



Escuela
Politécnica
Superior

Desarrollo de un videojuego multijugador cooperativo online en Unity 3D



Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:

Aitor Vidal Arnau

Tutor/es:

Miguel Ángel Lozano Ortega

Junio 2018



Universitat d'Alacant
Universidad de Alicante

“El hombre no deja de jugar porque se vuelve viejo, se vuelve viejo porque deja de jugar.”

- **George Bernard Shaw**

DEDICATORIA

Dedicado a todos aquellos interesados en el funcionamiento de un videojuego online, el manejo de la *API* de *Steam*, y la creación procedural de terrenos.

AGRADECIMIENTOS

Agradezco el apoyo brindado a lo largo de la realización del TFG por parte de mi familia, novia, amigos, y a mi tutor por confiar en este proyecto.

RESUMEN

El objetivo del proyecto es el desarrollo de un videojuego cooperativo de supervivencia usando el motor de videojuegos Unity 3D, el cual contará con un modo multijugador online.

Para ello, existen diferentes plataformas que nos permiten gestionar partidas multijugador y que, además, nos proporcionan una red social de jugadores. A lo largo de este proyecto se han estudiado las diversas plataformas existentes y se ha seleccionado e integrado la más adecuada para el videojuego en cuestión. Además, ninguna partida será igual puesto que en cada una se generará un mapa de juego de forma totalmente procedural.

Este videojuego recibirá el nombre de **Islands of Crafts**, en el cual los jugadores encarnarán a un muñeco de papel y deberán cooperar e intentar sobrevivir en una isla donde tanto la fauna como la flora serán elementos fabricados mediante materiales típicos de papelería como si de manualidades se trataran.

INDICE DE CONTENIDO

Tabla de contenido

1. INTRODUCCIÓN.....	18
2. MARCO TEÓRICO O ESTADO DEL ARTE.....	21
2.1 QUÉ ES UN VIDEOJUEGO Y SUS GÉNEROS.....	21
2.2 LOS VIDEOJUEGOS DE SUPERVIVENCIA	24
2.2.1 Dayz.....	25
2.2.2 Minecraft	26
2.2.3 Don't Starve	27
2.2.4 Rust	28
2.2.5 ARK: Survival Evolved.....	29
2.2.6 Terraria	30
2.2.7 The Legend of Zelda: Breath of the Wild	31
2.3 CONCLUSIÓN SOBRE EL ANÁLISIS DE LOS VIDEOJUEGOS DE SUPERVIVENCIA	32
3. OBJETIVOS	34
4. METODOLOGÍA	35
4.1 METODOLOGÍA DEL DESARROLLO	35
4.1.1 La metodología Kanban	35
4.2 GESTIÓN DEL PROYECTO	36
4.3 CONTROL DE VERSIONES Y REPOSITORIO	37
5. CUERPO DEL TRABAJO	38
5.1 ANÁLISIS Y ELECCIÓN DE HERRAMIENTAS.....	38
5.1.1 Motor de videojuegos	38
5.1.2 Software de modelado	43
5.1.3 Software de texturizado	47
5.1.4 Programas de dibujo y edición gráfica.	50
5.1.5 <i>Rigging, skinning</i> y animaciones.....	53
5.1.6 Sistemas de red para multijugador.	55
5.1.7 Plataformas de videojuegos digitales.....	59
5.1.8 Resumen de herramientas que se van a utilizar	62
5.2 DOCUMENTO DE DISEÑO DEL JUEGO (GDD)	63
5.2.1 Introducción.....	63
5.2.2 Mecánicas y jugabilidad.....	68

5.2.3	Interfaz.....	80
5.2.4	Cámara	83
5.2.5	Inteligencia Artificial	83
5.2.6	Música y efectos de sonido.....	84
5.2.7	Lenguaje.....	84
5.3	DESARROLLO E IMPLEMENTACIÓN DEL VIDEOJUEGO	85
5.3.1	Multijugador.....	93
5.3.2	Generación de un mundo procedural	111
5.3.3	Personaje principal	122
5.3.4	Otros personajes y elementos del mundo	128
5.3.5	Inventario y sistema de <i>crafting</i>	136
5.3.6	Eventos del mundo.....	148
5.3.7	Cámara	152
5.3.8	Sonido.....	153
5.3.9	Logo del videojuego	157
6.	CONCLUSIONES	158
7.	BIBLIOGRAFÍA Y REFERENCIAS	161

INDICE DE FIGURAS

Figura 1. Copia de tennis for two	22
Figura 2. Computer space.....	23
Figura 3. Pong.....	23
Figura 4. Dayz	25
Figura 5. Minecraft	26
Figura 6. Don't Starve	27
Figura 7. Rust.....	28
Figura 8. ARK: Survival Evolved	29
Figura 9. Terraria	30
Figura 10. The Legend of Zelda: Breath of the Wild.....	31
Figura 11. Metodología Kanban	35
Figura 12. Logo de Trello	36
Figura 13. Ejemplo de proyecto en Trello	37
Figura 14. Logo de GitHub	37
Figura 15. Logo de Unity.....	39
Figura 16. Logo de Unreal Engine.....	40
Figura 17. Programación basada en nodos en Unreal Engine.....	41
Figura 18. Logo cocos2d	41
Figura 19. Logo de Cryengine	42
Figura 20. Ejemplo proyecto de Unity3D	43
Figura 21. Ejemplo de creación de modelo 3D.....	44
Figura 22. Logo de Autodesk Maya	44
Figura 23. Logo de Autodesk 3ds max.....	45
Figura 24. Logo de Blender.....	46
Figura 25. Ejemplo proyecto 3Ds Max.....	47
Figura 26. Ejemplo de texturizado. 1 sin texturas, 2 con texturas	47
Figura 27. Logo de Autodesk Mudbox.....	48
Figura 28. Logo de Zbrush	49
Figura 29.Logo de Gimp	50
Figura 30. Logo de Photoshop.....	50
Figura 31. Logo de PaintTool SAI.....	51
Figura 32. Ejemplo de proyecto en Photoshop	52
Figura 33. Ejemplo de Proyecto en PaintTool SAI	52
Figura 34. Ejemplo de Rigging	53

Figura 35. Ejemplo de Skinning	54
Figura 36. Ejemplo de Animación 3D	54
Figura 37. Logo de mixamo	55
Figura 38. Cliente-Servidor en UNET	56
Figura 39. Photon Unity Network.....	56
Figura 40. Logo Photon Bolt	57
Figura 41. Steam.....	59
Figura 42. Logo de Steamworks	60
Figura 43. Logo de Origin	60
Figura 44. Logo de GooglePlay	60
Figura 45. Logo de Play Juegos.....	61
Figura 46. Logo de uPlay.....	61
Figura 47. Logo del videojuego.....	63
Figura 48. Logo de clasificación pegg.....	66
Figura 49. Etiqueta PEGI 3	66
Figura 50. Etiqueta PEGI Online	66
Figura 51. Imagen de Paper Mario: Color Splash	67
Figura 52. Ejemplo de muñeco de papel.....	69
Figura 53. Ejemplo de conejo de papel	69
Figura 54. Ejemplo de pollitos de cartón.....	69
Figura 55. Ejemplo de cerdo de plástico	70
Figura 56. Diferencias entre materiales	71
Figura 57. Texturizado en función de la altura del terreno.....	76
Figura 58. Ejemplo pino de papel.....	77
Figura 59. Ejemplo de palmeras de papel	77
Figura 60. Ejemplo flores de papel.....	78
Figura 61. Ejemplo fresno de cartón	78
Figura 62. Ejemplo arbusto de cartón	79
Figura 63. Transición Crear partida	80
Figura 64. Transición Unirse a partida.....	80
Figura 65. Transición Invitar amigo de Steam	81
Figura 66. Ejemplo de HUD muy similar	83
Figura 67. Momentos del día	84
Figura 68. Objetos de la escena Offline.....	85
Figura 69. Objetos de la escena Online.....	86
Figura 70. GameObject generados de manera automática	87

Figura 71. Jerarquía de objetos dentro del objeto Player	87
Figura 72. Ejemplo de jerarquía de objetos en un npc de vegetación.....	88
Figura 73. Ejemplo de jerarquía de objetos en un NPC de fauna.	88
Figura 74. Arquitectura GameObject Canvas	89
Figura 75. Casillas del inventario.....	90
Figura 76. Casillas del menú de fabricación	90
Figura 77. Objetos de GameOver	91
Figura 78. Jerarquía de objetos de CanvasSteamProfile	91
Figura 79. Jerarquía completa de objetos que componen LobbyManager	91
Figura 80. Objetos que perduran en el cambio de escena de offline a online.....	93
Figura 81. Tabla resumen de los objetos que aparecen en la escena offline y online.....	93
Figura 82. Acceso a los servicios online de Unity	94
Figura 83. Vinculación de la cuenta de Unity con el proyecto	94
Figura 84. Servicios de Unity Network	95
Figura 85. Opciones del apartado <i>Multiplayer</i> en el panel de servicios de Unet.....	95
Figura 86. Unity Dashboard.....	96
Figura 87. <i>Max players per room</i>	96
Figura 88. <i>Unity Network Live Mode</i>	97
Figura 89. Asset Network Lobby.....	98
Figura 90. Menú principal de Network Lobby.....	98
Figura 91. Script <i>LobbyManager</i>	99
Figura 92. Script <i>LobbyPlayer</i>	100
Figura 93. Listado de partidas en el Lobby	101
Figura 94. Menú de creación de partida	101
Figura 95. Ejemplo de cambio de colores y jugador preparado en el Lobby	102
Figura 96. Componente Network Identity	102
Figura 97. Ejemplo de SyncVar	103
Figura 98. Ejemplo de función Command	103
Figura 99. Ejemplo de función RpcClient	103
Figura 100. Componente network Transform.....	104
Figura 101. Componente Network Animator.....	105
Figura 102. Build Settings en Unity	106
Figura 103. Scripts del GameObject SteamManager	107
Figura 104. Script DisplayName	107
Figura 105. Ejemplo de Nick de Steam en el videojuego	108
Figura 106. Botón para invitar amigos de Steam	109

