = false;
public Dialogo dialogo;
public Item objetoAEntregar;
private Gamepad gamepad;
public int id;
public Transform posicion;

```

Ilustración 3.189: Código del Script "NPC_OBJETO" 1

Como vemos tenemos, a parte de las variables que ya se esperaban, una variable id que nos servirá, para, desde el sistema de guardado comprobar si ya se ha abierto el cofre al que estamos accediendo. El código del sistema de guardado para comprobar esto es el siguiente:

```
1 referencia
public static bool LoadCofres(int cofre)
{
    string path = Application.persistentDataPath + "cofres.dqr";
    if (File.Exists(path))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        FileStream fs = new FileStream(path, FileMode.Open);
        DatosCofres datos = formatter.Deserialize(fs) as DatosCofres;
        fs.Close();
        return datos.cofres[cofre];
    }
    else
    {
        return false;
    }
}
```

Ilustración 3.190: LoadCofres

En **LoadCofres** simplemente devolvemos true si ya hemos recibido el objeto del cofre o false si no existe el archivo, o si bien aún no se ha entregado.

```
2 referencia
public static void GuardarCofres(int cofre)
{
    BinaryFormatter formatter = new BinaryFormatter();
    string path = Application.persistentDataPath + "cofres.dqr";
    if (File.Exists(path))
    {
        FileStream fs = new FileStream(path, FileMode.Open);
        DatosCofres datos = formatter.Deserialize(fs) as DatosCofres;
        fs.Close();
        datos.setCofre(cofre);
        fs = new FileStream(path, FileMode.Create);
        formatter.Serialize(fs, datos);
        fs.Close();
    }
    else
    {
        FileStream fs = new FileStream(path, FileMode.Create);
        DatosCofres datos = new DatosCofres();
        datos.setCofre(cofre);
        formatter.Serialize(fs, datos);
        fs.Close();
    }
}
```

Ilustración 3.191: GuardarCofres

Igualmente, en **GuardarCofres**, ponemos a true el valor del cofre pasado. En caso de ya existir el archivo, lo leemos, modificamos y guardamos de nuevo. Si no existe lo creamos y modificamos.

Esto lo conseguimos gracias a una clase que hemos creado específica para guardar el estado de todos los cofres que tenemos en el juego. En total son 45, y por ello el tamaño del array de booleanos es 45.

```

[System.Serializable]
9 referencias
public class DatosCofres
{
    public bool[] cofres;

    1 referencia
    public DatosCofres()
    {
        cofres = new bool[45];
        for(int i = 0; i < cofres.Length; i++)
        {
            cofres[i] = false;
        }
    }

    2 referencias
    public void setCofre(int numero)
    {
        cofres[numero] = true;
    }
}

```

Ilustración 3.192: Clase DatosCofres

Tenemos además un script muy similar que hemos utilizado para entregar la “Marca de Erdrick” un objeto clave (la clase objeto que estamos mencionando la explicaremos en los próximos apartados, cuando hablemos de los NPC_TIENDA), solo que en ese caso queremos entregar el objeto solo cuando no tengamos uno igual en el inventario. Para ello simplemente accedemos al inventario del jugador y comprobamos si tiene dicho objeto.

Ahora mostraremos las comprobaciones que lleva a cabo nuestro script para saber si entregar o no el objeto:

```

if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()))
{
    bool aux = SiguienteFrase();
    if (!aux)
    {
        if (!SistemaGuardado.LoadCofres(id))
        {
            if (Inventario.addObjeto(objetoAEntregar))
            {
                SistemaGuardado.GuardarCofres(id);
                Dialogo auxiliar = new Dialogo();
                auxiliar.nombre = dialogo.nombre;
                auxiliar.frases = new string[1];
                auxiliar.frases[0] = "Has encontrado " + objetoAEntregar.nombre + " ";
                StartCoroutine(Triple(auxiliar));
            }
            else
            {
                Dialogo auxiliar = new Dialogo();
                auxiliar.nombre = dialogo.nombre;
                auxiliar.frases = new string[1];
                auxiliar.frases[0] = "No has podido recibir un objeto porque tienes el inventario lleno.";
                StartCoroutine(Triple(auxiliar));
            }
        }
        else
        {
            dialogando = false;
            Movimiento.bloqueado = false;
        }
    }
}

```

Ilustración 3.193: Código del Script "NPC_OBJETO" 2

3.4.1.4 Anciano_Castillo

Este Script es prácticamente igual a NPC_OBJETO y a NPC_ERDRICK, con la diferencia que es el que se usa en los Santuarios que nos encontraremos por el mundo. Cuando se cumplan las condiciones necesarias para ello, y dependiendo del Santuario en el que estemos se procederá de una u otra forma.

```
if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()))
{
    bool aux = SiguienteFrase();
    if (!aux)
    {
        if (this.gameObject.name == "Sabio_Lluvia")
        {
            if (Inventario.contiene(arpa))...
            else
            {
                dialogando = false;
                Movimiento.bloqueado = false;
            }
        }
        else if (this.gameObject.name == "Sabio_Sagrado")
        {
            if (Inventario.contiene(sagrado))...
            else
            {
                dialogando = false;
                Movimiento.bloqueado = false;
            }
        }
        else
        {
            if (Inventario.contiene(objeto))
            {
                if (this.gameObject.name == "Sabio")
                    SistemaGuardado.GuardarEstadoCastilloPmenos1(true);
                else if (this.gameObject.name == "Sabio_Galenholm")
                    SistemaGuardado.GuardarEstadoGalenholm(true);
                Destroy(gameObject);
            }
            else
            {
                dialogando = false;
                Movimiento.bloqueado = false;
            }
        }
    }
}
```

Ilustración 3.194: Código del Script "Anciano_Castillo"

Como vemos es muy similar, salvo que aquí se guarda otra información que servirá para que el mánager de esa escena en concreto sepa si ya se ha accedido a la ubicación que desbloquea este Script o no, para eliminar o no al hombre que guarda la entrada.

3.4.1.5 NPC_TIENDA

Este NPC es el que se encargará tanto de vendernos objetos en los distintos pueblos que vayamos visitando, como de comprarnos aquellas cosas que ya no necesitamos. Es de todos los NPCs el que más nos ha costado implementar ya que tiene que encargarse de manejar correctamente toda la interfaz de tienda, así como de añadir o no, los distintos objetos que intentemos comprar o vender.


```

private float timer = 0f;
private float timer_mover = 0f;
private const float tiempoEspera = 0.5f;

public bool dialogando = false;
public bool comprando = false;
public bool vendiendo = false;
private int seleccionado = 0;
private int maxDibujado = 4;
private int minDibujado = 0;
public GameObject prefabObjeto;
public Sprite transparente;
public Dialogo dialogo;
public Dialogo irse;
public Dialogo comprar_vender;
public Item[] objetosVenta;
public GameObject compra_venta;
public GameObject cuadro_dialogo;
public GameObject tienda;
public SpriteRenderer sprite_personaje;
private Gamepad gamepad;
public Transform posicion;
public ultimo_mov ultimo = ultimo_mov.abajo;
public GameObject mostradorObjetos;
public Sprite frente_0;
public Sprite frente_1;
public Sprite espalda_0;
public Sprite espalda_1;
public Sprite izquierda_0;
public Sprite izquierda_1;
public Sprite derecha_0;
public Sprite derecha_1;

```

Ilustración 3.195: Código del Script "NPC_TIENDA" 1

La mayoría de las variables de nuestro script son similares a las de otros ya vistos, con la excepción de los GameObjects compra_venta (es un cuadro para elegir si queremos comprar o vender), cuadro_dialogo, tienda y mostradorObjetos (este último es el objeto llamado Slots que mencionamos anteriormente en esta documentación, y donde instanciaremos los prefab de "Item"). Ahora bien, para explicar correctamente el funcionamiento de nuestro NPC_TIENDA, primeramente, necesitamos que existan los objetos, así como el inventario del jugador.

```

[CreateAssetMenu(menuName = "Datos Objetos")]
[System.Serializable]
public class Item : ScriptableObject
{
    public string id;
    public string nombre;
    [TextArea]
    public string descripcion;
    public bool equipado;
    public Sprite icono;

    //Stats
    public int ataque = 0;
    public int defensa = 0;
    public int bonusHP = 0;
    public int bonusMP = 0;
    public int agilidad = 0;
    public int precio = 0;

    public pasivas pasiva = pasivas.ninguna;
    public tipoObjeto tipoObjeto;

    0 referencias
    public void equipar()
    {
        equipado = true;
    }

    0 referencias
    public void desequipar()
    {
        equipado = false;
    }
}

```

Ilustración 3.196: Clase Item

Para crear los distintos objetos hemos creado una clase scripteable llamada Item. Además de esto, también necesitamos que sea Serializable para poder guardar qué objetos tendremos en nuestro inventario, de ahí ambas líneas encima del nombre de la clase.

Se puede ver que tenemos atributos para ataque, defensa, bonusHP, bonusMP, agilidad... Así como, un precio (al que se venderá en la tienda) y unas habilidades pasivas, además del tipo de objeto. Estas dos últimas son enums:

```
2 referencias
public enum pasivas
{
    ninguna,
    hpRecover,
    mpRecover,
    fuegoResist,
    sleepResist,
    maldito
}

43 referencias
public enum tipoObjeto
{
    escudo,
    arma,
    armadura,
    casco,
    accesorio,
    consumible,
    clave
}
```

Ilustración 3.197: Pasivas y tipoObjeto

Al hacer los objetos scripteables lo que conseguimos es la capacidad de crearlos desde el editor de Unity, de este modo:

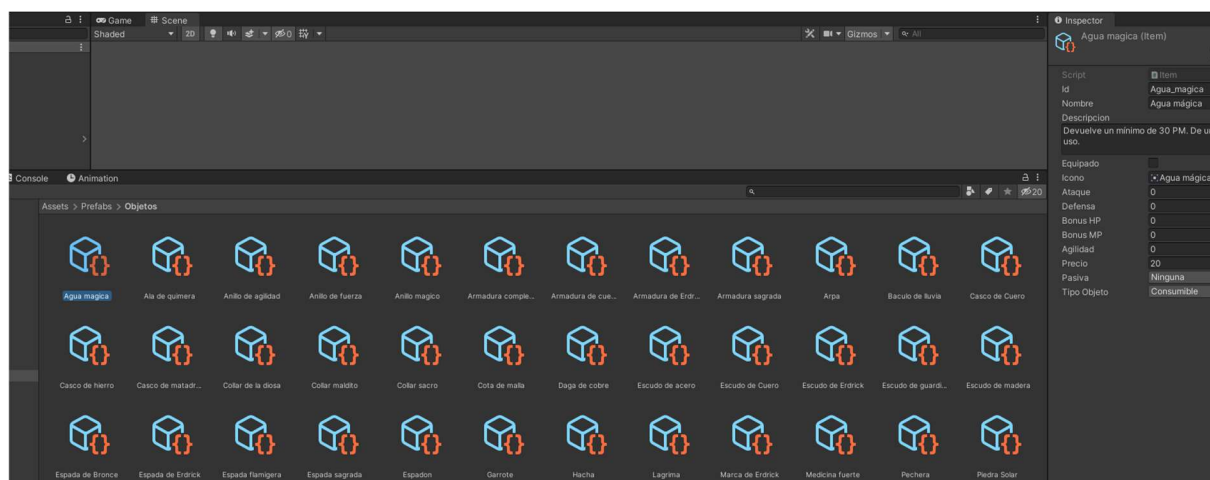


Ilustración 3.198: Item como objeto Scripteable

Nosotros hemos añadido un total de 44 objetos diferentes al juego con sus propias características. Si bien no vamos a poner toda la información relativa a estadísticas y pasivas de los objetos, sí diremos qué objetos hemos creado, su descripción y con qué imagen los podemos reconocer en el juego.

3.4.1.5.1 Consumibles



Ilustración 3.199: Agua mágica

El agua mágica restaura al menos 30 puntos de maná y solo se puede utilizar una vez.



Ilustración 3.200: Ala de quimera

El ala de quimera sirve para teletransportarte a un lugar seguro, es decir, si estás en mitad del mundo y tienes poca vida, puedes usarla para volver a un lugar seguro.



Ilustración 3.201: Medicina fuerte

La medicina fuerte recupera al menos 90 puntos de salud. Cuando se usa se pierde.



Ilustración 3.202: Planta medicinal

La planta medicinal es de un solo uso y te recupera al menos 30 puntos de salud.



Ilustración 3.203: Semilla de agilidad

Aumenta tu agilidad de forma permanente entre 2 y 4 puntos.



Ilustración 3.204: Semilla de defensa

Aumenta tu defensa de forma permanente entre 2 y 4 puntos.



Ilustración 3.205: Semilla de fuerza

Aumenta tu fuerza de forma permanente entre 2 y 4 puntos.



Ilustración 3.206: Semilla de vitalidad

Aumenta tu vitalidad de forma permanente entre 3 y 6 puntos



Ilustración 3.207: Semilla mágica

Aumenta tu maná de forma permanente entre 3 y 6 puntos.

3.4.1.5.2 Armas



Ilustración 3.208: Garrote

Un bastón de madera para la gente mayor. Aunque puede tener otros usos.



Ilustración 3.209: Espada de bronce

Un arma directa y dúctil fabricada con sólido cobre.



Ilustración 3.210: Daga de cobre

Una pequeña y sencilla arma forjada con cobre bruñido.



Ilustración 3.211: Espadón

Mandoble forjado a partir de acero que suele requerir usar ambos manos (a no ser que seas muy fuerte).



Ilustración 3.212: Espada flamígera

Se dice que esta espada fue forjada con el fuego de un dragón rojo y que aumenta la resistencia al fuego.



Ilustración 3.213: Espada sagrada

Un arma bendecida por los dioses que puede ayudar a recuperar vida cuando golpea.



Ilustración 3.214: Hacha

Gran hacha que infunde respeto al enemigo. Con ella saldrán corriendo.



Ilustración 3.215: Espada de Erdrick

Un brillante símbolo de la esperanza de Alefgard cuya hoja bendita es capaz de destruir el mal.

3.4.1.5.3 Armaduras



Ilustración 3.216: Armadura completa

Una armadura casi impenetrable que protege contra todos los tipos de ataque.



Ilustración 3.217: Armadura de cuero

Ropa de guerra ligera fabricada con la piel endurecida de monstruos.



Ilustración 3.218: Ropa de viaje

Atuendo de arpillera resistente, popular entre aquellos que pasan mucho tiempo de viaje.



Ilustración 3.219: Cota de malla

Es una armadura ligera capaz de reducir el daño recibido por el enemigo.



Ilustración 3.220: Pechera

Armadura robusta fabricada con un mineral desconocido.



Ilustración 3.221: Armadura sagrada

Armadura que posee un sistema de protección bendito que bloquea parte del daño mágico recibido.



Ilustración 3.222: Armadura de Erdrick

Ropaje del Luminario legendario. Ayuda a restablecer los HP fuera del combate.

3.4.1.5.4 Escudos



Ilustración 3.223: Escudo de madera

Tapa de barril reformada para usarse como escudo.



Ilustración 3.224: Escudo de cuero

Una gran rodela fabricada estirando piel de animal sobre una plataforma de madera.



Ilustración 3.225: Escudo de acero

Un escudo protector resistente para los guerreros curtidos.



Ilustración 3.226: Escudo de guardián

Rodela negra inmaculada que utilizan los guerreros más experimentados. Tiene una defensa bastante alta.



Ilustración 3.227: Escudo de Erdrick

Portado otrora por el Luminario legendario, resiste todos los tipos de ataques y magia. Además, te hace invulnerable al sueño.

3.4.1.5.5 Cascos



Ilustración 3.228: Casco de cuero

Tocado resistente fabricado de pieles curtidas y endurecidas cosidas.



Ilustración 3.229: Casco de hierro

Un resistente yelmo hecho de duro acero



Ilustración 3.230: Casco de matadragones

Yelmo que porta aquel que suele enfrentarse a dragones. Aumenta algo la resistencia al fuego.



Ilustración 3.231: Tiara de Erdrick

Portada otrora por el Luminario legendario. Aumenta la recuperación de MP fuera del combate.

3.4.1.5.6 Accesorios



Ilustración 3.232: Anillo de agilidad

Un anillo fortalecedor que potencia la agilidad de quien lo lleva.



Ilustración 3.233: Anillo de fuerza

Anillo con púas que está bendecido con un conjuro que potencia el poder de ataque.



Ilustración 3.234: Anillo mágico

Una alianza bendita que aumenta algo los PM.



Ilustración 3.235: Collar de la diosa

Un collar bendito que aumenta la regeneración de MP fuera de batalla.



Ilustración 3.236: Collar sacro

Collar sagrado que protege a su portador de dormirse en batalla.



Ilustración 3.237: Collar maldito

Un artefacto horrible y de mal agüero. Se dice que se encontró en el fondo de un despeñadero.

3.4.1.5.7 Clave



Ilustración 3.238: Arpa de Galen

Arpa cuyo sonido atrae a los monstruos.



Ilustración 3.239: Lágrima de la Diosa

Lagrima de la Diosa que erradica el mal.



Ilustración 3.240: Báculo de la Lluvia

Báculo mágico que contiene el poder de la lluvia en su interior.



Ilustración 3.241: Marca de Erdrick

Un emblema que portaba el Luminario original. Prueba de que eres su descendiente.



Ilustración 3.242: Piedra Solar

Roca misteriosa que contiene el poder del sol en su interior.

Una vez tenemos creados todos los objetos que utilizaremos, pasamos a implementar el inventario.

```
[System.Serializable]
Script de Unity (1 referencia de recurso) | 99+ referencias
public class Inventario : MonoBehaviour
{
    public Item[] objetos;
    private const int maxItems = 10;
    private static int num_items = 0;
    private const int maxStackedItems = 5;
    public static List<Item>[] inventario;
    public static int dinero;
    public static int casco;
    public static int arma;
    public static int armadura;
    public static int escudo;
    public static int accesorio;
}
```

Ilustración 3.243: Código del Script "Inventario" 1

En nuestro inventario hemos definido un máximo de 10 tipos de objetos diferentes, pudiendo juntar un máximo de 5 de cada tipo, a excepción de objetos equipables que como veremos solo podremos llevar 1. Tenemos una variable para el dinero y tenemos un array con todos los objetos del juego, para poder hacer las consultas necesarias. La estructura que hemos utilizado para nuestro inventario ha sido un array de listas de tipo Item, que son bastante fáciles de trabajar.

En el Script Inventario, a diferencia de la mayoría de los scripts que heredan de MonoBehaviour de nuestro juego, no tiene método Update, pero sí que hace uso del método Start, en el cual comprueba el archivo de guardado para recuperar su inventario:

```
{
    if(inventario == null)
    {
        inventario = new List<Item>[maxItems];
        DatosInventario aux = SistemaGuardado.LoadInventario();
        if (aux != null)
        {
            dinero = aux.dinero;
            for (int i = 0; i < aux.nombreObjetos.Length; i++)
            {
                foreach (Item item in objetos)
                {
                    if (item.nombre == aux.nombreObjetos[i])
                    {
                        inventario[num_items] = new List<Item>();
                        for (int j = 0; j < aux.numExistencias[i]; j++)
                        {
                            inventario[num_items].Add(item);
                        }
                        num_items++;
                        break;
                    }
                }
            }
        }
    }
}
```

Ilustración 3.244: Código del Script "Inventario" 2

Si aux no es nulo, cargamos los datos guardados en nuestro inventario. Para esto es que tenemos un array con todos los objetos del juego.

La clase que se encarga de almacenar los datos del inventario y de la que los obtenemos de vuelta es la siguiente:

```
[System.Serializable]
7 referencias
public class DatosInventario
{
    //Datos inventario
    public string[] nombreObjetos;
    public int[] numExistencias;
    public int dinero;
    //public bool[] equipados;

    1 referencia
    public DatosInventario()
    {
        if(Inventario.Size() != 0)
        {
            nombreObjetos = new string[Inventario.Size()];
            numExistencias = new int[Inventario.Size()];
            dinero = Inventario.dinero;
            int cont =0;
            foreach (List<Item> items in Inventario.getInventario()
            {
                if (items is null)
                    break;
                else
                {
                    nombreObjetos[cont] = items[0].nombre;
                    numExistencias[cont] = items.Count;
                }
                cont++;
            }
        }
        else
        {
            dinero = Inventario.dinero;
            nombreObjetos = new string[Inventario.Size()];
            numExistencias = new int[Inventario.Size()];
        }
    }
}
```

Ilustración 3.245: Clase DatosInventario

Necesitamos una clase de este tipo ya que de otro modo no podemos guardar los datos. El sistema de serialización de Unity solo permite serializar tipos de datos simples como, int, float, string, arrays [], etc. Pero no permite serializar objetos de otros tipos. Por eso la clase DatosInventario se compone de tipos que Unity puede serializar.

Siguiendo con nuestro inventario, este está formado por los siguientes métodos:

```
2 referencias
public static estadoCompra comprarItem(int precio, Item objeto)...

4 referencias
public static bool addObjeto(Item item)...

6 referencias
public static void ventaItem(int precio, Item objeto, int cantidad=1)...

27 referencias
public static List<Item>[] getInventario()...

2 referencias
public static void equipar(int seleccionado)...

2 referencias
public static void desequipar(int seleccionado)...

7 referencias
public static string usarObjeto(int seleccionado)...

12 referencias
public static bool contiene(Item item)...

2 referencias
public static int getPos(Item item)...

14 referencias
public static void eliminarItem(int seleccionado, int cantidad)...

62 referencias
public static int Size()...
```

Ilustración 3.246: Métodos del Script "Inventario"

Como vemos tenemos métodos habituales que podríamos encontrar en una lista, tales como **Size**, **getPos**, **contiene**, **eliminarItem**, **addObjeto**... Estos hacen básicamente lo que dicen. Size te devuelve el número de objetos diferentes que tienes en tu array, getPos te da la posición de un objeto o -1 si no lo encuentra, contiene te dice si existe el objeto que estás buscando en el inventario, eliminarItem elimina el objeto en la posición indicada, addObjeto intenta añadir un objeto y devuelve false o true dependiendo de si se consigue o no. Al ser todos métodos típicos que cualquier lista posee no los explicaremos en profundidad, simplemente bastará con mencionarlos para que sepamos sus funciones.

Además de esos métodos tenemos otros que son más interesantes y que serán los que nuestro NPC_TIENDA utilice para intentar vender o comprar cosas con nosotros. En el caso de que le vayamos a vender algo al NPC_TIENDA, tenemos la siguiente función:

```
public static void ventaItem(int precio, Item objeto, int cantidad=1)
{
    int cont = 0;
    foreach(List<Item> items in inventario)
    {
        if(items[0].id == objeto.id)
        {
            if(items.Count == cantidad)
            {
                for(int i = cont; i < num_items-1; i++)
                {
                    inventario[i] = inventario[i+1];
                }
                inventario[num_items] = null;
                num_items--;
            }
            else
            {
                for (int i = 0; i < cantidad; i++)
                {
                    items.RemoveAt(0);
                }
                dinero += precio;
                break;
            }
        }
        else
        {
            cont++;
        }
    }
}
```

Ilustración 3.247: Código del Script "Inventario" 3

Buscamos el objeto que vamos a vender y lo eliminamos, haciendo los cambios necesarios si nos quedamos sin ningún objeto de ese tipo (correr el resto de los valores a la izquierda y dejar el hueco al final, reduciendo después el total de objetos contenidos). Y, por supuesto, añadimos el dinero que nos haya ofrecido el NPC.

La función de compra es diferente y hace uso de un enum nuevo que contiene las diferentes posibilidades a la hora de intentar comprar un objeto:

```
19 referencias
public enum estadoCompra
{
    inventario_lleno,
    objeto_lleno,
    no_dinero,
    repetido,
    comprado
}
```

Ilustración 3.248: Enum estadoCompra

Puede ocurrir que no tengamos sitio en el inventario, o que tengamos el máximo de unidades de ese objeto, que no tengamos dinero, que ya tengamos un objeto de ese tipo (por ejemplo, un garrote), o que, en efecto, se realice la compra.

```
public static estadoCompra comprarItem(int precio, Item objeto)
{
    if (dinero < precio)
    {
        return estadoCompra.no_dinero;
    }
    else
    {
        foreach (List<Item> items in inventario)
        {
            if (items is null)
            {
                break;
            }
            else
            {
                if (items[0].id == objeto.id)
                {
                    if (objeto.tipoObjeto != tipoObjeto.consumible)
                    {
                        return estadoCompra.repetido;
                    }
                    else
                    {
                        if (items.Count == maxStackedItems)
                        {
                            return estadoCompra.objeto_lleno;
                        }
                        else
                        {
                            dinero -= precio;
                            items.Add(objeto);
                            return estadoCompra.comprado;
                        }
                    }
                }
            }
        }
    }
}
```

Ilustración 3.249: Código del Script "Inventario" 4

En el método **comprarItem** vemos que primero comprobamos el dinero, si no tenemos, devolvemos ese estado. Si sí tenemos dinero, miramos entre los objetos que tenemos en nuestro inventario para ver si alguno coincide con el que intentamos comprar, y en caso de hacerlo nos aseguramos de que sea un objeto consumible, o de otro modo devolver que es un objeto repetido. Si, ha coincidido con un objeto y este es de tipo consumible, comprobamos el número de objetos de ese tipo que tenemos, y si no ha alcanzado el máximo lo compramos, si por el contrario ya tenemos 5 copias de ese objeto, no lo podemos comprar.

```
if (num_items != maxItems)
{
    dinero -= precio;
    inventario[num_items] = new List<Item>();
    inventario[num_items].Add(objeto);
    num_items++;
    return estadoCompra.comprado;
}
else
{
    return estadoCompra.inventario_lleno;
}
```

Ilustración 3.250: Código del Script "Inventario" 5

Si no coincidió con ningún objeto comprobamos si queda sitio para añadir un tipo nuevo de objeto, si sí lo hay lo añadimos, y si no, devolvemos que el inventario está lleno.

Estas son las funciones principales que utilizará nuestro NPC_TIENDA, del cual hablaremos a continuación.

```

Mensaje de Unity | 0 referencias
void Update()
{
    if(Menu_Pausa.bloqueado_por_script == false)
    {
        if (!dialogando)...
        else if (!tienda.gameObject.activeInHierarchy)...
        else...
    }
}

```

Ilustración 3.251: Código del Script "NPC_TIENDA" 2

El NPC de la tienda primero, como todos los NPCs, comprueba que no tengamos el menú de pausa, en cuyo caso divide su funcionalidad como vemos en la imagen. Primeramente, como no estamos dialogando se ejecuta esa parte, en la que esperará a que hablemos con él, exactamente igual que el resto de NPCs. En la siguiente parte, que se ejecutará una vez entablemos conversación con él, pero aún no hayamos elegido si vamos a comprar o vender, se nos mostrará un diálogo igual que en los NPCs normales, solo que a continuación se nos preguntará si queremos comprar o vender.

```

if (!compra_venta.gameObject.activeInHierarchy)
{
    if (Gamepad.current != null)
    {
        gamepad = Gamepad.current;
        if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()))
        {
            bool aux = SiguienteFrase();
            if (!aux)
            {
                compra_venta.gameObject.SetActive(true);
            }
            timer = 0;
        }
    }
    else
    {
        if (Input.GetKeyDown(KeyCode.Return))
        {
            bool aux = SiguienteFrase();
            if (!aux)
            {
                compra_venta.gameObject.SetActive(true);
            }
            timer = 0;
        }
    }
}

```

Ilustración 3.252: Código del Script "NPC_TIENDA" 3

Aquí vemos que si no está compra_venta activo (compra_venta es el cuadro donde se nos da las opciones para comprar y vender) se trata de un diálogo normal como con cualquier otro NPC. Pero cuando se activa, tendremos que elegir entre comprar y vender.

```
if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()))
{
    if (compra_venta.transform.Find("Image").transform.Find("Si").transform.Find("cursor_si").gameObject.activeInHierarchy)
    {
        compra_venta.gameObject.SetActive(false);
        cuadro_dialogo.gameObject.SetActive(false);
        tienda.gameObject.SetActive(true);
        StartCoroutine(animarFrase("¿Qué desea comprar? ¡Tenemos de todo!"));
    }
    else
    {
        if(Inventario.Size() == 0)
        {
            Dialogo n = new Dialogo();
            n.nombre = "Tienda";
            n.frases = new string[1];
            n.frases[0] = "Lo siento, pero no llevas nada que comprar encima.";
            FindObjectOfType<Manager_Dialogos>().EmpezarTienda(n);
        }
        else
        {
            compra_venta.gameObject.SetActive(false);
            cuadro_dialogo.gameObject.SetActive(false);
            tienda.gameObject.SetActive(true);
            vendiendo = true;
            StartCoroutine(animarFrase("¿Qué desea vender? Compramos casi todo."));
            compra_venta.transform.Find("Image").transform.Find("Si").transform.Find("cursor_si").gameObject.SetActive(true);
            compra_venta.transform.Find("Image").transform.Find("No").transform.Find("cursor_no").gameObject.SetActive(false);
        }
    }
    timer = 0;
}
```

Ilustración 3.253: Código del Script "NPC_TIENDA" 4

Cuando pulsamos el botón del mando o Enter, comprobamos si se ha seleccionado comprar, en ese caso se muestra un mensaje y se activa la tienda. En caso de darle a vender, primeramente, se comprueba que no tengamos el inventario vacío, ya que no podríamos vender nada, y si no está vacío podremos vender lo que queramos. También ponemos el booleano vender a true, que será lo que diferencia comprar y vender en el próximo código que se ejecutará.

```
else
{
    List<Item>[] inventarioAuxiliar = Inventario.getInventario();
    tienda.transform.Find("Dinero").transform.Find("Dinero").GetComponent<Text>().text = Inventario.dinero.ToString() + " G";
    timer += Time.deltaTime;
    if (!comprando && !vendiendo)
    else if (comprando && !vendiendo)
    else if (vendiendo && !comprando)
    else if (vendiendo && comprando)
}
```

Ilustración 3.254: Código del Script "NPC_TIENDA" 5

Aquí primeramente asignamos a la variable de la tienda nuestro dinero, de modo que siempre podamos saber cuánto dinero tenemos y qué podemos comprar. Seguido, podemos ver que se diferencian 4 posibilidades, la primera, que sería la que se ejecutaría cuando compramos, pero aún no hemos instanciado los prefabs de Item en el ya mencionado objeto Slots que los ordenará verticalmente. El siguiente se ejecutará para la compra una vez ya hayamos instanciado dichos objetos, y el resto corresponde a lo mismo, pero para la venta, en cuyo caso en vez de instanciar los objetos que tiene el NPC, instanciamos los que tenemos en nuestro inventario.

```

if (Input.GetKeyDown(KeyCode.W) || gamepad.dpad.up.IsPressed())
{
    if (itemsInventario > 5)
    {
        if (seleccionado != 0)
        {
            seleccionado--;
            timer_mover = 0;
            if (maxDibujado - seleccionado == 5)
            {
                for (int i = 0; i < mostradorObjetos.transform.childCount; i++)
                {
                    Destroy(mostradorObjetos.transform.GetChild(i).GetComponent<Image>());
                    Destroy(mostradorObjetos.transform.GetChild(i).gameObject);
                }
                for (int i = seleccionado; i < seleccionado + 5; i++)
                {
                    GameObject aux = Instantiate(prefabObjeto, new Vector3(0, 0, 0), Quaternion.identity);
                    aux.transform.Find("Icono").GetComponent<Image>().sprite = inventarioAux[i][0].icono;
                    aux.transform.Find("Nombre").GetComponent<Text>().text = inventarioAux[i][0].nombre.ToString();
                    aux.transform.Find("Precio").GetComponent<Text>().text = inventarioAux[i][0].precio.ToString();
                    if (i == seleccionado)
                    {
                        aux.transform.Find("Seleccionado").gameObject.SetActive(true);
                        aux.transform.Find("Cursor").gameObject.SetActive(true);
                    }
                    aux.transform.SetParent(mostradorObjetos.transform);
                }
                maxDibujado--;
                minDibujado--;
            }
            else
            {
                mostradorObjetos.transform.GetChild(seleccionado - minDibujado + 1).transform.Find("Seleccionado").gameObject.SetActive(false);
                mostradorObjetos.transform.GetChild(seleccionado - minDibujado + 1).transform.Find("Cursor").gameObject.SetActive(false);

                mostradorObjetos.transform.GetChild(seleccionado - minDibujado).transform.Find("Seleccionado").gameObject.SetActive(true);
                mostradorObjetos.transform.GetChild(seleccionado - minDibujado).transform.Find("Cursor").gameObject.SetActive(true);
            }
        }
        else
        {
            if (seleccionado != 0)
            {
                mostradorObjetos.transform.GetChild(seleccionado).transform.Find("Seleccionado").gameObject.SetActive(false);
                mostradorObjetos.transform.GetChild(seleccionado).transform.Find("Cursor").gameObject.SetActive(false);
                seleccionado--;
                mostradorObjetos.transform.GetChild(seleccionado).transform.Find("Seleccionado").gameObject.SetActive(true);
                mostradorObjetos.transform.GetChild(seleccionado).transform.Find("Cursor").gameObject.SetActive(true);
                timer_mover = 0;
            }
        }
    }
}

```

Ilustración 3.255: Actualización del mostrador de objetos al movernos por la tienda

No queremos dar demasiados detalles del manejo de la interfaz para que todo vaya correctamente, y por ello solo vamos a enfocarnos en lo principal que es el acto de comprar o vender un objeto, y si tuviésemos que explicar las más de 1600 líneas de código que tenemos en el NPC para que se corresponda lo que ocurre con lo que se ve en pantalla acabaríamos teniendo una documentación excesivamente larga. Con mencionar que se ha trabajado para que todo vaya correctamente cuando borramos objetos, compramos nuevos, equipamos otros, etc. es suficiente. Ya que cada vez que cambiamos de objetos hay que actualizar el objeto seleccionado, todo el mostrador de objetos (la parte derecha de la tienda, que vemos en la foto de abajo),

cuando borramos hay que eliminar todos los objetos instanciados y volver a crearlos teniendo en cuenta el que se ha borrado (porque se ha vendido), etc.

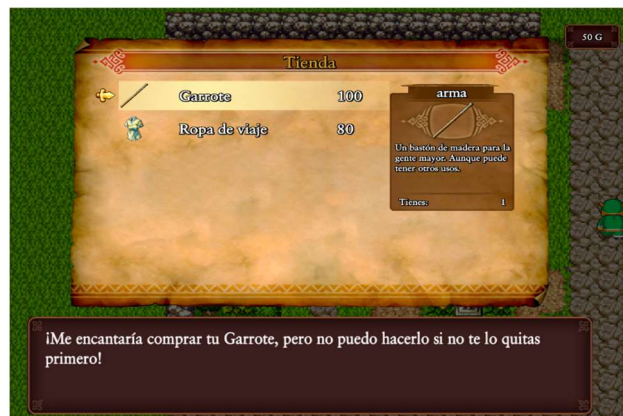


Ilustración 3.256: Tienda - Vender