Figura 107. Overlay de Steam para invitar amigos.....	109
Figura 108. Ejemplo de invitación enviada.....	110
Figura 109. Ejemplo de invitación enviada lista para aceptar.....	110
Figura 110. Ejemplo de usuario invitado recientemente a la partida	110
Figura 111. Textura de Ruido generada con Perlin Noise	111
Figura 112. Textura de ruido generada con Perlin Noise, con forma cuadrada.....	112
Figura 113. Código del algoritmo Perlin Noise	112
Figura 114. Función GenerateHeights() encargada de asignar a los valores que devuelve el algoritmo Perlin Noise	113
Figura 115. Función que se encarga de generar el terreno finalmente	113
Figura 116. Script TerrainGeneration, para la generacion del terreno	114
Figura 117. Menú de texturas del terreno	115
Figura 118. Editar una textura del terreno.....	116
Figura 119. Script para pintar de forma automática el terreno, PaintTerrain2	117
Figura 120. Inicios de terreno procedural	118
Figura 121. Terreno procedural en forma de isla, y texturizado completamente	118
Figura 122. Texturas del terreno con mapas de normales.....	119
Figura 123. Creación del mar y su material/textura	120
Figura 124. Script WaveController, para configurar el oleaje del mar.....	120
Figura 125. Material del mar	121
Figura 126. Resultado del mar con el material creado. Reflexión de la luz del sol	121
Figura 127. Imágenes del proceso de modelado del personaje principal.....	122
Figura 128. Parámetros del material del personaje principal	123
Figura 129. Resultado del personaje principal con el material configurado anteriormente	123
Figura 130. Skinning del personaje principal.....	124
Figura 131. Rigging del personaje principal	124
Figura 132. Transición de animaciones del personaje principal	125
Figura 133. Parámetros de las animaciones del personaje principal	125
Figura 134. Componente Animator del personaje principal	125
Figura 135. Script PjControl.....	126
Figura 136. Collider en el brazo derecho del personaje principal.....	126
Figura 137. Script Hit.....	127
Figura 138. Personaje principal con medidores de vida y nivel de color, además de su nombre de Steam	127
Figura 139. Script health	127
Figura 140. Script ColorLevel.....	128

Figura 141. Script SetupLocalPlayer	128
Figura 142. Resultado del modelado y texturizado de los animales.....	129
Figura 143. Animales en proceso de modelado	129
Figura 144. Ejemplo de texturizado. Texturizando las tijeras animadas.....	130
Figura 145. Ejemplo de creación de animaciones en el editor de Unity	130
Figura 146. Estados y animaciones de los animales.....	131
Figura 147. Script AnimalIA	131
Figura 148. Mostrando el NavMesh sobre el terreno	132
Figura 149. Componente navMeshAgent.....	133
Figura 150. Script Drop.....	133
Figura 151. Script SetupAnimalPlant.....	133
Figura 152. Script AnimalAttack	134
Figura 153. Resultado del modelado y texturizado de la vegetación dentro del juego.....	135
Figura 154. modelado de la vegetación en proceso.....	136
Figura 155. Assets de los objetos del juego	137
Figura 156. Ejemplo de la configuración de un Asset de un objeto del juego	137
Figura 157. Diagrama de clases sobre la jerarquía de los ítems del juego.....	138
Figura 158. Script ActivePointLight	139
Figura 159. Resultado de la iluminación del cuerpo del personaje principal.....	139
Figura 160. Objetos en proceso de modelado	140
Figura 161. Script ColorLevelRefuge.....	141
Figura 162. Prefabs de objetos del juego	141
Figura 163. Script Inventory	142
Figura 164. Script ItemPickup.....	142
Figura 165. Inventario con 5 recursos iguales.....	143
Figura 166. Inventario con un objeto equipado en mano.....	143
Figura 167. Resultado de un objeto equipado en mano sobre el modelo del personaje principal	144
Figura 168. GameObject itemhand dentro de la jerarquía de objetos del prefab del jugador ...	144
Figura 169. Script Billboard	144
Figura 170. Ejemplo de espada y armadura equipada sobre el modelo del jugador.....	145
Figura 171. Ejemplo de espada deteriorada, con menos nivel de color que el inicial	145
Figura 172. Script InventorySlot	146
Figura 173. Menú de fabricación.....	146
Figura 174. GameObject EventSystem	147
Figura 175. GameObject EventManager	148
Figura 176. Script CraftingSlot.....	148

Figura 177. Script DayNightCycle	149
Figura 178. Código sobre la rotación del sol	149
Figura 179. Días transcurridos y hora dentro del juego.....	150
Figura 180. Ventana de GameOver	150
Figura 181. GameObject Rain, con sistema de partículas	151
Figura 182. Ejemplo de lluvia dentro del juego.....	151
Figura 183. Script CameraController	152
Figura 184. Script PostProcessingBehaviour	152
Figura 185. Configuración de los parámetros del Asset de postprocesado.....	153
Figura 186. Script AudioManager.....	154
Figura 187. Parámetros de un sonido	154
Figura 188. GameObject de los sonidos durante la ejecución	155
Figura 189. Componente AudioSource	155
Figura 190. Parámetros de las melodías del juego, y los sonidos ambientales nocturnos	157
Figura 191. Ejemplo de retrato hecho con cartón	157
Figura 192. Imagen del proyecto de Photoshop del logo	157

1. Introducción

Desde la aparición de los primeros videojuegos en la década de 1960 hasta la actualidad, la industria del videojuego ha experimentado una gran evolución.

Esta evolución va ligada de manera directa al progreso tecnológico, pues es un hecho que estos avances han permitido que se creen diferentes maneras de jugar a un videojuego (mandos con sensor de movimiento, cámaras con captura de movimiento, realidad aumentada, realidad virtual...), que los gráficos cada vez sean más realistas, que existan videojuegos online capaces de conectar a personas que viven en diferentes partes del mundo, que aparezcan videojuegos con fines diferentes más allá de entretener (como la enseñanza)...

La evolución es tal que el presupuesto para la creación de algunos videojuegos por parte de compañías grandes llega incluso a superar a algunas superproducciones cinematográficas.

Por otra parte, la competencia que existe hoy en día en esta industria ha permitido que, en tan solo 50 años, grandes mentes creativas de este mundillo hayan conseguido crear una enormísima variedad de videojuegos, todos perfectamente distinguibles por categorías o géneros.

Existe tal variedad de géneros que pueden ir desde juegos de plataformas o lucha hasta juegos de terror o incluso juegos eróticos, estando algunos de éstos dirigidos a un público en específico que puede estar determinado por un mínimo de edad.

A lo largo de este proyecto vamos a meternos de lleno en un género que ha conseguido tener su espacio en la industria de los videojuegos en estos últimos años. Estamos hablando de los **juegos de supervivencia** o *survival* en inglés.

Por regla general, en gran parte de los videojuegos uno de los requisitos fundamentales a la hora de completarlos es no morir en el transcurso de estos, y es en este punto donde los videojuegos de supervivencia encuentran el principal objetivo. Y es que en un juego de supervivencia la complejidad no está en intentar avanzar hasta un determinado punto en el juego, sino que ésta radica en que el jugador tenga un especial cuidado con el bienestar de su personaje, ya que por regla general en estos videojuegos los personajes son más frágiles de lo normal y el jugador debe de estar pendiente de algunas de sus necesidades básicas en todo momento, como, por ejemplo, el hambre o la sed.

Además, el entorno que rodea al jugador suele ser hostil, y fácilmente le puede jugar una mala pasada (no encontrar comida, fenómenos meteorológicos, fauna agresiva, la oscuridad de la noche...), de tal manera que la complejidad de estos videojuegos radica en intentar sobrevivir en un medio totalmente desconocido y hostil que podrás ir explorando y conociendo en el transcurso de la partida.

Este tipo de videojuegos se caracterizan también por contar con una gran variedad de mecánicas que aportan una gran libertad al jugador (como puede ser la creación de una cantidad enorme de diversos objetos con diferentes utilidades), y un entorno muy vivo que permite que el jugador se sumerja de lleno en el mundo de tal manera que el entorno le consiga sorprender en más de una ocasión.

De esta manera, este proyecto tendrá como objetivo la creación de un videojuego de supervivencia multijugador en línea de hasta 4 jugadores recibiendo el nombre de **Islands of Crafts**. Cabe destacar que, este videojuego, al ser creado de forma individual y sin ningún tipo de presupuesto, no contará con tal ambiciosa cantidad de mecánicas y ese entorno tan vivo y sorprendente que he mencionado anteriormente.

El documento de este proyecto estará organizado de la siguiente manera:

- **Marco teórico o Estado del Arte:** En este apartado se explicará el concepto de videojuego, un poco de historia sobre los videojuegos y se hará mención a algunos

de los géneros más populares en la actualidad. Además, se mencionarán algunos de los motores gráficos más populares de la actualidad, algunas de las herramientas empleadas a la hora de dotar a un videojuego de un modo en línea, y las plataformas sociales de jugadores más usadas en estos últimos años.

•**Objetivos:** En el apartado de Objetivos se analizarán algunos de los objetivos que se siguen con la elaboración de este proyecto. Además, se explicará qué herramientas de las mencionadas en el apartado anterior ha escogido y por qué.

•**Metodología:** Aquí se explicará qué tipo de metodología se ha llevado a cabo a lo largo del proyecto para la organización de las tareas y dónde se almacena el proyecto. Además, se hará mención a todas las herramientas y programas usados en el proyecto.

•**Cuerpo del trabajo:** En este apartado se incluirán Documento de Diseño del videojuego (GDD) y se concretará todas y cada una de las partes del desarrollo de éste.

•**Conclusiones:** Aquí se llevará a cabo un pequeño resumen sobre los objetivos conseguidos a lo largo de todo el proyecto, y se comentarán algunas formas de mejorar el proyecto.

•**Bibliografía y referencias:** Todos los enlaces sobre las fuentes consultadas a la hora de realizar el proyecto en cuestión.