```
else if (timer > tiempoEspera && (Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()))
{
    estadoCompra aux = Inventario.comprarItem(objetosVenta[seleccionado].precio, objetosVenta[seleccionado]);
    switch (aux)
    {
        case estadoCompra.inventario_lleno:
            StopAllCoroutines();
            StartCoroutine(animarFrase("Lo siento, pero no puedes llevar más objetos, tu inventario está completo."));
            break;
        case estadoCompra.objeto_lleno:
            StopAllCoroutines();
            StartCoroutine(animarFrase("Lo siento, pero no puedes llevar más " + objetosVenta[seleccionado].nombre + "."));
            break;
        case estadoCompra.no_dinero:
            StopAllCoroutines();
            StartCoroutine(animarFrase("No tienes suficiente dinero para comprarlo."));
            break;
        case estadoCompra.comprado:
            StopAllCoroutines();
            StartCoroutine(animarFrase("¡Muchas gracias por su compra! ¡Espero que disfrute su " + objetosVenta[seleccionado].nombre + "!"));
            break;
        case estadoCompra.repetido:
            StopAllCoroutines();
            StartCoroutine(animarFrase("¡Jajaja! ¡Parece que ya tiene uno de esos! Como vendedor quiero vender cosas, pero... ¿Para qué vas a utilizar dos "+objetosVenta[seleccionado].nombre+" ig"));
            break;
    }
    timer = 0;
}
```

Ilustración 3.257: Código del Script "NPC_TIENDA" 6

Cuando pulsamos el botón o Enter, el NPC hace el intento de vendernos el objeto, y según el resultado que obtenga nos mostrará por pantalla uno u otro mensaje para hacernos saber lo que ha ocurrido.

```
else if (timer > tiempoEspera && (Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()))
{
    if (inventarioAux[seleccionado][0].equipado == true)
    {
        StopAllCoroutines();
        StartCoroutine(animarFrase("¡Me encantaría comprar tu " + inventarioAux[seleccionado][0].nombre + ", pero no puedo hacerlo si no te lo quitas primero!"));
        return;
    }
    if (maxDibujado == seleccionado && inventarioAux[seleccionado].Count == 1)
    else if (inventarioAux[seleccionado].Count == 1)
    {
        Inventario.ventaItem(int.Parse((inventarioAux[seleccionado][0].precio * 0.3f).ToString()), inventarioAux[seleccionado][0]);
        StopAllCoroutines();
        StartCoroutine(animarFrase("¡Muchas gracias! Haremos buen uso de sus objetos."));
    }
    timer = 0;
}
```

Ilustración 3.258: Código del Script "NPC_TIENDA" 7

La parte de venta es similar, primeramente, se comprueba si lo que le estás vendiendo está equipado o si es un objeto clave (en la imagen no aparece, pero se ha añadido al código), y si no es, se vende. Se tienen en cuenta diferentes situaciones de ahí los distintos if-else, pero en todos casos el resultado es el mismo, la venta del objeto.

3.4.1.6 NPCs utilizados

Ahora que ya hemos explicado los Scripts que utilizan nuestros NPC vamos a mostrar los distintos NPCs que tenemos.

3.4.1.6.1 Aldeana_1

Para este NPC hemos utilizado los siguientes Sprites



Ilustración 3.259: Sprites Aldeana_1

3.4.1.6.2 Aldeana_2

Para este NPC hemos utilizado los siguientes Sprites

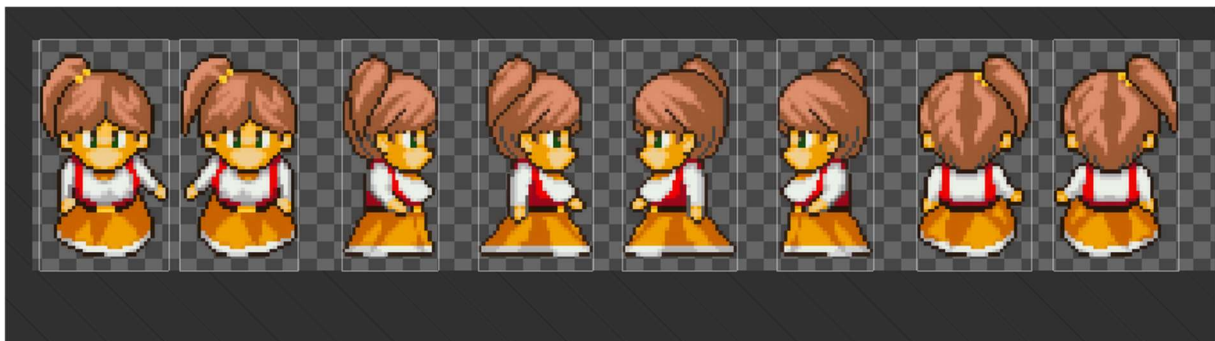


Ilustración 3.260: Sprites Aldeana_2

3.4.1.6.3 Aldeano_1

Para este NPC hemos utilizado los siguientes Sprites



Ilustración 3.261: Sprites Aldeano_1

3.4.1.6.4 Aldeano_2

Para este NPC hemos utilizado los siguientes Sprites



Ilustración 3.262: Sprites Aldeano_2

3.4.1.6.5 Aldeano_3 y Tienda

Para estos NPCs se han utilizado los mismos sprites

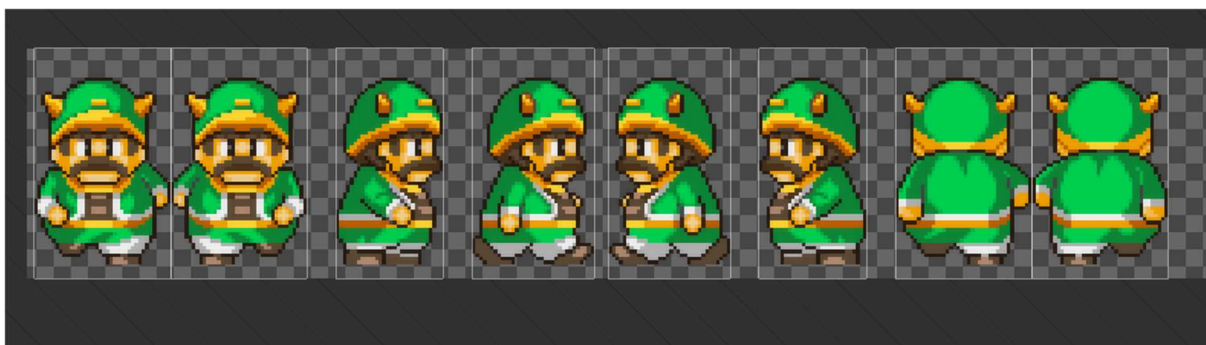


Ilustración 3.263: Sprites Aldeano_3 y Tienda

3.4.1.6.6 Anciana_1

Para este NPC se han utilizado estos Sprites



Ilustración 3.264: Sprites Anciana_1

3.4.1.6.7 Anciano_1, Anciano_durmiendo y todos los Sabios

Tanto para Anciano_1 y Anciano_durmiendo, así como para todos los sabios se han utilizado los siguientes Sprites



Ilustración 3.265: Sprites Anciano_1, Anciano_durmiendo y Sabios

3.4.1.6.8 Hombre, Hombre_durmiendo y Armero

Para Hombre, Hombre_durmiendo y para Armero se han utilizado los mismos Sprites



Ilustración 3.266: Sprites Hombre, Hombre_durmiendo y Armero

3.4.1.6.9 Bardo

Para este NPC se han utilizado los siguientes Sprites:



Ilustración 3.267: Sprites Bardo

3.4.1.6.10 Caballero

Para este NPC se han utilizado los siguientes Sprites:



Ilustración 3.268: Sprites Caballero

3.4.1.6.11 Guardia y Guardia_durmiendo

Tanto para Guardia como para Guardia_durmiendo se han utilizado los siguientes Sprites:



Ilustración 3.269: Sprites Guardia y Guardia_durmiendo

3.4.1.6.12 Monja

Para este NPC se han utilizado los siguientes Sprites:



Ilustración 3.270: Sprites Monja

3.4.1.6.13 Canciller

Para Este NPC se han utilizado los siguientes Sprites:

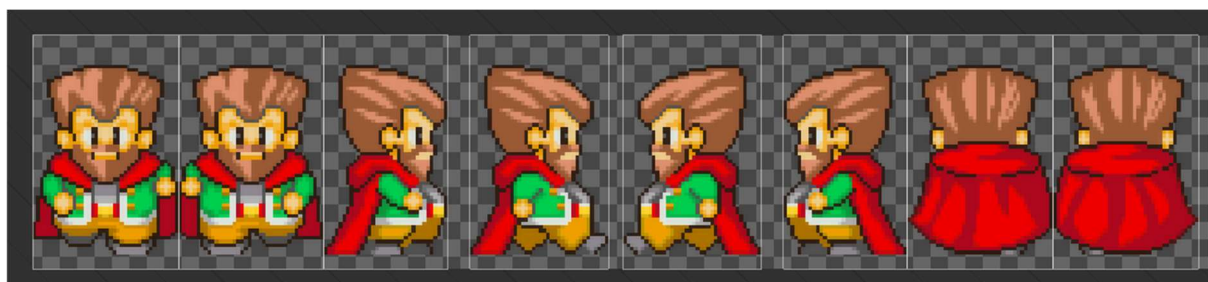


Ilustración 3.271: Sprites Canciller

3.4.1.6.14 Rey

Para este NPC se han utilizado los siguientes Sprites:



Ilustración 3.272: Sprites Rey

3.4.1.6.15 Princesa

Para este NPC se han utilizado los siguientes Sprites:



Ilustración 3.273: Sprites Princesa

3.4.1.6.16 Sacerdote

Para este NPC se han utiliza los siguientes Sprites:

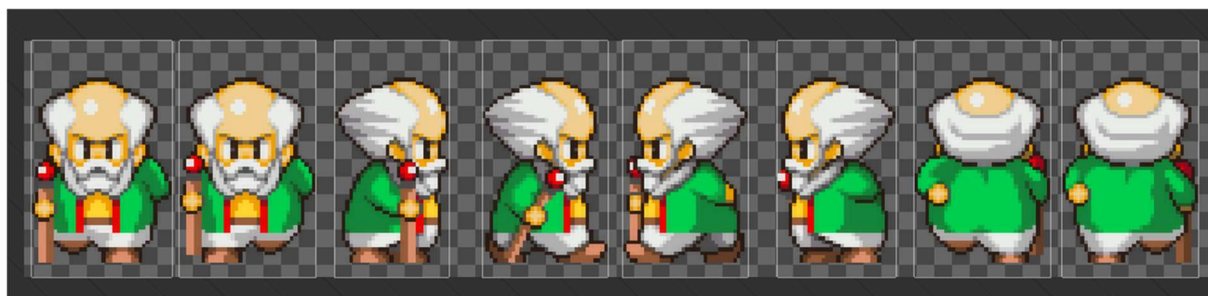


Ilustración 3.274: Sprites Sacerdote

3.4.1.6.17 Dragón Verde Jefe

Para este NPC se utilizan los siguientes Sprites:



Ilustración 3.275: Sprites Dragón Verde Jefe

3.4.1.6.18 DragonLord

Para este NPC se utilizan los siguientes Sprites:



Ilustración 3.276: Sprites DragonLord

3.4.2 Estadísticas

Ahora que ya tenemos los objetos podemos empezar a crear nuestros managers tanto para el menú de pausa como para el combate. Pero primero hemos tenido que crear las estadísticas del jugador, ya que tanto el mánager de pausa como el de batalla utilizan estas estadísticas.

```
[System.Serializable]
Script de Unity | 1 referencia
public class Stats_personaje : MonoBehaviour
{
    public static Stat ataque = new Stat();
    public static Stat defensa = new Stat();
    public static Stat agilidad = new Stat();
    public static Stat magia = new Stat();
    public static Stat def_magica = new Stat();

    public static int vida_maxima;
    public static int vida_actual;
    public static int mana_maximo;
    public static int mana_actual;

    Mensaje de Unity | 0 referencias
    private void Awake()
    {
        vida_actual = vida_maxima;
        mana_actual = mana_maximo;
    }

    26 referencias
    public static bool RecibirDanio(int dmg, bool critico)...

    8 referencias
    public static void RecibirCuracion(int curacion)...

    1 referencia
    public static void RestaurarMana(int curacion)...

    0 referencias
    public static void Morir()...

    3 referencias
    public static void reajustarVida()...

    3 referencias
    public static void reajustarMana()...
```

Ilustración 3.277: Clase Stats_personaje

Como vemos, hemos creado una clase Stats_personaje que tanto nuestro jugador como enemigos utilizarán. Para crear esta clase hemos hecho uso de otra llamada Stat.

```
[System.Serializable]
30 referencias
public class Stat
{
    [SerializeField]
    private int valor_base;

    99+ referencias
    public int get_valor()
    {
        return valor_base;
    }

    15 referencias
    public void set_valor(int nuevo, int antiguo = 0)
    {
        valor_base -= antiguo;
        valor_base += nuevo;
    }

    3 referencias
    public void quitarEquipo(int pieza)
    {
        valor_base -= pieza;
    }
}
```

Ilustración 3.278: Clase Stat

Esta clase Stat implementa dos métodos que nos servirán para modificar los valores de este al equipar/desequipar objetos o subir de nivel. Volviendo a la clase Stats_personaje, tenemos varias funciones para utilizar ya sea al consumir objetos dentro o fuera de batalla, como usar hechizos, como luchar.

```
24 referencias
public static bool RecibirDanio(int dmg, bool critico)
{
    if (critico)
    {
        vida_actual -= dmg * (1 - (Mathf.Clamp(defensa.get_valor(), 0, 175)) / (100 + (Mathf.Clamp(defensa.get_valor(), 0, 175))));
        if (vida_actual <= 0)
        {
            vida_actual = 0;
            return true;
        }
    }
    else
    {
        vida_actual -= dmg * (1 - (Mathf.Clamp(defensa.get_valor(), 0, 175)) / (75 + (Mathf.Clamp(defensa.get_valor(), 0, 175))));
        if (vida_actual <= 0)
        {
            vida_actual = 0;
            return true;
        }
    }
    return false;
}
```

Ilustración 3.279: Código del Script "Stats_personajes" 1

A la hora de luchar hemos definido funciones para recibir daño, sea daño normal o daño crítico, al igual que también lo hemos hecho para infligirlo. Las

siguientes funciones representan por orden daño, daño crítico, defensa, defensa crítica:

$$Daño = \left(Ataque - \frac{Defensa}{2} \right) * Random(0.95, 1.05) \quad (3.1)$$

$$Daño_{critico} = 1.45 * \left(Ataque - \frac{Defensa}{2} \right) * Random(1, 1.1) \quad (3.2)$$

$$Daño_{recibido} = Daño * \left(1 - \frac{defensa_{max=175}}{75 + defensa_{max=175}} \right) \quad (3.3)$$

$$Daño_{recibido\ critico} = Daño_{critico} * \left(1 - \frac{defensa_{max=175}}{100 + defensa_{max=175}} \right) \quad (3.4)$$

Usando estas funciones hemos tratado de balancear lo máximo posible el sistema de combate (que veremos en la subsección posterior a esta), consiguiendo números que hacen que el juego sea complicado los primeros niveles, luego se vuelva algo más fácil, pero al cambiar de área y encontrarnos con monstruos más fuertes volveremos a pasarlo difícil para sobrevivir. Creemos que así podemos conseguir un buen nivel de dificultad, pero aun así dándole a todo el mundo la posibilidad de pasarse el juego sin mayores dificultades.

Luego tenemos una función para recibir curaciones, pero no consideramos necesario explicarla ya que simplemente es añadir vida y en caso de pasarnos del límite, reajustarla con la función reajustarVida, y lo mismo sirve para el maná.

Nuestro jugador ha heredado de la clase Stats_personaje en una nueva clase llamada Stats_jugador, donde se incluye la información adicional que necesita nuestro jugador, pero que no necesitan los enemigos, como pueden ser el nivel, la experiencia acumulada, los distintos niveles de experiencia, hp, mana, ataque, agilidad, etc. Así como las magias, coste de estas, cuales hemos desbloqueado, etc.