2. Marco teórico o Estado del Arte

En este apartado explicaré en qué consiste la definición de videojuego, concretaré un poco sobre alguno de sus géneros y haré especial hincapié en el género de supervivencia. Además, haré referencia a algunos de los videojuegos de supervivencia en los que me he basado a la hora de la realización del proyecto.

Por otra parte, mencionaré algunos de los motores gráficos actuales más utilizados, qué herramientas se utilizan a la hora de dotar a un videojuego de un modo multijugador online, y las diferentes plataformas sociales que consiguen conectar a jugadores de todas partes del mundo.

2.1Qué es un videojuego y sus géneros

Si consultamos una de las enciclopedias más conocidas en la red, como Wikipedia, nos aportará la siguiente definición:

Un videojuego o juego de video es un juego electrónico en el que una o más personas interactúan, por medio de un controlador, con un dispositivo que muestra imágenes de vídeo. Este dispositivo electrónico, conocido genéricamente como «plataforma», puede ser una computadora, una máquina arcade, una videoconsola o un dispositivo portátil (un teléfono móvil, por ejemplo). Los videojuegos son, año por año, una de las principales industrias del arte y el entretenimiento.

Es una definición bastante básica y clara de lo que se podría considerar un videojuego.

Si echamos la vista atrás, nos encontramos con que los orígenes de los videojuegos se remontan a la década de 1950, con la creación de los que, podríamos denominar, los auténticos pioneros de este arte, como *Tennis for Two* (1958) o *Spacewar!* (1962),

aunque en este entonces apenas eran prototipos muy simples y de carácter experimental.

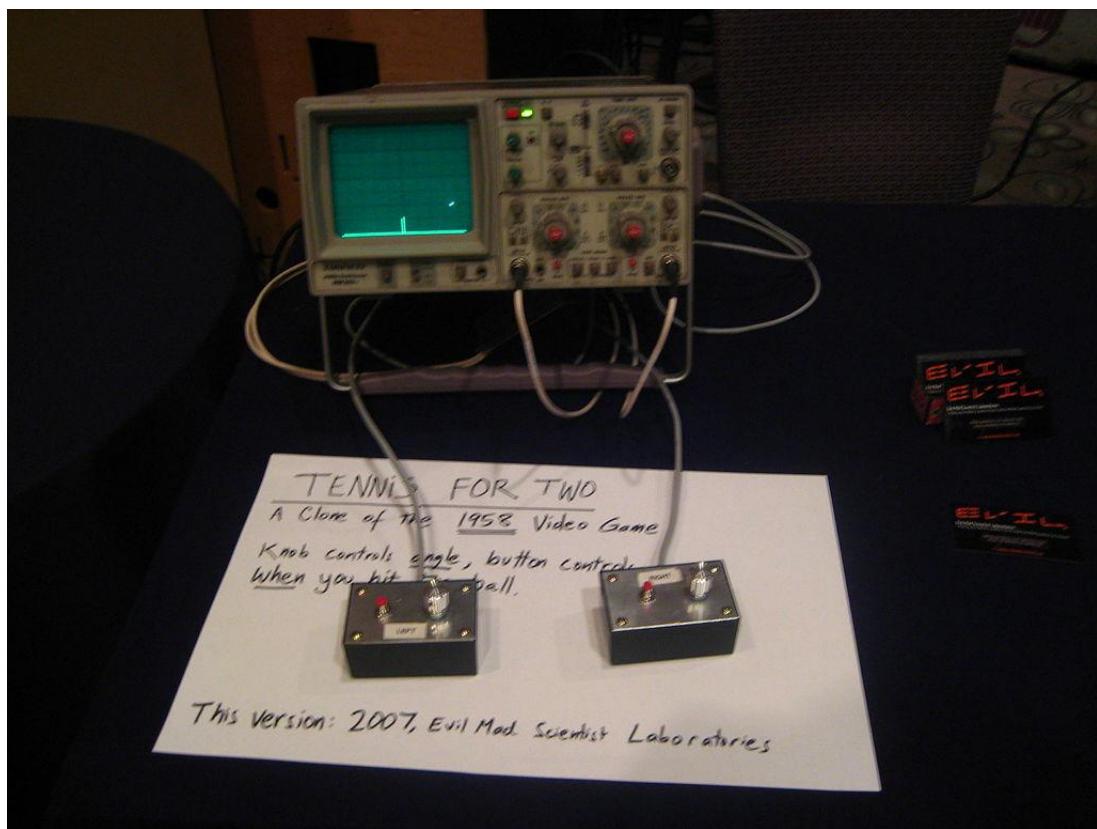


FIGURA 1. COPIA DE TENNIS FOR TWO

En la década de los 70 comenzaron a abaratarse los costes de fabricación de tal forma que esto permitió la aparición de las primeras máquinas y videojuegos dirigidos al gran público. Entre estos primeros títulos es obligatorio destacar los míticos *Computer Space* (1971) y *Pong* (1972).



FIGURA 2. COMPUTER SPACE

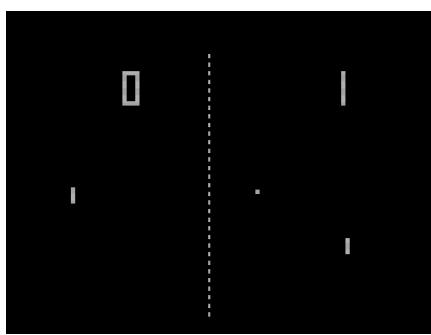


FIGURA 3. PONG

Con el paso del tiempo, los videojuegos han ido evolucionando a pasos agigantados, hasta el punto en el que en la actualidad podemos distinguir una enormísima variedad de videojuegos (en parte gracias a la existencia de herramientas de creación gratuitas y disponibles para todo el mundo), todos totalmente categorizados en lo que denominamos **géneros**.

El primer intento serio de clasificación de videojuegos conocido fue realizado en 1984 por Chris Crawford en su libro “The Art of Computer Game Design”, donde él identificó dos grandes grupos, uno de Juegos de Habilidad y Acción, y otro de Juegos de Estrategia y Cognitivos. A su vez, dentro de estos grandes grupos hizo una serie de subgrupos, por ejemplo, dentro del primer gran grupo estarían incluidos algunos géneros como los juegos de deportes, combate o carreras, y dentro del segundo gran grupo los juegos de aventuras, mazmorras, o incluso educacionales.

En la actualidad podemos encontrar una variedad mucho más enorme de géneros, entre los más populares destacan los siguientes:

- **Lucha:** Caracterizados por ser videojuegos donde se recrean combates en los que un jugador es capaz de controlar a un personaje y generalmente se emplean movimientos de artes marciales (aunque siempre puede haber excepciones e ir más allá de ello), en combates de 1 contra 1 normalmente.
- **Plataformas:** Un jugador controla a un personaje por un escenario evitando obstáculos físicos e intentando no morir hasta llegar a un objetivo concreto el cual suele ser el final del nivel.
- **Disparos** o *Shooter*: El jugador hace un uso continuado de armas de fuego eliminando enemigos ya sea en primera como en tercera persona, etc.
- **Rol** o *Rpg*: El jugador encarna a uno o varios personajes, teniendo en ocasiones la capacidad de decidir sobre el transcurso de la historia, los rasgos físicos de los personajes, habilidades y objetos equipados, estrategias, etc.

Un género que ha ganado popularidad en los últimos años es el de **supervivencia**, debido a que resulta un género adecuado para estudios *indie* que no cuentan con recursos para diseñar numerosos escenarios y/o niveles.

Este último va a ser el género que vamos a tratar a lo largo de este proyecto.

2.2 Los videojuegos de supervivencia

En primer lugar, cabe destacar que el género de Supervivencia dentro de los videojuegos se ha hecho hueco en el mercado actual principalmente gracias a los estudios independientes. La creación de videojuegos sin la necesidad de tener a una editora que lo respalde ha abierto las puertas a recuperar algunos de los géneros en los que ya nadie confiaba. Y uno de estos géneros es este, el género de **Supervivencia**.

El objetivo primordial de este tipo de videojuegos es uno: no morir. Generalmente, en un videojuego de supervivencia nos encontraremos en un entorno

hostil, donde tenemos que buscar objetos que nos permitan fabricar elementos que nos ayuden a defendernos, alimentarnos, refugiarnos...

Para llevar a cabo este proyecto se han tomado como referencia algunos de los videojuegos más populares de este género o que puedan tener cierta relación con el género:

- Dayz
- Minecraft
- Don't Starve
- Rust
- ARK: Survival Evolved
- Terraria
- The Legend of Zelda: Breath of the Wild

2.2.1 Dayz



FIGURA 4. DAYZ

Este videojuego nació a raíz de un *mod* del videojuego ArmA II (un *shooter*). Se trata de un título de mundo abierto donde se mezclan aventuras y acción dentro de una experiencia multijugador online masiva.

El videojuego está ambientado en un mundo postapocalíptico, poblado por zombis y humanos, donde las acciones tienen consecuencias permanentes, mantenerse a salvo de los entornos potencialmente hostiles es clave, y la muerte implica empezar desde cero, perdiendo todo lo que habíamos conseguido a lo largo de la partida. El videojuego ofrece una gran libertad, y la experiencia multijugador online con la que cuenta da la posibilidad al jugador de interactuar con desconocidos, siendo ésta un arma de doble filo.

2.2.2 Minecraft



FIGURA 5. MINECRAFT

Uno de los videojuegos más populares que existen, caracterizado principalmente por su enorme enfoque en la creatividad.

El videojuego, aparte de contar con un modo supervivencia, cuenta también con un modo creativo en el cual es puedes construir y explorar sin límite y sin ningún tipo de peligro, puesto que no puedes morir.