```

public class Stats_jugador : Stats_personaje
{
    public static int nivel = 1;
    public static int experiencia;
    public static int[] exp_niveles = {0,7,16,24,63,110,230,350,500,700,900,1100,1500,
                                        2000,2500,3000,3000,3000,3000,4000,4000,4000,
                                        4000,4000,4000,4000,4000,4000,3535};
    public static int[] niveles_maxHP = {18,25,31,37,42,49,54,60,66,73,79,85,92,97,99,104,
                                        113,120,126,133,140,147,155,164,172,185,197,207,220,245};
    public static int[] niveles_maxMP = {0,0,5,16,20,24,26,29,36,40,50,58,64,70,72,95,100,108,115,
                                        128,135,146,153,161,165,168,175,182,199,230};
    public static int[] niveles_ataque = {3,4,6,6,10,14,16,19,27,31,35,42,48,54,61,64,70,75,92};
    public static int[] niveles_agilidad = {4,4,6,8,10,10,17,20,22,31,35,40,48,55,64,70,78,84,86,88,90,90,
                                        94,98,100,105,107,115,120,130};
    public static int[] niveles_magia = {0,0,6,14,22,22,22,32,36,41,45,53,60,60,65,75,82,100,100,
                                        100,100,100,100,120,140,160,180,200,220,230 };

    public static int[] costeMagias = { 4, 5 , 4, 5, 25, 20};
    public static int magiasDesbloqueadas = 0;

    public static string[] descripcion_magias = {"Cura al menos 15 PH","Prende Fuego al enemigo e inflinge en
                                                "Previene al enemigo de utilizar conjuros.", "Cura una gran c

    @ Mensaje de Unity | 0 referencias
    private void Awake()...

    @ Mensaje de Unity | 0 referencias
    private void OnApplicationQuit()...

    14 referencias
    public static bool addExp(int exp)...

    2 referencias
    public static void equiparObjeto(Item nuevo, Item antiguo = null)...

    1 referencia
    public static void desequiparObjeto(Item item)...
}

```

Ilustración 3.280: Clase Stats_jugador

Como vemos en la imagen es bastante información extra. Pero no necesitamos mayor explicación, simplemente son los valores con los que irán escalando nuestros Stats a medida que subamos de nivel. Aunque lo que sí tenemos que saber es que el nivel máximo en este juego es el nivel 30. Ya que no es un juego excesivamente largo (quizás en torno a las 10 horas de juego) no necesitamos más niveles. Hay que mencionar también que hemos cambiado todos los valores originales (excepto los de experiencia) y añadido nuevos atributos que son la defensa mágica (que escala según el ataque mágico) y el ataque mágico.

Hemos añadido también funciones para añadir experiencia al acabar con un enemigo, y funciones para equipar y desequipar objetos que básicamente aumentan o disminuyen nuestras estadísticas según el objeto en cuestión. Estas funciones de equipar o desequipar objetos no son interesantes, pero la de añadir experiencia sí que lo es y la mostramos a continuación:

```
public static bool addExp(int exp)
{
    if (nivel == 30)
    {
        experiencia = 0;
        return false;
    }
    else
    {
        experiencia += exp;
        if (experiencia >= exp_niveles[nivel])
        {
            while(experiencia >= exp_niveles[nivel])
            {
                experiencia -= exp_niveles[nivel];
                vida_maxima = niveles_maxHP[nivel];
                mana_maximo = niveles_maxMP[nivel];
                agilidad.set_valor(niveles_agilidad[nivel], niveles_agilidad[nivel - 1]);
                magia.set_valor(niveles_magia[nivel], niveles_magia[nivel - 1]);
                def.magica.set_valor(Mathf.RoundToInt((niveles_magia[nivel] * 0.7f)), Mathf.RoundToInt(niveles_magia[nivel - 1] * 0.7f));
                if (nivel <= 19)
                {
                    ataque.set_valor(niveles_ataque[nivel], niveles_ataque[nivel - 1]);
                }
                nivel += 1;
                if (nivel == 3 || nivel == 4 || nivel == 7 || nivel == 10 || nivel == 17 || nivel == 19)
                {
                    magiasDesbloqueadas += 1;
                }
            }
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

Ilustración 3.281: addExp

Lo primero que hacemos es añadir experiencia, y posteriormente comprobamos si tenemos suficiente para subir de nivel, en cuyo caso mientras tengamos suficiente experiencia aumentamos niveles y estadísticas. Algo que deberíamos saber es que el ataque solo aumenta hasta el nivel 19, ya que, entre las armas, semillas de fuerza que podamos encontrar y demás, podría acabar desbalanceado el personaje. De igual modo, es interesante saber que la defensa no aumenta nunca. Solo consumiendo semillas y equipando objetos podremos aumentar nuestra defensa, también para balancear el juego. La función devuelve true o false para que el mánager de combate sepa cuando se sube de nivel, pero de eso hablaremos luego en mayor profundidad.

3.4.3 Mánager Menú Pausa actualizado

Ahora que ya hemos explicado el funcionamiento de nuestros Stats, podemos comenzar a hablar del mánager de nuestro menú de pausa. Anteriormente se encontraba incompleto porque no contábamos con la existencia de los objetos ni de las estadísticas del jugador, pero ahora que disponemos de ellas podemos completarlo.

3.4.3.1 Inventario

Para la parte de inventario de nuestro menú de pausa hemos utilizado un código muy similar al de la tienda. La forma de distribuir y mostrar los objetos es la

misma salvo porque en los objetos equipados aparece una letra mayúscula 'E' que nos lo indica, y porque no aparece el precio de los objetos.



Ilustración 3.282: Interfaz Inventario Completa

Si tenemos que destacar un cambio, es que en el inventario podemos elegir entre varias opciones según el tipo de objeto que sea, son equipar/desequipar, usar, desechar.

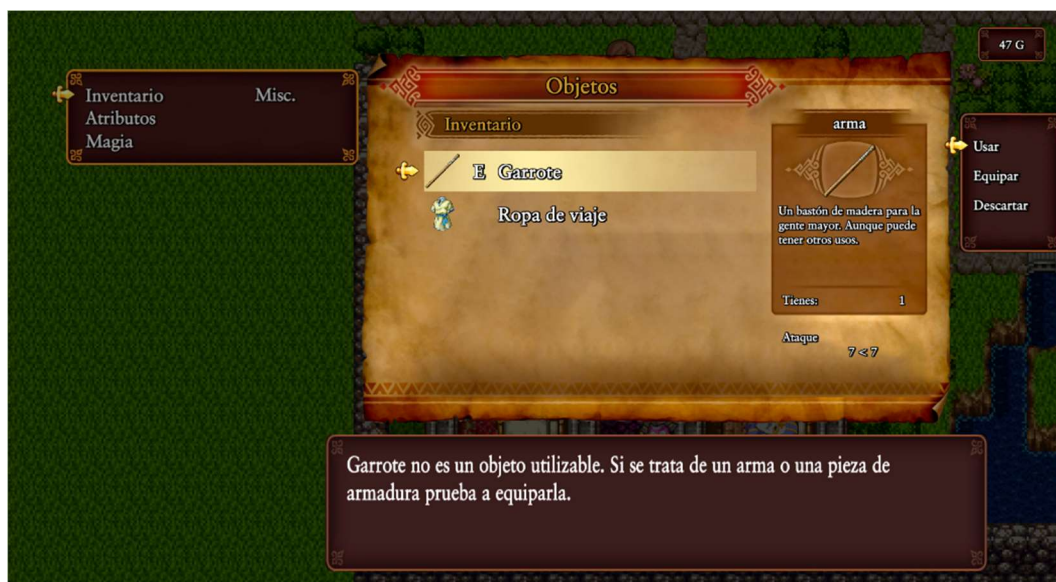


Ilustración 3.283: Usar, Equipar, Desechar objetos

El código es el mismo que el utilizado en todos los menús de selección hasta el momento, ya sea el menú de inicio, como el de selección en la tienda, por tanto, no consideramos necesario mostrar código, ya que es simplemente código reutilizado y adaptado para que funcione correctamente en nuestro inventario. Aunque sí mostraremos código de las funciones para equipar y desequipar añadidas a la clase Inventario:

```
public static void equipar(int seleccionado)
{
    int pos = -1;
    for(int i = 0; i < num_items; i++)
    {
        if (inventario[i][0].tipoObjeto == inventario[seleccionado][0].tipoObjeto && i != seleccionado && inventario[i][0].equipado == true)
        {
            pos = i;
            break;
        }
    }
    if(pos == -1)
    {
        Stats_jugador.equiparObjeto(inventario[seleccionado][0]);
        inventario[seleccionado][0].equipado = true;
    }
    else
    {
        Stats_jugador.equiparObjeto(inventario[seleccionado][0], inventario[pos][0]);
        inventario[seleccionado][0].equipado = true;
        inventario[pos][0].equipado = false;
    }
    switch (inventario[seleccionado][0].tipoObjeto)...
```

Ilustración 3.284: equipar

En equipar lo que hacemos es, primeramente, comprobar si ya tenemos un objeto equipado, en cuyo caso pasamos junto al nuevo a equipar por la función **equiparObjeto** de nuestro Script Stats_jugador, que añadirá el nuevo valor, y restará el antiguo, para que solo permanezca la diferencia entre equipamiento, ya sea para bien, o para mal. Si no se pasa otro objeto, como sería el caso donde no tenemos nada equipado, no se resta ningún valor, solo se suma. Además de esto, marcamos el atributo equipado del objeto como verdadero, y el del antiguo como falso (si hubiese antiguo).

Desequipar es mucho más sencillo y solo requiere de la llamada al método **desequiparObjeto**, pasándole el objeto a desequipar. En ese caso se restará los stats que proporcione dicho objeto y se marcará como false el atributo equipado del objeto.

3.4.3.2 Atributos

Pasamos ahora al apartado de atributos. En los atributos lo único que tenemos que hacer es acceder a la clase Stats_jugador y obtener los valores para todos los

atributos de nuestro jugador, así como calcular la diferencia entre la experiencia para pasar al siguiente nivel y la actual para mostrar la experiencia restante hasta el próximo nivel. Todo lo demás como hemos dicho es simplemente pasar los datos a las variables de Texto de la pantalla. Lo único que cambia son los Sliders y la parte inferior izquierda.

```
SetMaxHP(Stats_jugador.vida_maxima);
SetHP(Stats_jugador.vida_actual);
SetMaxMP(Stats_jugador.mana_maximo);
SetMP(Stats_jugador.mana_actual);
max_hp.text = Stats_jugador.vida_maxima.ToString();
max_mp.text = Stats_jugador.mana_maximo.ToString();
ataque.text = Stats_jugador.ataque.get_valor().ToString();
def_magica.text = Stats_jugador.def_magica.get_valor().ToString();
defensa.text = Stats_jugador.defensa.get_valor().ToString();
agilidad.text = Stats_jugador.agilidad.get_valor().ToString();
magia.text = Stats_jugador.magia.get_valor().ToString();
nivel.text = "Lv. " + Stats_jugador.nivel.ToString();
exp.text = Stats_jugador.experiencia.ToString();
mp_actual.text = Stats_jugador.mana_actual.ToString();
hp_actual.text = Stats_jugador.vida_actual.ToString();
```

Ilustración 3.285: Sliders y atributos

A los Slider hay que asignarles un valor máximo, que en nuestro caso sería la vida_maxima y el mana_maximo, y un valor, que sería nuestra vida_actual y nuestro mana_actual.

En la parte inferior izquierda se muestra el equipo actual, pero además de eso, también se muestra el tipo de objeto que llevamos equipado, es decir, si llevamos un garrote se mostrará como una lanza, si llevamos una espada se mostrará como tal, etc.

```
else
{
    if(items[0].tipoObjeto == tipoObjeto.arma && items[0].equipado == true)
    {
        arma.text = items[0].nombre;
        if(items[0].nombre == "Daga de cobre")
        {
            arma.transform.parent.GetComponent<Image>().sprite = daga;
        }
        else if(items[0].nombre == "Espadón")
        {
            arma.transform.parent.GetComponent<Image>().sprite = espadon;
        }
        else if(items[0].nombre == "Garrote")
        {
            arma.transform.parent.GetComponent<Image>().sprite = lanza;
        }
        else if(items[0].nombre == "Hacha")
        {
            arma.transform.parent.GetComponent<Image>().sprite = hacha;
        }
        else
        {
            arma.transform.parent.GetComponent<Image>().sprite = espada;
        }
    }
    else if (items[0].tipoObjeto == tipoObjeto.armadura && items[0].equipado == true)
    else if (items[0].tipoObjeto == tipoObjeto.casco && items[0].equipado == true)
    else if (items[0].tipoObjeto == tipoObjeto.escudo && items[0].equipado == true)
    else if (items[0].tipoObjeto == tipoObjeto.accesorio && items[0].equipado == true)
    {
        accesorio.text = items[0].nombre;
        if (items[0].nombre.Contains("Collar"))
        {
            accesorio.transform.parent.GetComponent<Image>().sprite = collar;
        }
        else
        {
            accesorio.transform.parent.GetComponent<Image>().sprite = anillo;
        }
    }
}
```

Ilustración 3.286: Modificación de Sprites según equipamiento



Ilustración 3.287: Atributos con Garrote equipado



Ilustración 3.288: Atributos con Garrote desequipado

3.4.3.3 Magia

En la parte de magia tampoco tenemos demasiado trabajo, ya que solamente son seis opciones, y podremos seleccionar unas u otras según el nivel de magias que hayamos desbloqueado. No merece la pena enseñar código pues solo se trata del manejo de la interfaz para no permitir utilizar magias o bien de combate, o bien no desbloqueadas. Cuando decidimos utilizar algún hechizo de curación (que son los permitidos fuera de combate) comprobamos cuál de ellos es, y ejecutamos el hechizo a la vez que sustraemos el maná necesario. Y, por supuesto, antes de hacer nada, comprobamos que se tenga el suficiente maná.


```

else if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()) && timer_boton > tiempo_espera_boton)
{
    if (Stats_jugador.mana_actual < Stats_jugador.costeMagias[seleccionado_magias])
    {
        StartCoroutine(animarFrase("No tienes suficiente mana para utilizar el conjuro.", menu_pausa.transform.Find("Canvas").trans
    }
    else
    {
        menu_pausa.transform.Find("Canvas").transform.Find("Magia").gameObject.SetActive(false);
        if(seleccionado_magias == 0)
        {
            int a = Random.Range(15, 20);
            Stats_jugador.RecibirCuracion(a);
            Stats_jugador.mana_actual -= Stats_jugador.costeMagias[0];
        }
        else if (seleccionado_magias == 4)
        {
            int a = Random.Range(80, 110);
            Stats_jugador.RecibirCuracion(a);
            Stats_jugador.mana_actual -= Stats_jugador.costeMagias[4];
        }
        else
        {
        }
    }
    timer_boton = 0;
}

```

Ilustración 3.289: Hechizos fuera de combate



Ilustración 3.290: Hechizos bloqueados

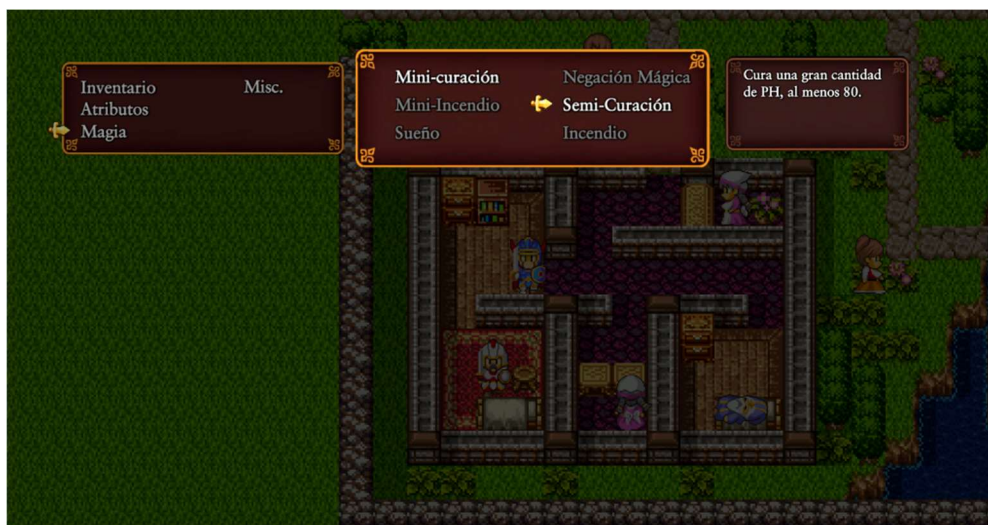
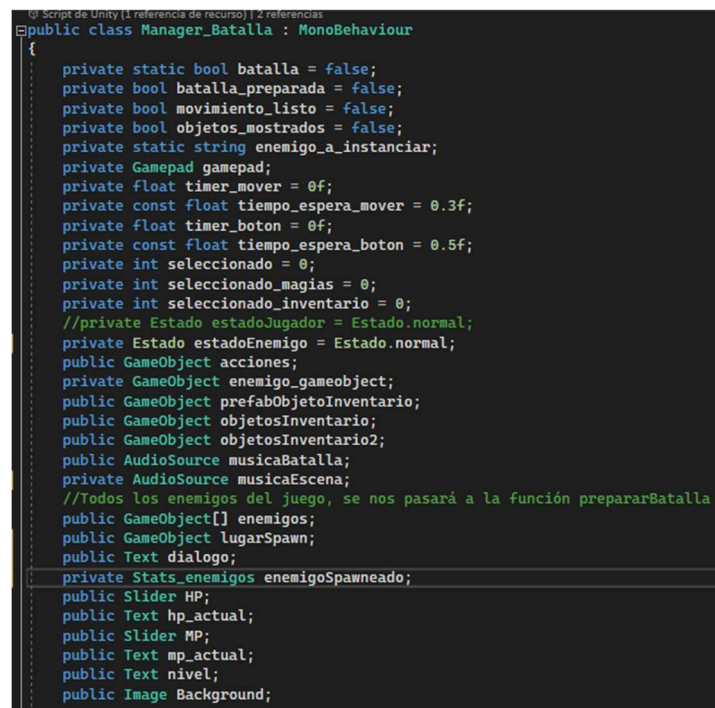


Ilustración 3.291: Todos los hechizos desbloqueados

Esta es toda la funcionalidad nueva añadida a nuestro mánager del menú de pausa, ya que, si recordamos, la parte del guardado ya la añadimos en el sprint anterior.