En lo que respecta al modo supervivencia, como en la mayoría de exponentes del género, comenzamos sin nada y tendremos que ir avanzando poco a poco para poder conseguir más y mejores recursos, además de comida para no morir de hambre, o armas para defenderte de los enemigos. En definitiva, el modo supervivencia tiene como objetivo mantenernos con vida, sin dejar de lado el factor creativo. En este videojuego tenemos dos parámetros que nos indicarán la salud actual del personaje: la vida y el hambre. Algo que caracteriza en gran medida a este videojuego

es la gran capacidad de creación que, junto a su modo supervivencia, da lugar a un mundo de posibilidades infinitas.

Además, el mapa en el que se juega cada partida es imposible que sea igual al de la partida anterior, puesto que el mismo juego cuenta con un algoritmo de creación de mapas procedural, el cuál incluso permite la creación de ecosistemas posicionados de forma procedural, islas, e incluso mazmorras.

2.2.3 Don't Starve



FIGURA 6. DON'T STARVE

Videojuego de supervivencia que, como Minecraft, nos lleva a sobrevivir partiendo de la nada, obligándolos a explorar el mapa en busca de recursos, alimentos, y otros elementos que nos ayuden a sobrevivir en un mundo especialmente hostil con una dificultad algo más elevada que en otros videojuegos de su género debido a la gran cantidad de elementos que llevarán a sorprender en más de una ocasión al jugador.

Algo que diferencia este videojuego de otros de su mismo género es que al inicio cuenta con una selección de diferentes personajes, cada uno con una habilidad diferente. La

idea de contar con una habilidad diferente es la de cambiar la manera de jugar del mismo jugador en cada partida.

En este videojuego estaremos obligados a estar pendientes de tres parámetros a lo largo de toda la partida el hambre, la vida y la cordura. Morir de hambre o perder toda la vida tiene consecuencias bastante claras, sin embargo, perder cordura cambiará nuestra manera de ver el mundo, llegando incluso a invocar monstruos imaginarios de los cuales no podremos defendernos.

Destacar que el juego cuenta con un mapa generado de forma procedural, de tal manera que el mapa a explorar por el jugador en cada partida será diferente.

2.2.4 Rust



FIGURA 7. RUST

Rust es otro de esos juegos de supervivencia que en los últimos años ha hecho mucho eco, aunque la pésima optimización de éste ha dado lugar a múltiples críticas. Con una cámara en primera persona, comenzaremos con una piedra, dos botiquines y una antorcha, y a partir de ahí el juego da toda libertad del mundo para explorar el

entorno. Con la piedra podremos romper elementos del entorno, como árboles, para obtener recursos y crear herramientas que nos permitan obtener mejores materiales, comidas, etc.

La supervivencia en este juego va totalmente ligada a tres factores fundamentales, los cuales serían hambre, hidratación y salud.

Además, cuenta con un modo multijugador online en el cuál confiar en los demás jugadores (o no) podría ser clave a lo largo de las partidas.

2.2.5 ARK: Survival Evolved

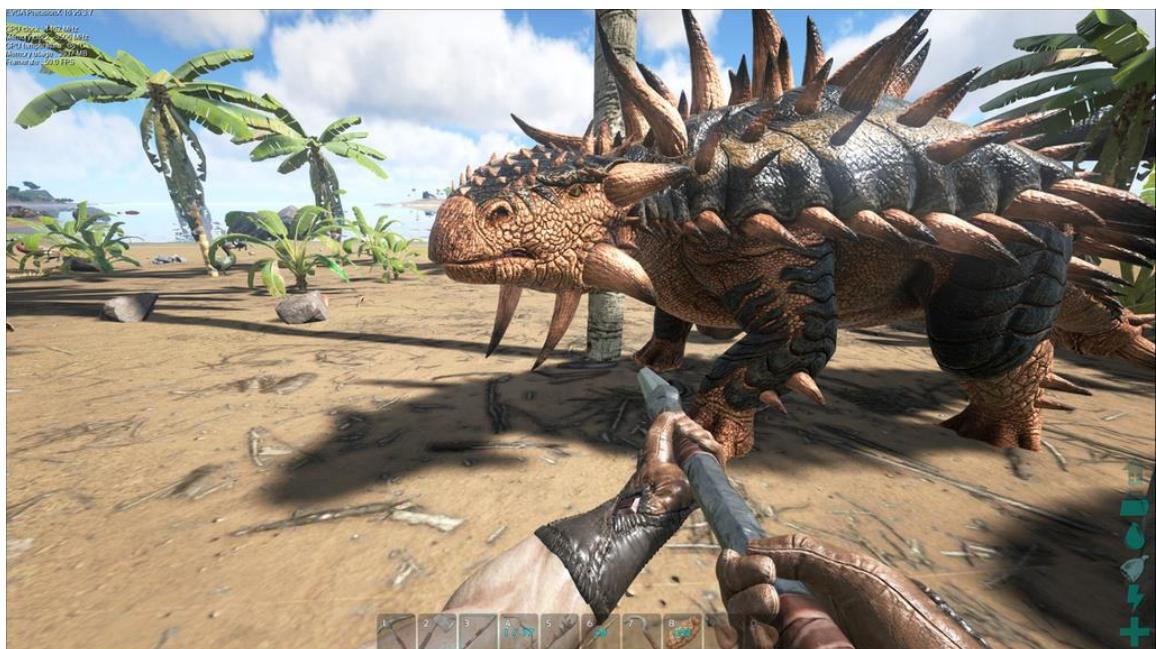


FIGURA 8. ARK: SURVIVAL EVOLVED

Otro videojuego clave en los últimos años sobre este género, y otro videojuego con problemas de optimización bastante graves, al igual que el mencionado anteriormente.

Obviando dichos problemas, estamos ante un juego en primera persona que ofrece un amplio abanico de posibilidades.

En este videojuego somos un humano que llega a una misteriosa isla, donde tendremos que sobrevivir cazando dinosaurios y construyendo un refugio.

En este título, una de las novedades destacables es que podemos domar dinosaurios y tenerlos como mascotas, además de usarlos para combatir contra otros dinosaurios o incluso contra otros jugadores.

Los jugadores cuentan con la posibilidad de crear tribus, dando lugar a guerras entre tribus. A su vez, esto crea la necesidad de estos grupos de jugadores de defenderse del resto, creando verdaderas fortificaciones.

Para sobrevivir en este juego, además de defenderse del resto de jugadores y animales, deberemos tener en cuenta una serie de parámetros: Temperatura corporal, salud, hambre, y sed. Todos estos parámetros son fundamentales a la hora de sobrevivir, puesto que tener cualquiera de ellos en niveles bajos puede llevar a la muerte fácilmente.

2.2.6 Terraria



FIGURA

9.

TERRARIA

Muy similar a Minecraft, pero en 2D.

Comenzamos con una espada corta de cobre, un pico y un hacha, que nos permitirán derrotar enemigos y conseguir materiales o ingredientes para llevar a cabo recetas de nuevos objetos.

Este título tiene un modo multijugador local a pantalla dividida de hasta cuatro jugadores, o incluso online de hasta ocho jugadores.

En este videojuego el único factor determinante que pueda llevar a la muerte a los personajes es la vida, no existe el hambre o la sed como en otros títulos.

2.2.7 The Legend of Zelda: Breath of the Wild



FIGURA 10. THE LEGEND OF ZELDA: BREATH OF THE WILD

En el caso de este título, no estamos ante un videojuego de supervivencia como tal, pero consideré interesante el hecho de incluirlo en este listado debido a que éste incluye algunas características propias de este género.

En este videojuego, dos factores que pueden determinar la muerte de nuestro personaje son la vida y la temperatura corporal. En el momento en el que la temperatura corporal es demasiado elevada o demasiado baja, nuestro personaje comenzará a perder vida progresivamente hasta morir. Para controlar la temperatura corporal, deberemos equiparnos con vestimenta adecuada capaz de equilibrarla teniendo en cuenta el entorno en el que nos encontramos y, además, la hora del día, puesto que las noches son más frías que el día, por ejemplo.

Otra característica a destacar de este título es que contamos con un marcador de ruido que nos indicará el nivel de ruido que está haciendo nuestro personaje, siendo ésta la

clave para atacar en sigilo a los enemigos o huir de estos sin que se den cuenta. También, cuando nuestro personaje realiza algunos movimientos (como la escalada) veremos que disminuye otro factor, el de la resistencia. La resistencia determinará el cansancio de nuestro personaje, limitando algunas de sus acciones cuando la resistencia se encuentre en niveles mínimos.

Para finalizar, *The Legend of Zelda: Breath of the Wild* cuenta con la capacidad de cocinar alimentos, como en casi cualquier título de supervivencia, pero en este caso no servirá para saciar el hambre (puesto que este factor no está presente) sino para curarnos o aportar a nuestro personaje algunas características extras, como puede ser resistencia a daños de algún tipo de elemento o, simplemente, vida extra más allá de la vida total que dispongamos.

2.3 Conclusión sobre el análisis de los videojuegos de Supervivencia

Tras el análisis de algunos de los diferentes videojuegos de Supervivencia que se encuentran en el mercado, podemos sacar en claro los elementos más destacables de este tipo de videojuegos y que, en gran parte, comparten entre sí.

- **Generación procedural:** La generación procedural del terreno y los mapas de estos videojuegos es algo a lo que muchas desarrolladoras recurren. Esto facilita la tarea de diseño de niveles y escenarios, además aporta una variedad enorme de mapas distintos.
- **Sistema de fabricación o *crafting*:** Esta es una mecánica que la gran mayoría de videojuegos de este tipo posee. A esto se añade la capacidad de recolectar recursos que se encuentren a lo largo de la partida y, con la ayuda de estos, construir diferentes elementos útiles que ayuden a los jugadores a avanzar y evolucionar en la partida.