3.4.4 Mánager de Batalla

Ahora veremos el mánager de batalla, este script lo portará la propia interfaz de batalla y solo se ejecutará bajo ciertas condiciones como ya veremos.



```
Script de Unity (1 referencia de recurso) | 2 referencias
public class Manager_Batalla : MonoBehaviour
{
    private static bool batalla = false;
    private bool batalla_preparada = false;
    private bool movimiento_listo = false;
    private bool objetos_mostrados = false;
    private static string enemigo_a_instanciar;
    private Gamepad gamepad;
    private float timer_mover = 0f;
    private const float tiempo_espera_mover = 0.3f;
    private float timer_boton = 0f;
    private const float tiempo_espera_boton = 0.5f;
    private int seleccionado = 0;
    private int seleccionado_magias = 0;
    private int seleccionado_inventario = 0;
    //private Estado estadoJugador = Estado.normal;
    private Estado estadoEnemigo = Estado.normal;
    public GameObject acciones;
    private GameObject enemigo_gameobject;
    public GameObject prefabObjetoInventario;
    public GameObject objetosInventario;
    public GameObject objetosInventario2;
    public AudioSource musicaBatalla;
    private AudioSource musicaEscena;
    //Todos los enemigos del juego, se nos pasará a la función prepararBatalla
    public GameObject[] enemigos;
    public GameObject lugarSpawn;
    public Text dialogo;
    private Stats_enemigos enemigoSpawnado;
    public Slider HP;
    public Text hp_actual;
    public Slider MP;
    public Text mp_actual;
    public Text nivel;
    public Image Background;
```

Ilustración 3.292: Atributos Manager_Batalla

Como vemos tenemos una gran cantidad de atributos que utilizaremos para manejar las batallas. El atributo batalla nos sirve para saber que estamos en una batalla y es lo que dará paso al método **Update** para ejecutar el código que deba. Tenemos también **batalla_preparada**, que estando a false hace que el método Update llame al método **prepararBatalla**, al cual se le pasará el enemigo que queremos instanciar. Pero, para instanciar el enemigo debemos saber contra qué enemigo vamos a pelear, para esto tenemos que recurrir al Script Movimiento, en el que debemos añadir ciertas funcionalidades.

```

public LayerMask bosque;
public LayerMask pradera;
public LayerMask pradera_oscura;
public LayerMask montania;
public LayerMask veneno;
public LayerMask arena;
public LayerMask area1;
private string[] monstruos_area1 = {"Limo", "Lima", "Dracanino"};
public LayerMask area2;
private string[] monstruos_area2 = {"Limo", "Lima", "Dracanino", "Fantasma"};
public LayerMask area3;
private string[] monstruos_area3 = {"Dracanino", "Dracólito", "Escorpión", "Fantasma"};
public LayerMask area4;
private string[] monstruos_area4 = {"Escorpión Férreo", "Asesino Óseo", "Limo"};
public LayerMask area5;
private string[] monstruos_area5 = {...};
public LayerMask area6;
private string[] monstruos_area6 = {...};
public LayerMask area7;
private string[] monstruos_area7 = {...};
public LayerMask area8;
private string[] monstruos_area8 = {...};
public LayerMask area9;
private string[] monstruos_area9 = {"Caballero Andante", "Dracanina", "Quil"};
public LayerMask area10;
private string[] monstruos_area10 = {"Dracanino", "Dracólito", "Escorpión"};
public LayerMask area11;
private string[] monstruos_area11 = {...};
public LayerMask area12;
private string[] monstruos_area12 = {"Caballero Aberrante", "Caballero Aberrante"};
public Sprite bosque_sprite;
public Sprite domdora;
public Sprite cueval;
public Sprite cueva2;
public Sprite castillo;
public Sprite bosque_tarde_sprite;
public Sprite bosque_noche_sprite;

```

Ilustración 3.293: Atributos nuevos Movimiento 1

```

public Sprite bosque_noche_sprite;
public Sprite bosque_amanecer_sprite;
public Sprite pradera_sprite;
public Sprite pradera_tarde_sprite;
public Sprite pradera_noche_sprite;
public Sprite pradera_amanecer_sprite;
public Sprite montania_sprite;
public Sprite montania_tarde_sprite;
public Sprite montania_noche_sprite;
public Sprite montania_amanecer_sprite;
public Sprite pradera_oscura_sprite;
public Sprite pradera_oscura_tarde_sprite;
public Sprite pradera_oscura_noche_sprite;
public Sprite pradera_oscura_amanecer_sprite;
public Sprite veneno_sprite;
public Sprite veneno_tarde_sprite;
public Sprite veneno_noche_sprite;
public Sprite veneno_amanecer_sprite;
public Sprite arena_sprite;
public Sprite arena_tarde_sprite;
public Sprite arena_noche_sprite;
public Sprite arena_amanecer_sprite;
public SpriteRenderer sprite_personaje;
private Animator flash_batalla;

```

Ilustración 3.294: Atributos nuevos Movimiento 2

Tenemos una cantidad de atributos nuevos muy grande, pero no tiene mucha complicación. Hemos creado TileMaps nuevos en las áreas que queremos que aparezcan enemigos, a estos TileMaps los hemos denominado Zona1, Zona2, etc. Y al igual que hicimos con los colliders, hemos creado unos colliders para cada una de estas zonas, de modo que si surge una batalla podamos saber en qué zona estamos y llamar a uno de los monstruos de esa zona, de ahí los arrays de string con nombres

de los monstruos, estos hacen referencia a los enemigos que aparecerán en cada una de las zonas. A parte de estos colliders para las zonas, también hemos añadido otros para detectar si estamos en el bosque, en la pradera, en las montañas, etc. Y poder cambiar el fondo de batalla. Para ello hemos utilizado los assets del juego DQ II, y hemos creado por nosotros mismos en Photoshop las versiones de amanecer, día, tarde y noche, para todos los lugares posibles del juego (a excepción de cuevas o castillos, donde no tiene sentido hacerlo), a excepción de algunas como los bosques, praderas, arena y montaña que ya venían en el juego (en los 4 modos) y que nos dio la idea de hacerlas para el resto también. Todos los sprites que vemos en la segunda imagen son estas imágenes que cambian según el momento del día en que juguemos al juego. Creemos que es buena idea incluirlo porque le da más inmersión al juego.