- **Sistema de inventario y equipo:** Esta mecánica va incluida de manera obligatoria en videojuegos de este género, sobre todo si cuentan con un sistema de fabricación. Poder gestionar los diferentes objetos del inventario, y equipar algunos objetos sobre los personajes de los jugadores es algo muy común.
- **Modo multijugador:** No todos los videojuegos de este género poseen esta característica, pero muchos de ellos sí. La cooperación entre varios jugadores es algo que enriquece mucho la jugabilidad en este tipo de videojuegos.
- **Mecánica *roguelike*:** Este tipo de mecánicas aporta una serie de características a los videojuegos de supervivencia, como la muerte permanente, la escasa o nula narrativa, o la presencia de mecánicas aleatorias y/o procedurales (como la generación de mazmorras, cuevas, terreno, o incluso eventos aleatorios).
- **Personaje principal frágil:** El objetivo de los videojuegos de supervivencia es el de sobrevivir y, como en todo videojuego, para conseguir un objetivo tienen que existir una serie de obstáculos o dificultades. En el caso de este tipo de juegos, siendo este el único y claro objetivo, la dificultad se vuelca completamente sobre la salud del personaje principal, el cual, generalmente, es bastante frágil. En resumen, es fácil morir.

A la hora de la realización de este videojuego se han de tener en cuenta todas y cada una de las características mencionadas anteriormente, puesto que son las premisas básicas que permiten distinguir un videojuego de supervivencia del resto de géneros.

3. Objetivos

Este proyecto tiene como objetivo la creación de un videojuego de supervivencia multijugador online con estilo *cartoon* para PC. Los jugadores podrán invitar a sus amigos a las partidas a través de una plataforma digital de videojuegos como Steam. Los mapas se crearán de forma procedural en cada partida, siendo siempre totalmente distintos.

La herramienta principal para la creación de este videojuego será el motor gráfico Unity3D, utilizando el lenguaje de programación C#.

De esta manera, podemos distinguir una serie de objetivos:

- Aprender a utilizar el motor gráfico Unity3D.
- Estudiar el algoritmo necesario para la generación procedural de terrenos e implementarlo.
- Estudiar y aplicar técnicas de modelado, texturizado y shaders que aporten un estilo *cartoon* al videojuego.
- Estudiar distintas APIs de red e implementar una compatible con el motor gráfico que permita la creación y gestión de partidas, además de una correcta sincronización entre los jugadores y sus acciones dentro de la partida.
- Estudiar las diferentes plataformas digitales e implementar el servicio social de jugadores de una ellas.

El fin de este proyecto es académico, en ningún momento se tiene previsto un uso comercial de éste.

4. Metodología

4.1 Metodología del desarrollo

Para la elaboración de este proyecto se ha optado por usar la metodología japonesa Kanban.

4.1.1 La metodología Kanban



FIGURA 11. METODOLOGÍA KANBAN

Kanban es una metodología ágil basada en tarjetas que irán a través de diversas etapas de trabajo hasta su finalización.

El método Kanban tiene sus raíces en cuatro principios básicos:

- Comience lo que hace ahora: Se inicia con las funciones y procesos que ya se tienen y estimula cambios continuos, incrementales y evolutivos a su sistema.
- Se acuerda perseguir el cambio incremental y evolutivo: El equipo debe estar de acuerdo que el cambio continuo, gradual y evolutivo es la manera de hacer mejoras en el sistema y debe apoyarse a ello. Esta metodología anima a los pequeños y continuos cambios incrementales.
- Respetar el proceso actual, los roles, las responsabilidades y los cargos: Tenemos que facilitar el cambio futuro; acordando respetar los roles actuales,

responsabilidades y cargos, eliminamos los temores iniciales. Esto nos debería permitir obtener un mayor apoyo a nuestra iniciativa Kanban.

- Liderazgo en todos los niveles: Se debe alentar hechos de liderazgo en todos los niveles de la organización de los contribuyentes individuales a la alta dirección.

Los beneficios que otorga dicha metodología son los siguientes:

- Estímulo del rendimiento.
- Organización y colaboración.
- Distribución del trabajo.

4.2 Gestión del proyecto



FIGURA 12. LOGO DE TRELLO

Para la gestión de las actividades de este proyecto se ha usado la herramienta *Trello*. Es una herramienta basada en la metodología ágil Kanban, la cual hace uso de un registro de actividades a través de tarjetas virtuales, permitiendo esto una organización de las tareas, agregar listas, adjuntar archivos, agregar comentarios, fechas, colores, subtareas, clasificarlas por grupos, etc.

En definitiva, se trata de un tablón virtual en el que se pueden colgar ideas, tareas, imágenes o enlaces, pudiendo usarse para cualquier tipo de tarea que requiera organizar información.

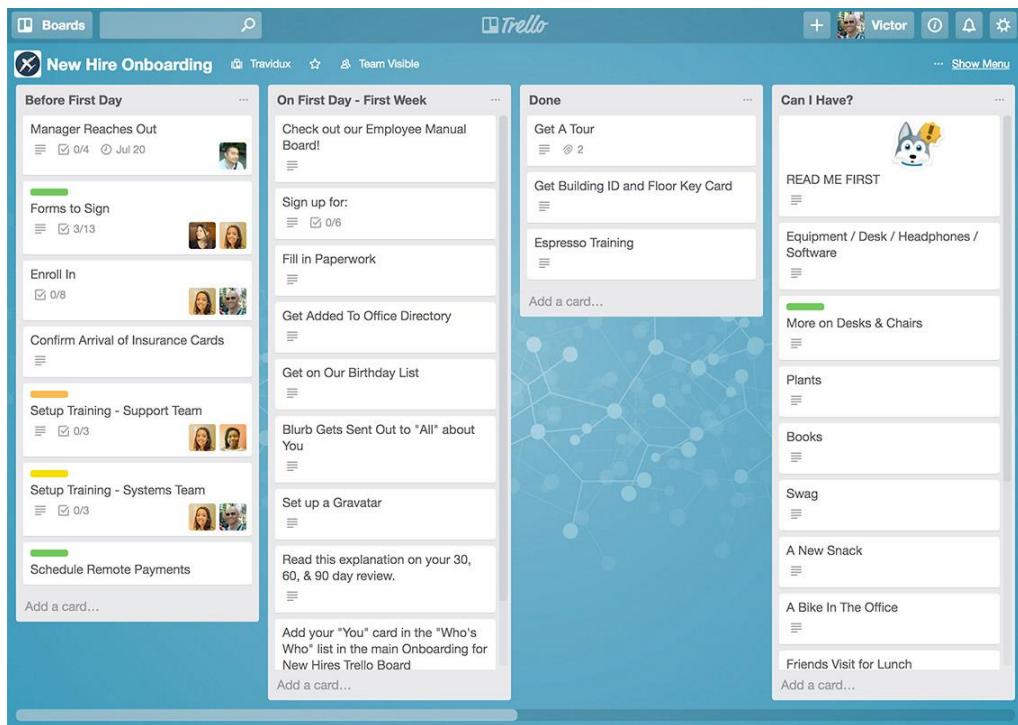


FIGURA 13. EJEMPLO DE PROYECTO EN TRELLO

4.3 Control de versiones y repositorio



FIGURA 14. LOGO DE GITHUB

Siendo un trabajo individual el uso de un repositorio podría carecer de sentido; sin embargo, he optado por hacer uso de dicha herramienta por el simple hecho de tener una copia de seguridad en el caso de que ocurra algún problema, de tal manera que siempre existirá la opción de volver a alguna versión anterior.

El repositorio se encuentra alojado en *GitHub*.

5. Cuerpo del trabajo

En este punto del documento se encuentra toda la información sobre el desarrollo del proyecto.

Antes de esto, voy a hacer un estudio sobre todas las herramientas disponibles para la elaboración de éste y cuáles son por las que finalmente me he decantado.

A continuación de esto, escribiré todo el documento de diseño (GDD: Game Design Document) y, para finalizar, escribiré sobre todo el desarrollo e implementación del videojuego.

5.1 Análisis y elección de herramientas

Antes de ponerse manos a la obra, hay que hacer un estudio de todas las herramientas de las que podemos hacer uso y elegir con cuáles de ellas vamos a trabajar.

5.1.1 Motor de videojuegos

A la hora de la creación de un videojuego, una herramienta indispensable para ello son los motores gráficos.

La funcionalidad básica de un motor gráfico es la de proveer al videojuego de un motor de renderizado (para gráficos 2D y/o 3D), motor de físicas, sonidos, *scripting*, animación, etc. Básicamente, los motores gráficos aportan todo tipo de herramientas que permiten crear todos los elementos que todo videojuego actual posee. Existe una variedad enorme de motores gráficos, aunque voy a destacar los más populares.

5.1.1.1 Unity



FIGURA 15. LOGO DE UNITY

Uno de los motores gráficos más usados actualmente.

Es totalmente gratuito, aunque cuenta con una versión de pago con algunas diferencias como es el hecho de poder disfrutar de algunos servicios en la nube, o mayor capacidad de jugadores en videojuegos multijugador online.

Se trata de un motor gráfico con una curva de aprendizaje suave, permitiendo crear juegos al poco tiempo de uso. Algo a destacar de esta gran herramienta es la enorme cantidad de tutoriales y ayudas que hay en la web, además de una amplia documentación.

Unity puede usarse junto con programas como Blender, 3ds Max, Maya, ZBrush, Cinema 4D o Adobe Photoshop. Los cambios realizados a los objetos creados con estos programas se actualizan automáticamente en todas las instancias de ese objeto durante todo el proyecto sin necesidad de volver a importar manualmente.

Los lenguajes de programación con los que se puede trabajar en este programa son: **C#, JavaScript y Boo**. Los tres lenguajes se consideran lenguajes de programación orientado a objetos, siendo el primero el más utilizado por la comunidad de Unity y, por tanto, sobre el que más información hay al respecto en internet.

Unity actualmente se encuentra en su versión 5, y está disponible para Windows, Linux y macOS.

Entre los videojuegos más populares creados mediante Unity tenemos **Hearthstone** o **Inside**.

5.1.1.2 Unreal Engine



FIGURA 16. LOGO DE UNREAL ENGINE

Gratis y potente, Unreal Engine es uno de los motores gráficos más utilizados en la industria actualmente. Este motor gráfico permite crear gráficos muy realistas, pero a su vez la curva de aprendizaje es bastante más elevada que en el caso anterior, y es que para poder conseguir algo decente habrá que dedicarle muchas horas.