Ilustración 3.295: Fondo de Batalla – Castillo



Ilustración 3.296: Fondo de Batalla – Arena



Ilustración 3.297: Fondo de Batalla - Arena amanecer



Ilustración 3.298: Fondo de Batalla - Arena Noche

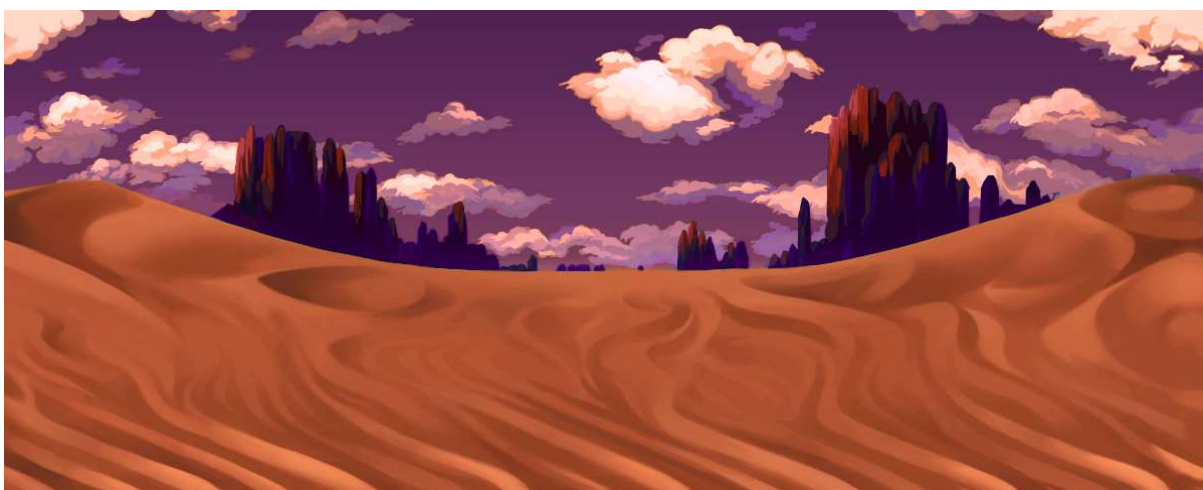


Ilustración 3.299: Fondo de Batalla - Arena Tarde

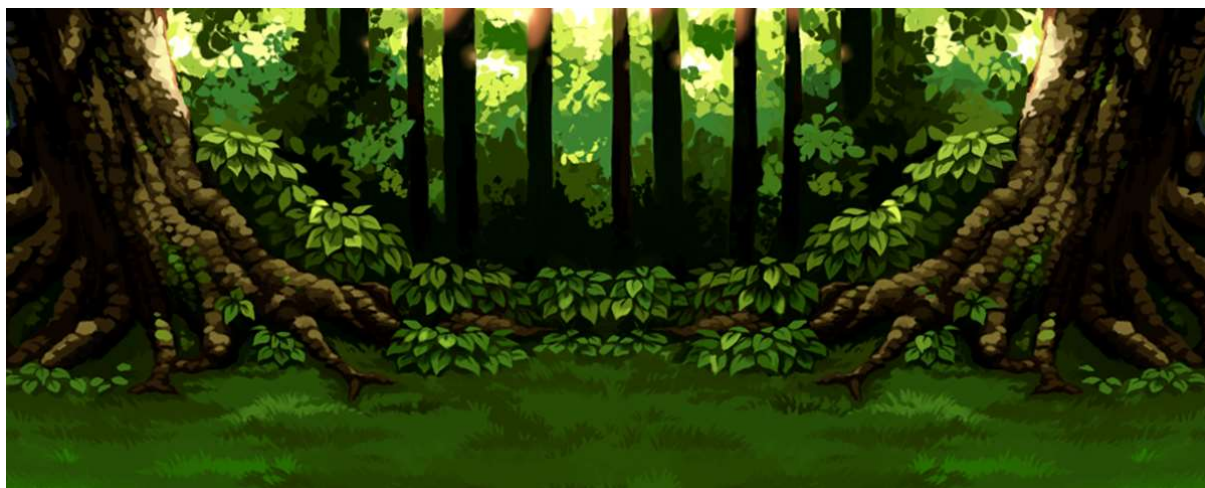


Ilustración 3.300: Fondo de Batalla – Bosque



Ilustración 3.301: Fondo de Batalla - Bosque Amanecer



Ilustración 3.302: Fondo de Batalla - Bosque Noche



Ilustración 3.303: Fondo de Batalla - Bosque Tarde



Ilustración 3.304: Fondo de Batalla – Cueva



Ilustración 3.305: Fondo de Batalla - Cueva 2



Ilustración 3.306: Fondo de Batalla - Ciudad Destruida de Domdora



Ilustración 3.307: Fondo de Batalla – Montañas

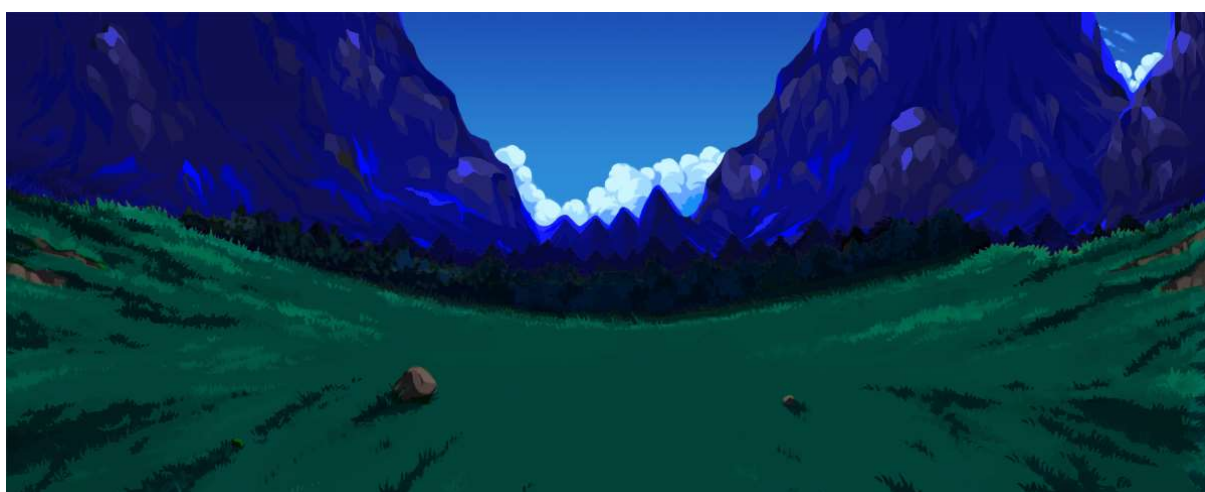


Ilustración 3.308: Fondo de Batalla - Montañas Amanecer



Ilustración 3.309: Fondo de Batalla - Montañas Noche

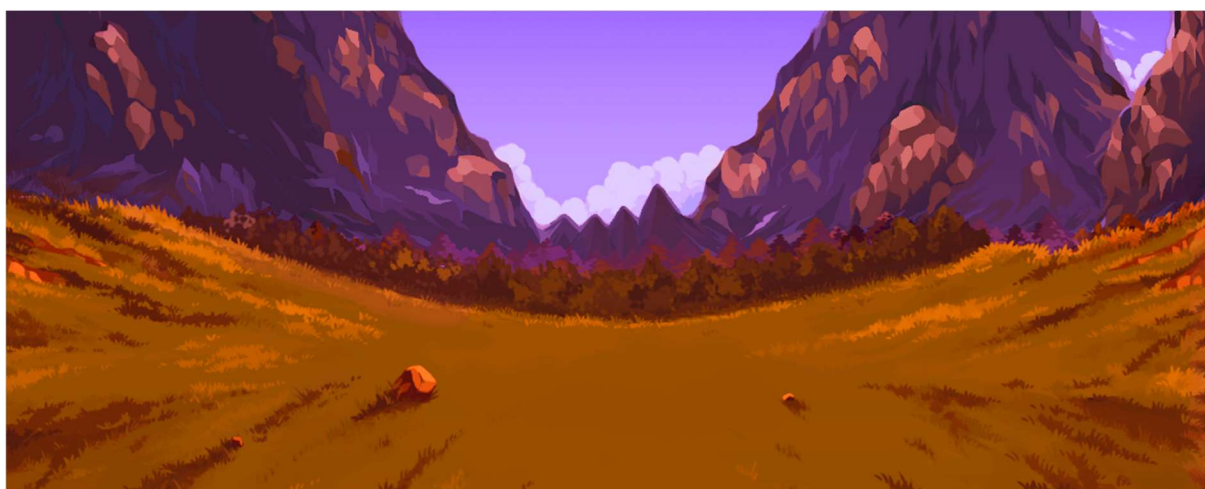


Ilustración 3.310: Fondo de Batalla - Montañas Tarde

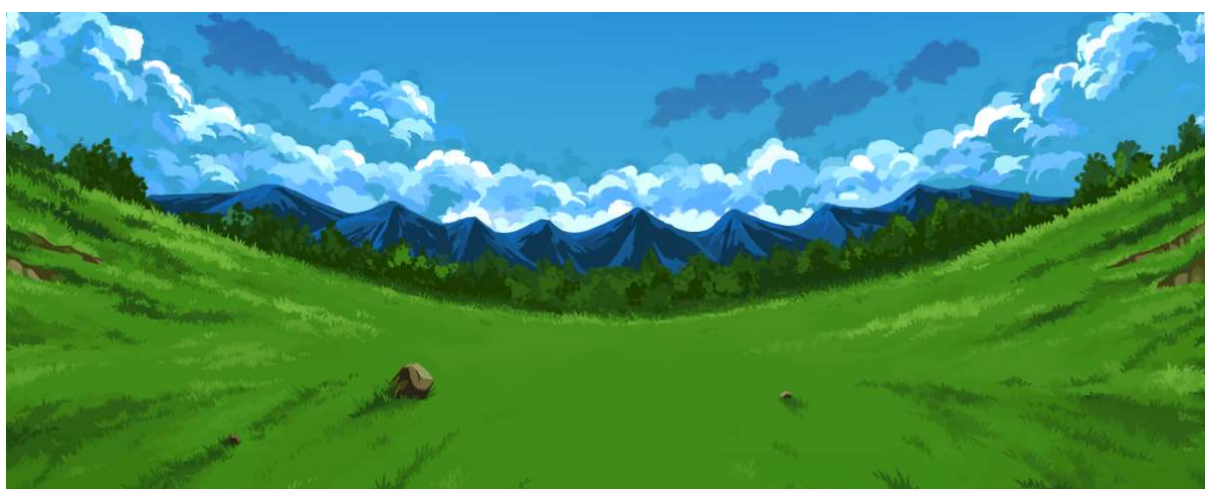


Ilustración 3.311: Fondo de Batalla – Pradera



Ilustración 3.312: Fondo de Batalla - Pradera Amanecer



Ilustración 3.313: Fondo de Batalla - Pradera Noche

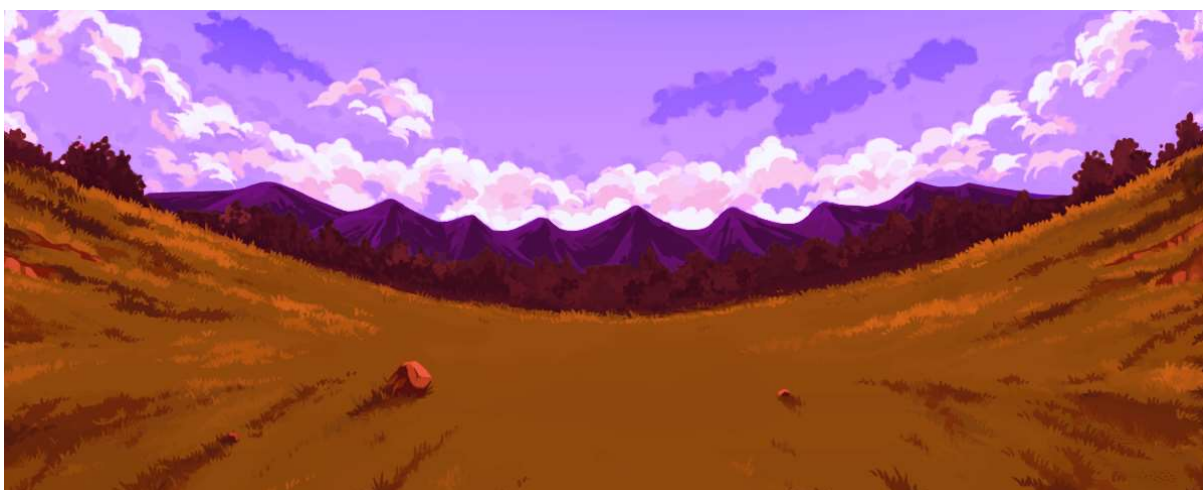


Ilustración 3.314: Fondo de Batalla - Pradera Tarde

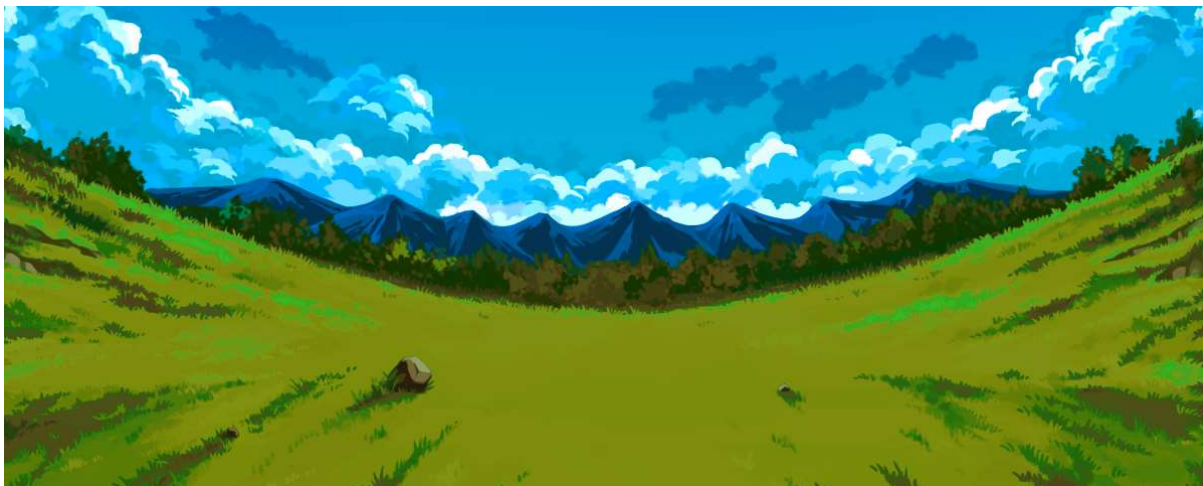


Ilustración 3.315: Fondo de Batalla - Pradera Oscura

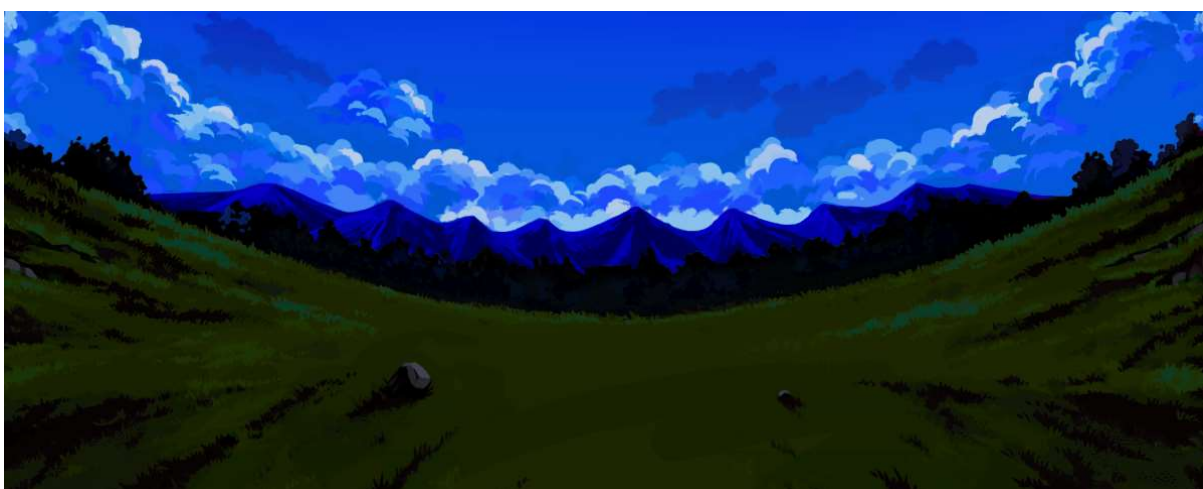


Ilustración 3.316: Fondo de Batalla - Pradera Oscura Amanecer



Ilustración 3.317: Fondo de Batalla - Pradera Oscura Noche



Ilustración 3.318: Fondo de Batalla - Pradera Oscura Tarde

```
if (!Physics2D.OverlapCircle(posicion.position + new Vector3(0.5f, 1, 0), .2f, colliders))
{
    posicion.position += new Vector3(0, 1, 0);
    timer = 0;
    movido = true;
    ultimo = ultimo_mov.arriba;
    Bloqueado();
    if(SceneManager.GetActiveScene().name == "Ciudad_Destruida_De_Domdora" && posicion.position == new Vector3(11, -17, 0))
    {
        bloqueado = true;
        StartCoroutine(BatallaDomdora());
    }
    else
    {
        if (5 > UnityEngine.Random.Range(0, 100) && bloqueado == false)
        {
            print("Inicio batalla");
            bloqueado = true;
            StartCoroutine(spawnBatalla());
        }
    }
}
```

Ilustración 3.319: Iniciar Batalla (?)

Pasamos ahora a hablar de los cambios en el código del script movimiento. Como vemos en la imagen de arriba, después de realizar un movimiento hacemos dos comprobaciones, la primera si nos encontramos en un sitio específico donde siempre queremos que ocurra una batalla (es un caso especial) y en la segunda, si no se da esta primera, tiramos un aleatorio, y con un 5 % de probabilidad empezará una batalla y llamaremos a **spawnBatalla**.

```

IEnumerator spawnBatalla()
{
    string aux = getEnemigo();
    if(aux == "")
    {
        bloqueado = false;
        yield return null;
    }
    else
    {
        flash_batalla = GameObject.Find("Flash").GetComponent<Animator>();
        flash_batalla.SetTrigger("Start");
        yield return new WaitForSeconds(0.1f);
        GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Batalla_Background").transform.Find("Background").GetComponent<Image>().enabled = false;
        GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Batalla_Background").transform.Find("Background").transform.Find("Background_border").GetComponent<Image>().enabled = false;
        GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Batalla_Background").gameObject.SetActive(true);
        flash_batalla = GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Batalla_Background").Find("Background").GetComponent<Animator>();
        flash_batalla.SetTrigger("Start");
        yield return new WaitForSeconds(1f);
        GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Batalla_Background").transform.Find("Background").GetComponent<Image>().enabled = true;
        GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Batalla_Background").transform.Find("Background").GetComponent<Image>().sprite = getBackground();
        GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Batalla_Background").transform.Find("Background").transform.Find("Background_border").GetComponent<Image>().enabled = true;
        GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Acciones").gameObject.SetActive(true);
        GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Estado").gameObject.SetActive(true);
        GameObject.Find("Manager_Batalla").transform.Find("Batalla").transform.Find("Eventos").gameObject.SetActive(true);
        Manager_Batalla.triggerBatalla(aux);
    }
}

```

Ilustración 3.320: spawnBatalla

Dentro de spawnBatalla lo primero que hacemos es llamar a la función **getEnemigo**. Esta función realiza una comprobación del mismo modo que lo hacemos en la ilustración 3.295, para comprobar si chocamos con un collider, solo que en este caso se hace para cada una de las áreas definidas en el juego, y en caso de estar en alguna (las ciudades al no tener este tipo de áreas son zonas seguras) se obtendrá un enemigo de forma aleatoria de las listas de enemigos que vimos antes en los atributos. Después de ya tener el nombre del enemigo que queremos enfrentar activamos una animación que hace que parpadee la pantalla en blanco, activamos la interfaz de batalla y justo seguido activamos otra animación que hará que la batalla aparezca desde el fondo de la pantalla hacia nosotros, posteriormente solo tendremos que activar aquellas opciones de la interfaz de batalla que no estén activadas y llamar a **triggerBatalla**, donde ya volveríamos al Script Manager_Batalla.

```

2 referencias
public static void triggerBatalla(string enemigo)
{
    enemigo_a_instanciar = enemigo;
    batalla = true;
}

```

Ilustración 3.321: triggerBatalla

En triggerBatalla asignaremos el enemigo y pondremos batalla a true. Después de esto, como ya veníamos diciendo al inicio de esta subsección, llamamos a

prepararBatalla que ultimar  los datos necesarios que queremos que aparezcan en la pantalla.

```
private void prepararBatalla(string enemigo)
{
    HP.maxValue = Stats_jugador.vida_maxima;
    HP.value = Stats_jugador.vida_actual;
    hp_actual.text = Stats_jugador.vida_actual.ToString();
    MP.maxValue = Stats_jugador.mana_maximo;
    MP.value = Stats_jugador.mana_actual;
    mp_actual.text = Stats_jugador.mana_actual.ToString();
    nivel.text = "Lv. " + Stats_jugador.nivel.ToString();
    if(Stats_jugador.vida_actual/Stats_jugador.vida_maxima <= 0.5)
    {
        if(Stats_jugador.vida_actual/Stats_jugador.vida_maxima <= 0.25)
        {
            hp_actual.color = Color.red;
            mp_actual.color = Color.red;
            nivel.color = Color.red;
        }
        else
        {
            hp_actual.color = Color.yellow;
            mp_actual.color = Color.yellow;
            nivel.color = Color.yellow;
        }
    }

    musicaEscena = GameObject.Find("Cancion").GetComponent();
    musicaEscena.Pause();
    musicaBatalla.Play();

    for (int i = 0; i < enemigos.Length; i++)
    {
        if(enemigos[i].GetComponent<Stats_enemigos>().nombre == enemigo)
        {
            enemigo_gameobject = Instantiate(enemigos[i], lugarSpawn.transform).gameObject;
            enemigo_gameobject.transform.position -= new Vector3(0, 20, 0);
            enemigoSpawneado = enemigo_gameobject.GetComponent<Stats_enemigos>();
            batalla_preparada = true;
            movimiento_listo = true;
            break;
        }
    }
}
```

Ilustraci n 3.322: prepararBatalla

Ya que en la interfaz de batalla tenemos Sliders asignamos sus valores para que muestren nuestra vida, mana, nivel. Pausamos la m sica e iniciamos la m sica de combate, que como ya dijimos es "Battle". Luego, miramos en la lista de todos los prefabs de enemigos y elegimos el que corresponda con el enemigo que obtuvimos anteriormente y poniendo a true tanto batalla preparada como **movimiento_listo**, que ser  el atributo que nos d  la se al para poder realizar alguna acci n en los turnos. En este punto ya tenemos todo listo para empezar el combate, pero antes de pasar a la explicaci n del combate vamos a ver los distintos enemigos que podemos enfrentar, aunque al igual que con los objetos, no vamos a mostrar toda su informaci n, solamente su nombre y una imagen. En total tenemos 41 enemigos diferentes que mostramos a continuaci n:



Ilustración 3.323: Limo

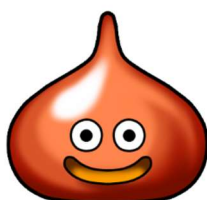


Ilustración 3.324: Lima



Ilustración 3.325: Dracanino



Ilustración 3.326: Fantasma



Ilustración 3.327: Prestidigitador



Ilustración 3.328: Dracólito

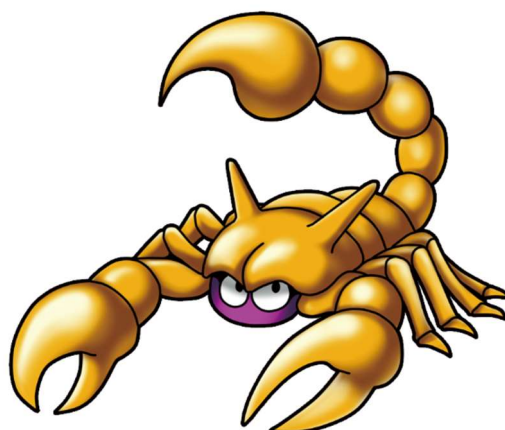


Ilustración 3.329: Escorpión



Ilustración 3.330: Esqueleto



Ilustración 3.331: Selenita



Ilustración 3.332: Espectro Pendenciero



Ilustración 3.333: Babosidad



Ilustración 3.334: Dracanina



Ilustración 3.335: Ilusionista



Ilustración 3.336: Lobobo



Ilustración 3.337: Escorpión Férreo

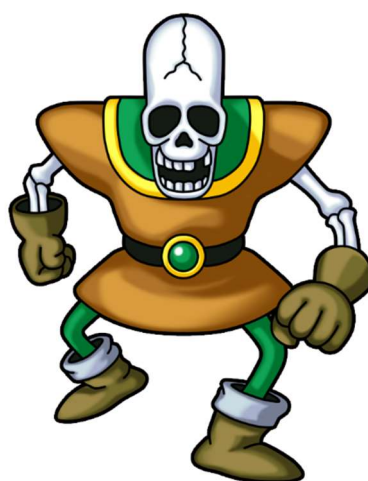


Ilustración 3.338: Asesino Óseo



Ilustración 3.339: Lobobo Espantajo

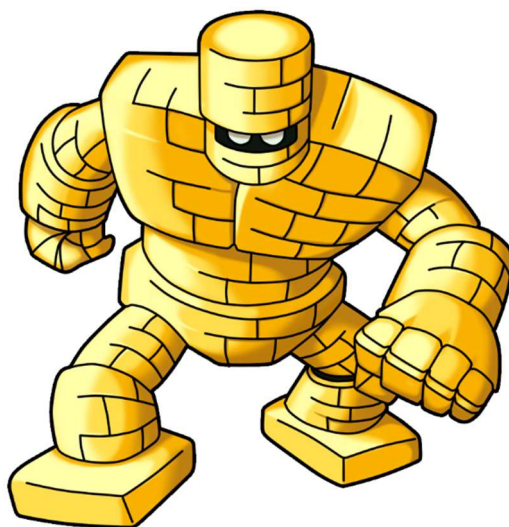


Ilustración 3.340: Gólem de Oro



Ilustración 3.341: Quimera



Ilustración 3.342: Espectro Infernal

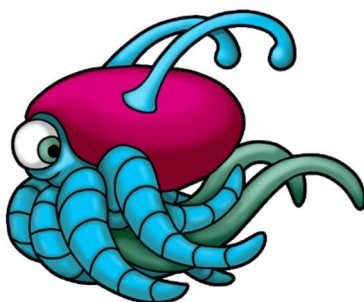


Ilustración 3.343: Selenita Lunático



Ilustración 3.344: Diablosidad



Ilustración 3.345: Esqueleto Soldado

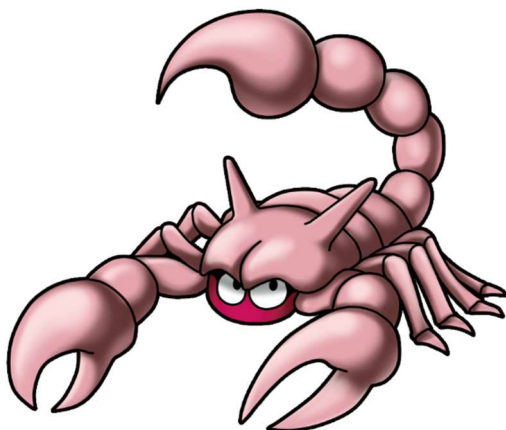


Ilustración 3.346: Escorpión Letal



Ilustración 3.347: Caballero Andante

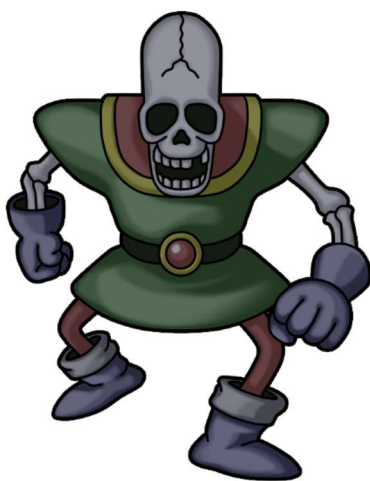


Ilustración 3.348: Esqueleto Oscuro



Ilustración 3.349: Quimera Engañosa



Ilustración 3.350: Limo Metálico



Ilustración 3.351: Lobobo Desgarrador



Ilustración 3.352: Quimera Cósmica



Ilustración 3.353: Dragón Verde



Ilustración 3.354: Mago Mayor

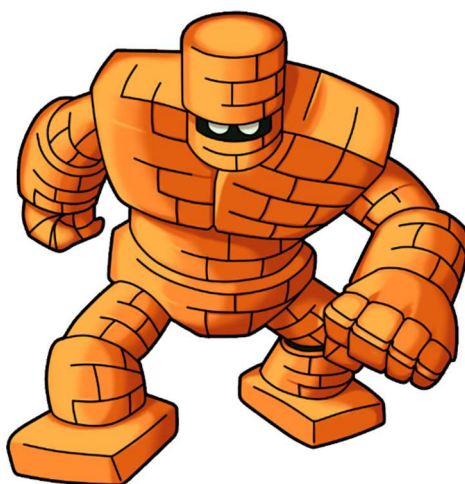


Ilustración 3.355: Gólem

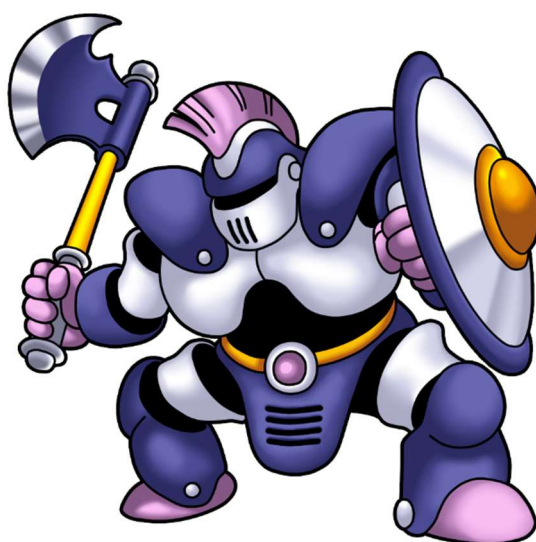


Ilustración 3.356: Caballero Aberrante



Ilustración 3.357: Dragón Azul

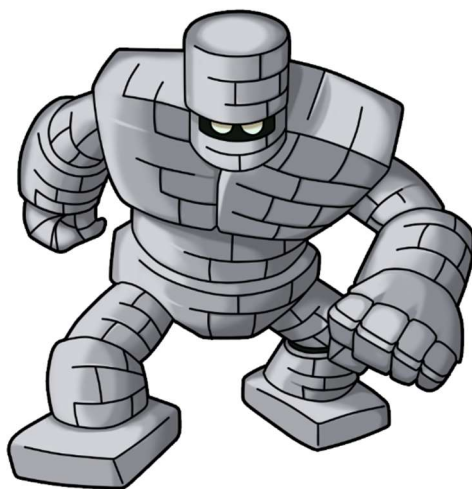


Ilustración 3.358: Gólem de Piedra



Ilustración 3.359: Caballero Abominable



Ilustración 3.360: Dragón Temido



Ilustración 3.361: Draconarius - Primera Forma



Ilustración 3.362: Draconarius - Segunda Forma

Ahora que ya hemos visto todos los enemigos que tenemos en nuestro juego podemos pasar a ver cómo funciona el combate.

```
if (batalla)
{
    if (!batalla_preparada)
    {
        acciones.transform.parent.Find("Eventos").transform.Find("Evento").GetComponent<Text>().text = "";
        prepararBatalla(enemigo_a_instanciar);
    }
    else
    {
        if (movimiento_listo)
        {
            if (acciones.transform.Find("Inventario").gameObject.activeInHierarchy)...
            else if (acciones.transform.Find("Magias").gameObject.activeInHierarchy)...
            else...
        }
    }
}
```

Ilustración 3.363: Código del Script "Manager_Batalla" 1

En la imagen superior se puede observar que mientras tengamos nuestro turno disponible se comprueba si tenemos abierto el inventario, luego si tenemos abierta las magias, y posteriormente, si no tenemos ninguna de estas abiertas. Por defecto, no tendremos nada abierto, observaremos la interfaz como la imagen que tenemos a continuación:



Ilustración 3.364: Interfaz de Batalla activa

En este estado, podemos tomar 4 opciones, atacar, objetos, magia, huir.

```

else if ((Input.GetKeyDown(KeyCode.Return) || gamepad.buttonSouth.IsPressed()) && timer_boton > tiempo_espera_boton)
{
    switch (seleccionado)
    {
        case 0:
            StartCoroutine(turnoAtaque());
            break;
        case 1:
            if (Inventario.Size() > 0)
            {
                List<Item>[] inventarioAux = Inventario.getInventario();
                foreach (List<Item> items in inventarioAux)
                {
                    if (items == null)
                    {
                        StartCoroutine(animarFrase("No tienes ningún objeto consumible en el inventario.));
                        break;
                    }
                    else
                    {
                        if (items[0].tipoObjeto == tipoObjeto.consumible)
                        {
                            acciones.transform.Find("Inventario").gameObject.SetActive(true);
                        }
                    }
                }
            }
            else
            {
                StartCoroutine(animarFrase("No tienes ningún objeto consumible en el inventario.));
                break;
            }
        case 2:
            if (Stats_jugador.magiasDesbloqueadas > 0)
            {
                acciones.transform.Find("Magias").gameObject.SetActive(true);
            }
            else
            {
                StartCoroutine(animarFrase("Aún no has desbloqueado ninguna habilidad mágica. Inténtalo una vez tengas al menos una.));
                break;
            }
        case 3:
            StartCoroutine(Huir());
            break;
    }
    timer_boton = 0;
}

```

Ilustración 3.365: Código del Script "Manager_Batalla" 2

Cuando pulsamos el botón del mando o Enter, se ejecutarán distintas acciones según la opción escogida, primero veremos objetos y magia.

Como se puede ver, en el caso de elegir objetos (el caso 1) primeramente se comprueba si tenemos objetos consumibles, ya que de lo contrario no podremos utilizar nada. Si sí tenemos objetos consumibles se nos abrirá una interfaz con los objetos que podemos utilizar.



Ilustración 3.366: Uso de objetos en combate

Aquí accederemos a la sección de inventario que veíamos antes en el código, y desde ahí controlaremos la elección del objeto, o pulsando el botón de volver atrás, cerrar esa ventana.

Si usamos un objeto, llamaremos a la función **UsarObjeto**, en la que nuestro personaje usará su objeto, se nos mostrará la información pertinente en el cuadro de diálogo y se nos actualizarán las barras de vida y mana arriba a la izquierda según correspondan.

```
IEnumerator UsarObjeto(Item item)
{
    StartCoroutine(animarFrase(Inventario.usarObjeto(Inventario.getPos(item))));
    acciones.transform.Find("Inventario").gameObject.SetActive(false);
    HP.value = Stats_jugador.vida_actual;
    MP.value = Stats_jugador.mana_actual;
    hp_actual.text = Stats_jugador.vida_actual.ToString();
    mp_actual.text = Stats_jugador.mana_actual.ToString();
    if (((float)Stats_jugador.vida_actual) / ((float)Stats_jugador.vida_maxima) <= 0.5)
    {
        if (((float)Stats_jugador.vida_actual) / ((float)Stats_jugador.vida_maxima) <= 0.25)
        {
            hp_actual.color = Color.red;
            mp_actual.color = Color.red;
            nivel.color = Color.red;
        }
        else
        {
            hp_actual.color = Color.yellow;
            mp_actual.color = Color.yellow;
            nivel.color = Color.yellow;
        }
    }
    yield return new WaitForSeconds(1f);
}
```

Ilustración 3.367: Código del Script "Manager_Batalla" 3

Seguido de esto, el enemigo efectuará su turno de ataque, ya que obviamente no va a permitir que nos curemos lo que queramos y que nos restauremos todo el maná. Primeramente, comprobamos si el enemigo está dormido, en cuyo caso tiramos un random (30% de probabilidad) y dependiendo del resultado despertamos o no al enemigo.

```
if (Random.Range(0f, 1f) < 0.3 && estadoEnemigo == Estado.dormido)
{
    estadoEnemigo = Estado.normal;
    StartCoroutine(animarFrase("El enemigo se ha despertado."));
    yield return new WaitForSeconds(1.5f);
}
if (estadoEnemigo == Estado.dormido)
{
    StartCoroutine(animarFrase("El enemigo está dormido..."));
    yield return new WaitForSeconds(1.5f);
}
```

Ilustración 3.368: Código del Script "Manager_Batalla" 4

Luego, si el enemigo está despierto o se ha despertado, realizará su turno de ataque, en el que tiraremos otro aleatorio para decidir si el golpe será o no será crítico (5 % de probabilidad).

Si es crítico se utilizará la fórmula de daño crítico que indicamos arriba, en caso contrario se utilizará la de daño normal. Una vez calculado el daño, lo mostramos por el cuadro de diálogo y efectuamos el daño al jugador, a su vez, comprobando si el golpe resulta fatal o no.

```

if (5 > Random.Range(0, 100))
{
    int dmg = (int)(1.45f * Mathf.Clamp( enemigoSpanneado.ataque.get_valor() - ((float)Stats_jugador.defensa.get_valor() / 2f), 1, 999) * Random.Range(1f, 1.1f));
    if (Stats_jugador.RecibirDanio(dmg, true))
    {
        SiguienteLinea("Golpe critico! Has recibido " + dmg + " (1 - (Mathf.Clamp(Stats_jugador.defensa.get_valor(), 0, 175)) / (75 + (Mathf.Clamp(Stats_jugador.de
        yield return new WaitForSeconds(2f));
        batallaPreparada = false;
        seleccionado = 0;
        seleccionado_magias = 0;
        seleccionado_inventario = 0;
        batalla = false;
        musicaBatalla.Stop();
        Destroy(enemigo_gameobject);
        Destroy(musicaEscena_gameobject);
        for (int i = 0; i < acciones.transform.parent.childCount; i++)
        {
            if (acciones.transform.parent.GetChild(i).gameObject.name == "Eventos")
            {
                acciones.transform.parent.GetChild(i).Find("Evento").GetComponent<Text>().text = "";
                acciones.transform.parent.GetChild(i).gameObject.SetActive(false);
            }
        }
        Inventario.dinero = (int)(Inventario.dinero * 0.9);
        Stats_jugador.vida_actual = Stats_jugador.vida_maxima;
        Stats_jugador.mana_actual = Stats_jugador.mana_maximo;
        StartCoroutine(AnimacionCambioEscena("World_Map"));
    }
    else
    {
        SiguienteLinea("Golpe critico! Has recibido " + dmg + " (1 - (Mathf.Clamp(Stats_jugador.defensa.get_valor(), 0, 175)) / (75 + (Mathf.Clamp(Stats_jugador.de
    }
}

```

Ilustración 3.369: Código del Script "Manager_Batalla" 5

Si resulta ser un golpe que nos debilite, se preparará el Manager_Batalla para terminar su modo actual, se eliminará el enemigo instanciado, se nos curará y se nos reducirá el dinero a un 90% del que teníamos en el momento de morir. Posteriormente se nos llevará al mapa del mundo, al último lugar que hubiésemos visitado. Si no nos debilita, volvemos a ejecutar otro turno.

Si decidimos utilizar una magia, al igual que con el inventario, se abrirá una ventana nueva en la interfaz. Esta ventana contendrá todos los hechizos, marcando en gris (y no dejando seleccionarlos) aquellos que no tengamos desbloqueados. Cuando elijamos un conjuro y decidamos ejecutarlo, primeramente, se comprobará el maná, pero si poseemos del suficiente, se llamará a la función **AtaqueMagico**.

```

IEnumerator AtaqueMagico(int seleccion)
{
    movimiento_listo=false;
    switch (seleccion) ...
    yield return null;
}

```

Ilustración 3.370: Código del Script "Manager_Batalla" 6

El método `AtaqueMagico` no es más que un switch con las distintas posibilidades de magia que podamos haber hecho. En caso de utilizar magias de curación o de efectos de estado, siempre atacaremos primero, para posteriormente recibir el golpe del enemigo, si al usar una magia como por ejemplo “Sueño” no lo hemos dormido. En caso contrario, si usamos una magia de ataque, se tendrá en cuenta la agilidad tanto del enemigo como la del jugador para determinar quién atacará primero. Es interesante saber que las fórmulas para daño mágico son diferentes a las fórmulas de daño físico, no solo porque utilicen el poder mágico, sino porque tienen daños mínimos y máximos dependiendo de la habilidad.

```

if (5 > Random.Range(0, 100))
{
    int dmg = (int)(1.1f * Mathf.Clamp(stats.jugador.magia.get_valor() - ((float)enemigoSpameado.def_magica.get_valor() / 5f), 60, 110));
    if (enemigoSpameado.RecibirDanioMagico(dmg))
    {
        SiguienteLinea("Golpe critico! El " + enemigoSpameado.nombre + " ha recibido " + dmg + " (1 - (Mathf.Clamp(enemigoSpameado.defensa.get_valor(), 0, 175)) / (75 + (Mathf.Clamp(enemigoSpameado.defensa.get_valor(), 0, 175)) / 75)) * " + dmg + " = " + dmg + " de daño mágico.");
        yield return new WaitForSeconds(2f);
        SiguienteLinea("Has recibido " + enemigoSpameado.experiencia + " puntos de experiencia.");
        yield return new WaitForSeconds(1.5f);
        SiguienteLinea("Has recibido " + enemigoSpameado.oro + " de oro.");
        inventario.dinero += enemigoSpameado.oro;
        yield return new WaitForSeconds(1.5f);
        if (stats.jugador.addExp(enemigoSpameado.experiencia))
        {
            SiguienteLinea("Has subido a nivel " + stats.jugador.nivel + "!");
            hp.maxValue = stats.jugador.vida_maxima;
            hp.value = stats.jugador.vida_actual;
            hp.actual.text = stats.jugador.vida_actual.ToString();
            mp.maxValue = stats.jugador.mana_maxima;
            mp.value = stats.jugador.mana_actual;
            mp.actual.text = stats.jugador.mana_actual.ToString();
            nivel.text = "Lv. " + stats.jugador.nivel.ToString();
            if (stats.jugador.vida_actual / stats.jugador.vida_maxima <= 0.5)
            {
                if (stats.jugador.vida_actual / stats.jugador.vida_maxima <= 0.25)
                {
                    hp.actual.color = Color.red;
                    mp.actual.color = Color.red;
                    nivel.color = Color.red;
                }
            }
        }
    }
}

```

Ilustración 3.371: Código del Script "Manager_Batalla" 7

Además de lo que acabamos de decir, también comprobamos si un enemigo ha muerto, en cuyo caso nos asignaremos experiencia, y en ocasiones subiremos de nivel indicándolo como se ve en la imagen.