Unreal Engine se encuentra actualmente en su versión número 4, aunque aún existen videojuegos que siguen usando su versión número 3 como es el caso de **Smite**.

Este motor gráfico está disponible para Windows, Linux y macOS.

Un punto interesante que ofrece este motor es la opción de programación basada en nodos y componentes, la cual facilita el trabajo y hace el prototipado muy rápido.

Este sistema de *blueprints* es muy parecido a crear diagramas de flujo y aporta a este programa una herramienta muy versátil y potente.

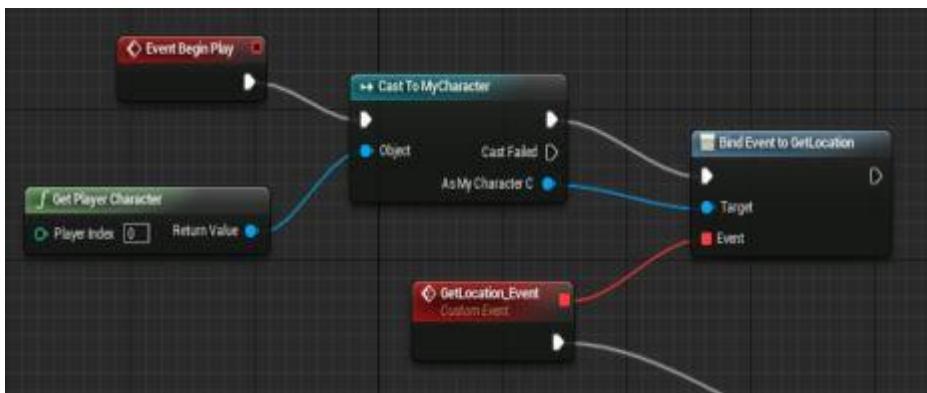


FIGURA 17. PROGRAMACIÓN BASADA EN NODOS EN UNREAL ENGINE

5.1.1.3 Cocos2d



FIGURA 18. LOGO COCOS2D

Motor gráfico cuya peculiaridad es la creación de videojuegos que se pueden implementar en diferentes plataformas (Android, iOS, Windows, Windows Phone, Mac, Linux, etc) manteniendo la misma base de código y con pocas adaptaciones específicas para cada plataforma.

Este motor gráfico se utiliza principalmente para la creación de videojuegos en dispositivos móviles.

El lenguaje de programación que se usa para trabajar aquí es C++, elevando de esta manera su curva de aprendizaje con respecto a otros motores como Unity.

5.1.1.4 Cryengine



FIGURA 19. LOGO DE CRYENGINE

Motor gráfico conocido gracias a sus increíbles efectos de iluminación y gran realismo. El principal atractivo de Cryengine 3, su última versión, es la facilidad con la que podremos crear escenarios de una calidad asombrosa.

Por desgracia, con este motor gráfico no contaremos con demasiadas facilidades a la hora de crear otra cosa que no sea un *shooter*.

Algunos de los títulos creados con este motor son **Far Cry** y **Crysis**.

5.1.1.5 Herramienta seleccionada: Unity

De entre todos los motores gráficos anteriores, me he decantado por **Unity3D**. Entre mis razones, destaco que es uno de los motores más utilizados actualmente en la industria de los videojuegos, además aprender más sobre él era uno de mis objetivos principales de este proyecto, por lo que mi decisión estaba clara desde el inicio.

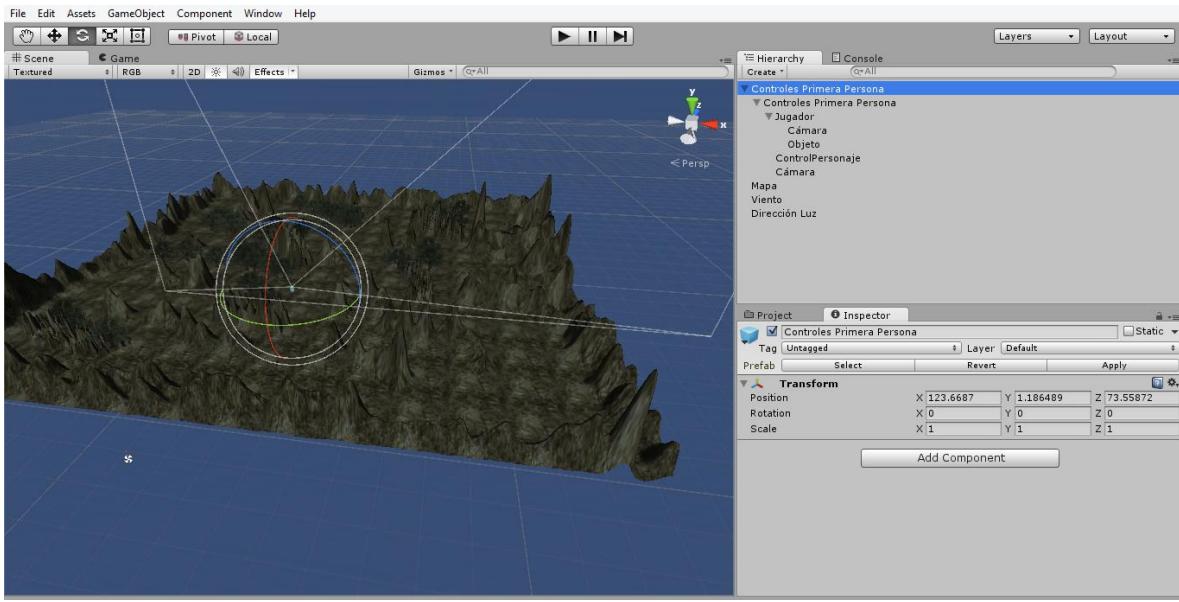


FIGURA 20. EJEMPLO PROYECTO DE UNITY3D

5.1.2 Software de modelado

Se define como modelado 3D al proceso de desarrollo de una representación matemática de cualquier objeto tridimensional a través de un software especializado. El resultado recibe el nombre de modelo 3D.

Hoy en día, los modelos 3D son usados en una amplia variedad de campos: en la industria médica (órganos 3D), en la industria del cine (personajes y objetos), y en la industria del videojuego (personajes, objetos y otros recursos para un videojuego).

Como el objetivo de este proyecto es la elaboración de un videojuego, he investigado sobre los programas de modelado 3D más utilizados en este campo.

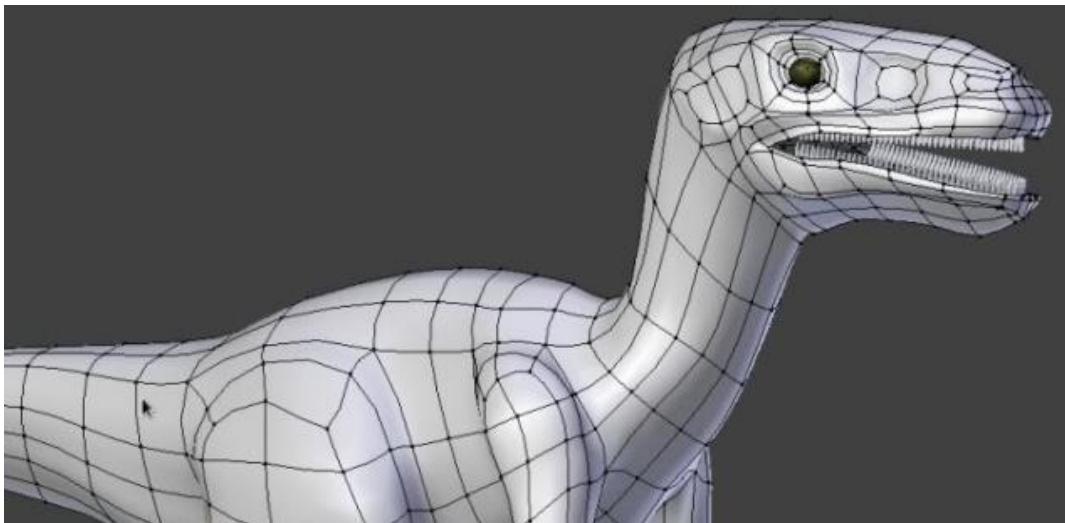


FIGURA 21. EJEMPLO DE CREACIÓN DE MODELO 3D

5.1.2.1 Autodesk Maya



FIGURA 22. LOGO DE AUTODESK MAYA

Programa informático dedicado al desarrollo de gráficos 3D por ordenador, efectos especiales y animación. Es totalmente compatible con Windows, Linux y macOS.

Maya se caracteriza por su potencia y las posibilidades de expansión y personalización de su interfaz y herramientas. El programa posee diversas herramientas modelado, animación, renderización, simulación de ropa y cabello, dinámicas (simulación de fluidos), etc. Como dato curioso, este software 3D es el único acreditado con un Óscar gracias al enorme impacto que ha tenido en la industria cinematográfica.

Este programa es de pago, sin embargo, existe una licencia de estudiantes totalmente gratuita.

Algunas de las películas que han hecho uso de este software son Jurassic Park y Terminator 2.

5.1.2.2. Autodesk 3ds Max



FIGURA 23. LOGO DE AUTODESK 3DS MAX

Programa de creación de gráficos y animación 3D desarrollado por Autodesk.

Totalmente compatible con los sistemas operativos de Windows, Linux y macOS, este software, con una arquitectura basada en *plugins*, es uno de los programas de modelado y animación 3D más utilizado en la actualidad, especialmente para la creación de videojuegos, siendo el más utilizado en esta industria.

Este programa es de pago, sin embargo, existe una licencia de estudiantes totalmente gratuita.

Algunas de las películas que han hecho uso de 3ds max son Iron man y Avatar.

5.1.2.3 Blender



FIGURA 24. LOGO DE BLENDER

Programa informático dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. En Blender, además, se pueden desarrollar videojuegos ya que éste posee un motor gráfico interno.

A diferenciar del resto, este programa es totalmente gratuito.

Como dato curioso, se utilizó Blender para la película de Spider man 2.