Ahora hablaremos de las dos posibilidades que nos quedan. Habiendo visto cómo funciona el uso de objetos y el uso de magia, el uso de ataques básicos y de **Huir** es más fácil de explicar. En el caso de que escojamos huir por el motivo que sea, tendremos siempre, a excepción de los 2 jefes que tiene el juego, un 50% de probabilidades de escapar, en cuyo caso se terminará la batalla y podremos continuar moviéndonos a donde queramos, si no conseguimos escapar, el enemigo nos atacará, obviamente, pudiendo incapacitarnos. Si elegimos la opción de **Atacar**, golpearemos con el arma que tengamos equipada, y según cuál sea se verá una animación u otra en pantalla, por lo demás, sigue el mismo funcionamiento que el ataque mágico, pero con las fórmulas de cálculo de daño del ataque normal.

Esto sería todo lo que necesitamos saber de nuestro mánager de batalla, y con esto, ya hemos explicado todas las funcionalidades principales del juego. Tenemos un jugador que puede moverse libremente por el mundo, interactuar con los distintos tipos de NPCs, recibir objetos, encontrar objetos, comprar, vender, usar, tirar, equipar, desequipar, etc. Tenemos un menú de pausa donde revisar nuestro inventario, curarnos, ver nuestras estadísticas, así como guardar la partida y salir del juego. Un mánager de combate, que nos muestra distintos escenarios según el lugar del mundo del juego donde estemos y según la hora del día del mundo real, con distintos monstruos cada uno con sus fortalezas y debilidades.

3.5 Pruebas finales

En este apartado incluimos los diferentes testeos y pruebas que se han realizado para comprobar que todo funcione correctamente y que cumpla con lo que se le pide, tanto a nivel de que todo funcione correctamente, como a nivel de que haga lo que se espera.

3.5.1 Pruebas de verificación del sistema

En las pruebas de verificación del sistema hemos intentado forzar errores a aparecer para ver si funcionaba todo de forma correcta y en caso de no ser así, solucionarlo. Durante las pruebas hemos encontrado y solucionado los siguientes errores.

- Bug en el inventario al utilizar un objeto
- Bug al eliminar un objeto
- Bug visual al movernos hacia abajo en la tienda.
- Bug del sistema de Posada, donde te podías curar por encima del límite.
- Bug de objetos que hacía que al cerrar el juego y volver abrirlo el equipo apareciese desequipado, pero se mantuviesen los stats, pudiendo aumentarlos hasta el infinito de este modo.
- Animaciones de magias incorrectas.

- Error al cargar la posición cambiando entre pantallas en la versión “release” puesto que una parte del código de Movimiento que se necesitaba que se ejecutase antes que otra de todos los managers no lo hacía (solucionado pasando dicha parte del código al método Awake, que se ejecuta antes que Start).

Una vez solucionados estos errores no nos hemos encontrado con ningún otro, y hemos podido completar el juego sin mayor problema.

3.5.2 Pruebas de validación del sistema

En las pruebas de validación buscamos demostrar que nuestro sistema cumple con lo que se espera de él, es decir, que haga lo que debe, para ello podemos recurrir al apartado 1.6 de nuestra documentación, donde se define el alcance del proyecto. En este caso cumplimos con todo lo que se especifica que abarca nuestro proyecto:

- Podemos viajar por todo el mundo de Alefgard.
- Tenemos los 41 enemigos originales (con sus estadísticas actualizadas).
- Se incluyen las dos nuevas estadísticas Poder mágico y Defensa mágica.
- Versión mejorada del sistema de combate, pero manteniendo las opciones del original.
- Podemos hablar con todos los NPCs del juego.
- Podemos comprar en las diferentes tiendas del juego.
- Tenemos mayor número de objetos en el juego, lo que se traduce en mayor variedad en las tiendas.
- Añadimos la posibilidad de llevar una pieza más de armadura, el casco.
- El menú de pausa cumple con todo lo estipulado.
- El juego se encuentra traducido completamente al español, manteniendo la narrativa original.

4 CONCLUSIONES Y TRABAJOS FUTUROS

A lo largo de nuestro proyecto hemos intentado crear una versión mejorada del DQ original, y para ello nos hemos basado en otros juegos, y en versiones más nuevas de la saga, así como en remakes de esta. Consideramos que hemos conseguido un resultado bastante bueno ya que cumplimos con todos los requisitos que nos impusimos al inicio del proyecto con creces, añadiendo animaciones y haciendo todo mucho más entendible para el jugador, además de haber implementado mejoras sobre todo a nivel visual con la interfaz y de inmersión, como pueden ser todos los diferentes escenarios de batalla cambiando según la hora del día.

Si bien consideramos que hemos hecho un buen trabajo y que el resultado ha sido satisfactorio, hay cosas que podrían mejorarse a futuro, como, por ejemplo:

- Añadir en la interfaz de la tienda y el inventario algún elemento que nos permita saber de antemano si hay más objetos bajando hacia abajo.
- Permitir a los enemigos utilizar magia.
- Incluir más animaciones en los enemigos.
- Añadir sonidos al golpear contra un collider o al atacar dependiendo de qué tipo de ataque utilicemos.

Estas son solo algunas de las cosas que podríamos modificar, y que por superar el límite de tiempo no hemos podido incluir en el proyecto, por eso, si bien el proyecto está completo y es perfectamente funcional, algunos de estos cambios podrían mejorarlo, sobre todo a nivel de feedback de lo que está ocurriendo para el jugador.

5 APÉNDICES

En esta sección incluiremos documentación que ha sido relevante a la hora de definir el proyecto

5.1 Guía original del Trabajo Fin de Título

En esta subsección se incluye a continuación la guía original de TFG, así como la modificación para cambiarlo de TFG general a específico, pasando de llamarse “Remake de un videojuego clásico” a “Remake del videojuego Dragon Quest”.

5.1.1 Guía original

Este trabajo consiste en la realización de un remake de algún videojuego clásico utilizando un motor de videojuegos actual, como Unreal Engine, CryEngine o Unity. El videojuego original, así como el motor a utilizar, quedan a elección del alumnado.

Hay que describir las mecánicas de juego y la experiencia de usuario perseguida, incluyendo el sistema de recompensas, los mecanismos de progreso del usuario, los recursos gráficos y sonoros empleados, etc. Para la elaboración de escenas, personajes, NPCs, objetos, animaciones y todo tipo de material gráfico y sonoro, se podrán utilizar elementos descargables de la red, incluyendo sprites originales o cualquier otro recurso. Se recomienda elegir un videojuego de la década de los 80s y 90s, buscando la sencillez.

5.1.1.1 Conocimientos previos

Es necesario tener experiencia con el uso de motores de videojuegos.

5.1.1.2 Objetivos del TFG

- Diseño del remake del videojuego clásico elegido, especificando las características y el sistema/entorno objetivo.
- Descripción de las dinámicas, mecánicas y componentes de juego, que deberían ser las mismas del juego original. Se permite añadir contenido adicional o variaciones.

- Diseño de la interfaz de usuario y los mecanismos de interacción.
- Diseño del entorno virtual 2D/3D, incluyendo todo el material gráfico y sonoro.
- Desarrollo del juego propuesto, incluyendo la programación de los componentes necesarios.
- Realización de pruebas de evaluación del sistema y análisis de resultados.

5.1.1.3 Metodología a Desarrollar

- Describir el videojuego clásico elegido para realizar el remake. Describir las dinámicas, mecánicas y componentes de juego, lo que incluye narrativa, objetivos, recompensas, etc.
- Realizar un breve estado del arte sobre motores de videojuegos y seleccionar uno de ellos. Justificar las decisiones tomadas.
- Seleccionar y utilizar una metodología basada en Ingeniería del Software para el análisis de requisitos, diseño, implementación, prueba y documentación. Se recomienda utilizar una metodología incremental.
- Detallar la estimación temporal y costes de desarrollo.
- Realizar el desarrollo del remake, incluyendo el modelado del entorno virtual 2D/3D.
- Realizar pruebas para el prototipo y documentar los resultados.
- Redacción de la documentación final requerida.

5.1.1.4 Documentos y Formatos de entrega

- Documentación generada durante el proceso, de acuerdo con las buenas prácticas de la ingeniería del software. Esto se aplicará tanto en los proyectos de Ingeniería como en los trabajos teóricos o experimentales, es decir, siempre que se genere algún tipo de desarrollo, tanto software como webs, scripts, etc.
- Para el formato de documentación de los **Proyectos de Ingeniería** se utilizará la norma **UNE 157801**. Tal y como especifican las normas de estilo

de la EPSJ: *‘Los Proyectos de Ingeniería deberán presentar una estructura y contenido adaptados a la norma **UNE 157001** - Criterios generales para la elaboración de proyectos - u otra derivada de ésta, en función de la naturaleza del proyecto’*. *‘...para el caso de los sistemas de información informatizados (sistemas informáticos, sistemas de información geográfica, etc.) está la norma derivada de la anterior **UNE 157801** - Criterios generales para la elaboración de proyectos de sistemas de información’*. Para ello, se puede utilizar también la Norma CCII-N2016-02 (**Norma Técnica para la realización de la Documentación de Proyectos en Ingeniería Informática**) del Consejo General de Colegios Profesionales de Ingeniería en Informática, que incluye e implementa la norma **UNE 157801**.

- Memoria de trabajo, incluyendo manuales y anexos, en formato PDF.
- Código fuente completo y ejecutables del prototipo o software desarrollado.
- Documentos y archivos adicionales a los que se haga mención expresa en la memoria.
- Vídeo de demostración de la aplicación o de la experimentación realizada.
- Anexos para la presentación del trabajo:
 - Informe favorable del tutor.
 - Autorización de publicación, si procede.

5.1.2 Modificación del TFG

Se formalizó mediante la presentación de una solicitud al organismo correspondiente, la modificación del TFG “Remake de un videojuego clásico” para cambiar el título, y cambiar de carácter general a específico, siendo el nuevo título “Remake del videojuego Dragon Quest”.

5.2 Instalación y configuración del sistema

En cuanto a la instalación y configuración del sistema, saber que con tan solo ejecutar el .exe de la carpeta del ejecutable es suficiente. Si bien, el juego ha sido desarrollado enteramente para Windows, por lo que para su ejecución en otros sistemas pueda requerir del uso de emuladores o máquinas virtuales.

5.3 Manuales de usuario

En este apartado mostraremos de forma gráfica cómo jugar a nuestro juego.

Los controles son bastante similares a los de cualquier juego. Hemos creado con Photoshop un pequeño manual con los controles.

Controles

Podemos usar teclado o mando



- Para movernos utilizamos el **pad** o el **stick izq.**
- Para abrir el menú de pausa usamos **start**.
- Para movernos por los menús usamos el **pad**.
- Para interactuar con NPCs y realizar acciones usamos **A**. Para volver atrás usamos **B**.



- Para movernos utilizamos **W A S D**
- Para abrir y cerrar los menús usamos **esc.** (incluidos los de combate).
- Para interactuar con NPCs y realizar acciones usamos **Enter**



Ilustración 5.1: Controles Dragon Quest remake

6 DEFINICIONES Y ABREVIATURAS

En esta sección nos limitaremos a mencionar las abreviaturas utilizadas y a explicar cierta terminología utilizada durante la redacción de esta nuestra documentación:

- Dragon Quest → DQ
- Nintendo Entertainment System → NES
- Juego de Rol → RPG
- Banda Sonora Original → OST
- Pixelart → Forma de arte digital donde las imágenes se editan a nivel de píxeles.
- Electronics Arts → EA
- Unreal Engine → UE
- Epic Games → Epic
- Unreal Engine 5 → UE5
- Renderizado Dinámico de Alto Rango → HDRR
- Realidad Virtual → RV
- Realidad Aumentada → RA
- Juego de rol masivo en línea → MMORPG
- Non-Player Character → NPC
- Adobe Photoshop 2021 → Photoshop
- Kilo Líneas de Código → KLDC
- Esfuerzo → E
- Tiempo → T
- Personal → P
- Factor de Ajuste de Esfuerzo → FAE

7 BIBLIOGRAFÍA

- Alan, T. (2015). *Mastering Unity Scripting*. Packt.
- Andalucía, M. d. (s.f.). *juntadeandalucia*. Obtenido de juntadeandalucia: <https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416>
- Ashley, G., & Simon, J. (2016). *Mastering Unity 2D Game Development - Second Edition*. Packt.
- BOE. (s.f.). *Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública*. Obtenido de Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública: [https://www.boe.es/eli/es/res/2018/02/22/\(3\)/dof/spa/pdf](https://www.boe.es/eli/es/res/2018/02/22/(3)/dof/spa/pdf)
- Brackeys. (s.f.). *Brackeys Youtube*. Obtenido de Brackeys Youtube: <https://www.youtube.com/c/Brackeys>
- Cerrada, J. A. (2005). *Introducción a la Ingeniería del Software*. Centro de Estudios Ramón Areces.
- Database, G. U. (s.f.). *Game UI Database*. Obtenido de Game UI Database: <https://www.gameuidatabase.com/gameData.php?id=269>
- Enix, S. (s.f.). *SmashPedia*. Obtenido de SmashPedia: https://es.ssbwiki.com/wiki/Archivo:Armadura_de_Erdrick_DQII.png
- Itch.io. (30 de 08 de 2022). *Itch.io*. Obtenido de Itch.io: <https://itch.io/game-development/engines/most-projects>
- LordRoto. (s.f.). *Youtube*. Obtenido de Youtube: https://www.youtube.com/watch?v=qk-wzULTbC4&list=RDqk-wzULTbC4&start_radio=1&rv=qk-wzULTbC4&t=0
- Nicolas, A. B. (2021). *Hands-On Unity 2021 Game Development: Create, customize, and optimize your own professional games from scratch with Unity 2021, 2nd Edition*. Packt.
- Pressman, R. (2010). *Ingeniería del Software*. McGraw-Hill.
- Pressman, R. S. (2006). *Ingeniería del software: un enfoque práctico*. McGraw-Hill.
- Roto, L. (s.f.). *Youtube*. Obtenido de Youtube: <https://www.youtube.com/watch?v=STahp4WPCKE&list=PL2jLKwo6ZTmQ0vKgGwbcp5m2sp7vy9GXY&index=4>
- Roto, L. (s.f.). *Youtube*. Obtenido de Youtube: https://www.youtube.com/watch?v=ekx_Zzodvks&list=PL2jLKwo6ZTmQ0vKgGwbcp5m2sp7vy9GXY&index=3
- Roto, L. (s.f.). *Youtube*. Obtenido de Youtube: <https://www.youtube.com/watch?v=OBMdDHgtiuE&list=PL2jLKwo6ZTmQ0vKgGwbcp5m2sp7vy9GXY&index=6>

- Roto, L. (s.f.). *Youtube*. Obtenido de Youtube:
<https://www.youtube.com/watch?v=u073u6uGi18&list=PL2jLKwo6ZTmQ0vKgGwbcp5m2sp7vy9GXY&index=17>
- Roto, L. (s.f.). *Youtube*. Obtenido de Youtube:
<https://www.youtube.com/watch?v=smuQfg5r2ns&list=PL2jLKwo6ZTmQ0vKgGwbcp5m2sp7vy9GXY&index=16>
- Roto, L. (s.f.). *Youtube*. Obtenido de Youtube:
<https://www.youtube.com/watch?v=YtQVIlInfkCo>
- Roto, L. (s.f.). *Youtube*. Obtenido de Youtube:
<https://www.youtube.com/watch?v=FCWW0SRrR0I&list=PL2jLKwo6ZTmQ0vKgGwbcp5m2sp7vy9GXY&index=5>
- Sommerville, I. (2011). *Ingeniería del software*. Pearson Educación.
- Sugiyama, K. (s.f.). *Discogs*. Obtenido de Discogs:
<https://www.discogs.com/es/release/16029014-Kouichi-Sugiyama-%E4%BA%A4%E9%9F%BF%E7%B5%84%E6%9B%B2-%E3%83%89%E3%83%A9%E3%82%B4%E3%83%B3%E3%82%AF%E3%82%A8%E3%82%B9%E3%83%88I>
- TIGA. (2019, 08 12). *PR NewsWire*. Retrieved from PR NewsWire:
<https://www.prnewswire.com/news-releases/tiga-survey-reveals-that-unity-3d-engine-dominates-the-uk-third-party-engine-market-300900187.html>