5.1.2.4 Herramienta seleccionada: Autodesk 3ds Max

De entre todos los programas de modelado anteriores, me he decantado por **3ds Max**. La principal razón es que este software de modelado es con el que más familiarizado estoy y, teniendo en cuenta el gran uso que tiene hoy día en la industria de los videojuegos y teniendo la oportunidad de poder usar una licencia de estudiante, siempre que se tenga la oportunidad es bueno aprender más sobre el uso de este.

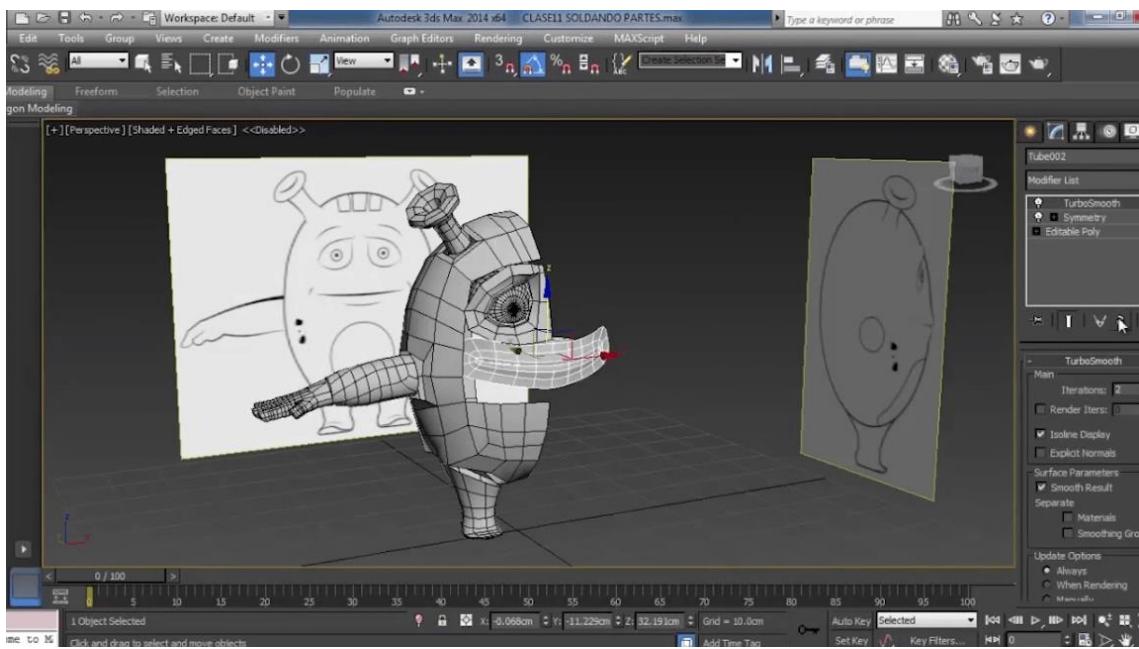


FIGURA 25. EJEMPLO PROYECTO 3Ds MAX

5.1.3 Software de texturizado

En el campo de los gráficos por computadora, texturizar consiste en crear una imagen de mapa de bits, la cual cubriría la superficie de un objeto virtual, siendo este tridimensional o bidimensional, con un programa específico.

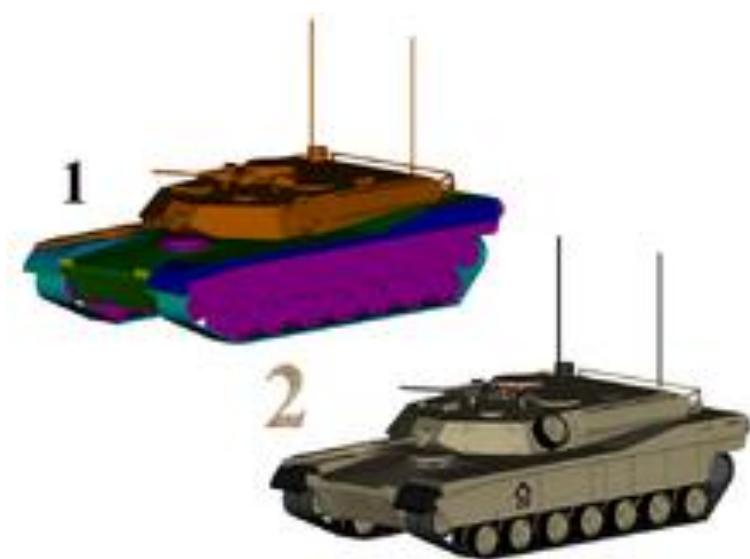


FIGURA 26. EJEMPLO DE TEXTURIZADO. 1 SIN TEXTURAS, 2 CON TEXTURAS

Para el texturizado se pueden utilizar perfectamente cualquiera de los tres programadas de modelado que he mencionado anteriormente.

Aunque existen algunas diferencias; por ejemplo, en Blender es posible utilizar una herramienta que recibe el nombre de *texture painting*, siendo ésta muy intuitiva ya que te permite pintar las texturas sobre el propio modelo como si se tratase de una escultura real.

Existen otros que también ofrecen excelentes resultados.

5.1.3.1 Autodesk Mudbox



FIGURA 27. LOGO DE AUTODESK MUDBOX

Se trata de un software de modelado 3d, texturizado y pintura digital desarrollado por Autodesk. Se utilizó por primera vez en la película de King Kong en 2005. Este software es totalmente compatible con Windows, Linux y macOS.

Con mudbox es posible texturizar *pintando* directamente sobre el modelo.

5.1.3.2 Zbrush



FIGURA 28. LOGO DE ZBRUSH

Software de modelado 3d, escultura y pintura digital, totalmente compatible con los sistemas operativos Windows y macOS.

Este programa destaca principalmente por ser un software capaz de esculpir detallados modelos de un modo semejante a pintar en los mismos. De esta manera, es capaz de pintar sobre los modelos como si se estuviese pintando una escultura de verdad. Este programa se utilizó en películas como Underworld o El señor de los anillos.

5.1.3.3 Herramienta seleccionada: Autodesk 3ds Max

Finalmente, para el texturizado de los modelos de mi proyecto me he decantado por utilizar el mismo programa de modelado, **3Ds Max**, puesto que éste ya cuenta con herramientas de texturizado y prefiero utilizar el menor número de programas posibles.

5.1.4 Programas de dibujo y edición gráfica.

5.1.4.1 GIMP



FIGURA 29. LOGO DE GIMP

Programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Se trata de un software libre y gratuito, compatible con los sistemas operativos Linux, Solaris, Windows, macOS, entre otros.

Una de las características destacables de GIMP es que permite el tratado de imágenes en capas, para poder modificar cada objeto de la imagen en forma totalmente independiente a los demás elementos en otras capas de la imagen. También pueden subirse o bajarse de nivel las capas, en una pila, para facilitar el trabajo de la imagen.

Este software permite leer y escribir una gran cantidad de formatos como JPG, GIF, PNG, TIFF, BMP, PIX entre otros. Además, como dato curioso también es capaz de abrir la mayoría de ficheros PSD (provenientes de Photoshop).

5.1.4.2 Adobe Photoshop



FIGURA 30. LOGO DE PHOTOSHOP

Editor de gráficos rasterizados usado principalmente para el retoque de fotografías y gráficos. Es líder mundial del mercado de las aplicaciones de edición de imágenes y domina este sector de tal manera que su nombre es ampliamente empleado como sinónimo para la edición de imágenes en general.

Este software de edición gráfica soporta una gran cantidad de tipos de imágenes como BMP, JPG, PNG, y GIF entre muchos otros.

Al igual que GIMP, se trabaja por capas.

5.1.4.3 PaintTool SAI



FIGURA 31. LOGO DE PAINTTOOL SAI

Programa ligero que se utiliza principalmente para dibujar y pintar, y que apenas posee opciones de edición de imagen más allá de ajustar el Brillo/Contraste y Tono/Saturación.

El único sistema operativo compatible con este programa es Windows, aunque existen formas no oficiales de hacerlo funcionar en Linux y MacOs.

Este programa soporta un amplio rango de formatos populares como PNG, JPG, BMP, e incluso PSD (formato de Photoshop).

Se puede trabajar por capas separadas o agrupadas.

5.1.4.4 Herramientas seleccionadas: Adobe Photoshop y PaintTool SAI

En este caso me he decantado por dos herramientas, **Photoshop** y **PaintTool SAI**. Photoshop y GIMP son dos programas muy similares, así que he elegido el primero porque es el que mejor sé utilizar y porque es el único de los dos que, antes de haber comenzado este proyecto, ya estaba instalado en mi equipo.

En el caso de PaintTool SAI, si bien es cierto todo lo que se puede hacer en este programa también se puede hacer en Photoshop, pero, simplemente por la experiencia dibujando en PC, el programa que más he utilizado es PaintTool SAI por lo que me conozco bastante mejor sus herramientas que de las que dispone Photoshop. Para el programa de dibujo dispongo de una tableta digital.

Photoshop lo utilizaría para creación de texturas y algunos efectos (HUD, logo del juego, etc), y PaintTool SAI lo utilizaría para bocetos, dibujar el logo, HUD, y pintar.

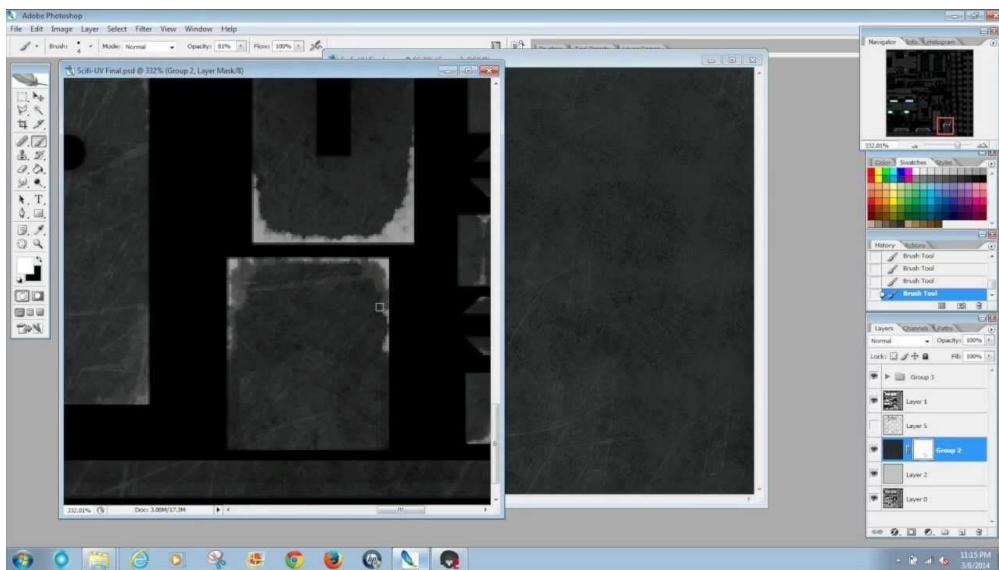


FIGURA 32. EJEMPLO DE PROYECTO EN PHOTOSHOP

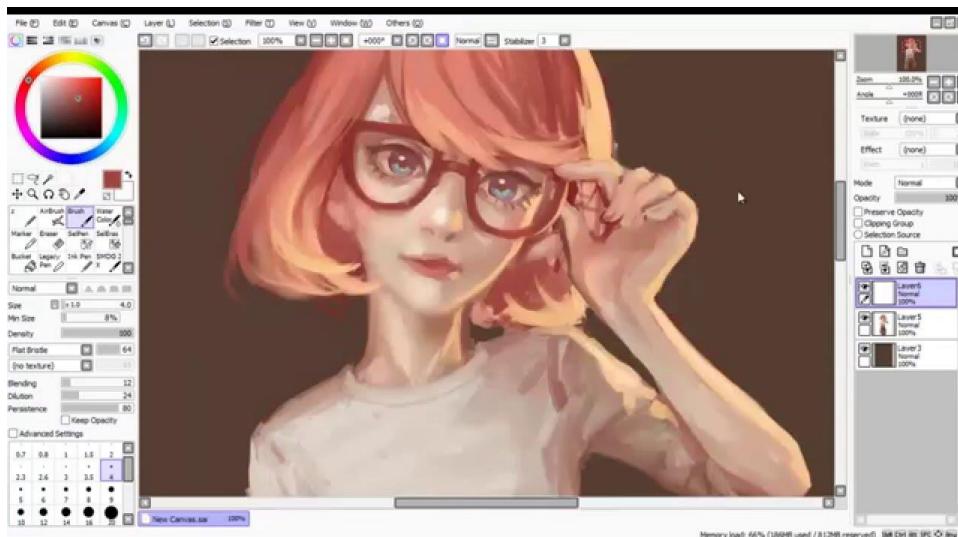


FIGURA 33. EJEMPLO DE PROYECTO EN PAINTTOOL SAI

5.1.5 Rigging, skinning y animaciones.

Se define como *rigging* al proceso de crear un sistema de controles digitales y agregárselos a un modelo 3D para que así pueda ser animado fácilmente y eficientemente. Es, principalmente, un paso clave dentro del proceso de la creación de una animación 3D.

Esos controles digitales que se asignan al modelo 3D se consiguen gracias a un sistema esquelético que va de acuerdo con la forma básica del personaje.

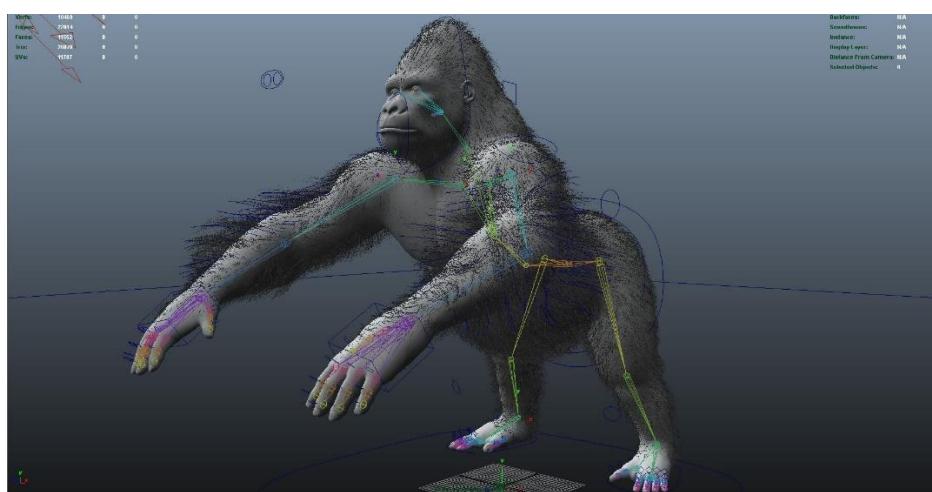


FIGURA 34. EJEMPLO DE RIGGING

En el caso del *skinning*, es el proceso por el cual se asigna el *rigging* a una malla 3D. Hay muchas posibilidades, desde las que permiten la deformación de la malla hasta las que permiten a los objetos comportarse como sólidos al estilo de una mecánica restringida. Básicamente, el *skinning* es una relación de parentesco entre la malla y el esqueleto donde el que manda es este último.

Con este proceso se define a qué polígonos afecta el movimiento de cada hueso y en qué medida.

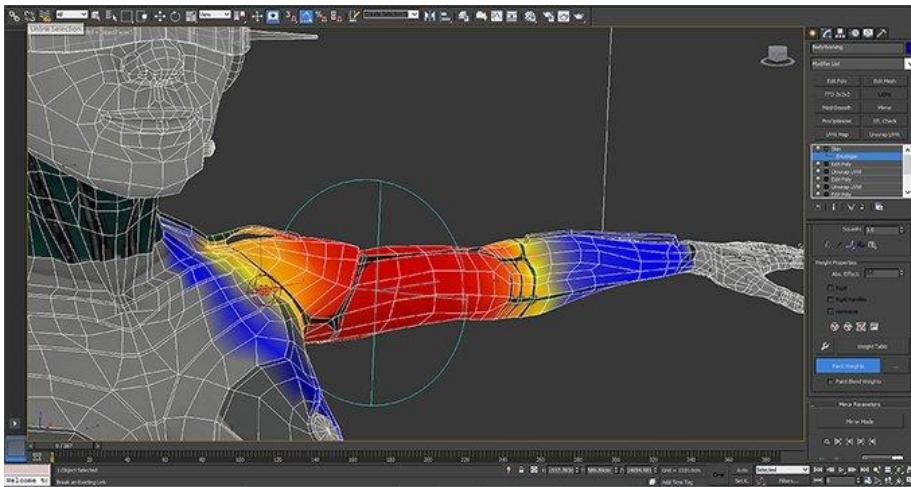


FIGURA 35. EJEMPLO DE SKINNING

Y, finalmente, con el *rigging* y *skinning* se hacen posibles las animaciones en 3D. Este es un proceso complejo que consiste en la deformación o movimientos de un modelo 3D a lo largo del tiempo.

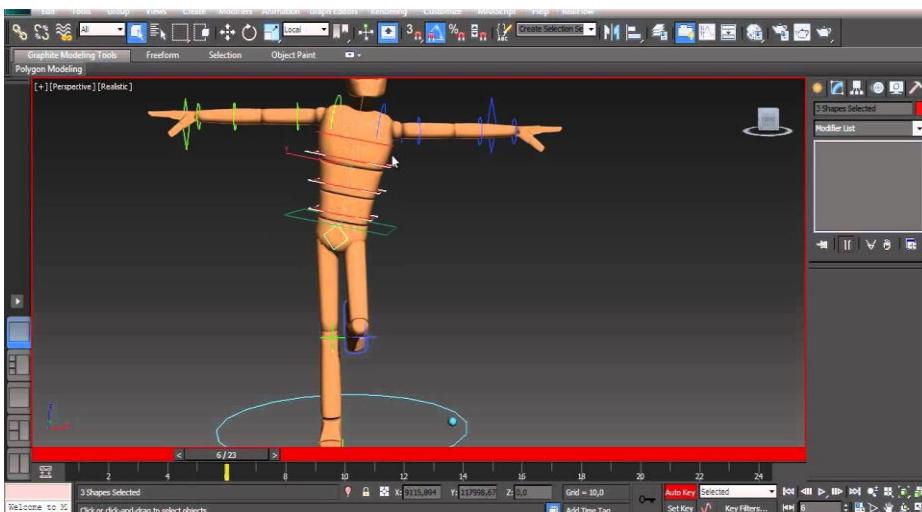


FIGURA 36. EJEMPLO DE ANIMACIÓN 3D

5.1.5.1 Herramienta seleccionada: Autodesk Maya

Para estos tres procesos mencionados anteriormente, los programas que he puesto sobre la mesa a la hora de decidir cuál escoger son los mismos que mencioné sobre el modelado 3D (3Ds Max, Blender y Maya).

En este caso, voy a escoger **Autodesk Maya**, es el software que más se utiliza en la actualidad para ello y, además, es con el que tengo más experiencia, sobretodo en *rigging* y *skinning*.

Por otro lado, para las animaciones más complejas las descargaré directamente desde la página web de Adobe, **Mixamo**, la cual posee una gran base de datos de animaciones de gran calidad.



FIGURA 37. LOGO DE MIXAMO

5.1.6 Sistemas de red para multijugador.

A la hora de dotar de multijugador online a un videojuego, será necesario utilizar un sistema de red que permita esto.

5.1.6.1 UNET (Unity Networking)

Sistema de red propio del motor gráfico Unity, que facilita en gran medida la configuración de dicho sistema evitando tener que preocuparse acerca de los detalles de implementación de “bajo nivel”.

Los videojuegos tienen un servidor y múltiples clientes. No existe un servidor dedicado, ya que uno de los clientes juega el rol de servidor, que recibe el nombre de Host o anfitrión.

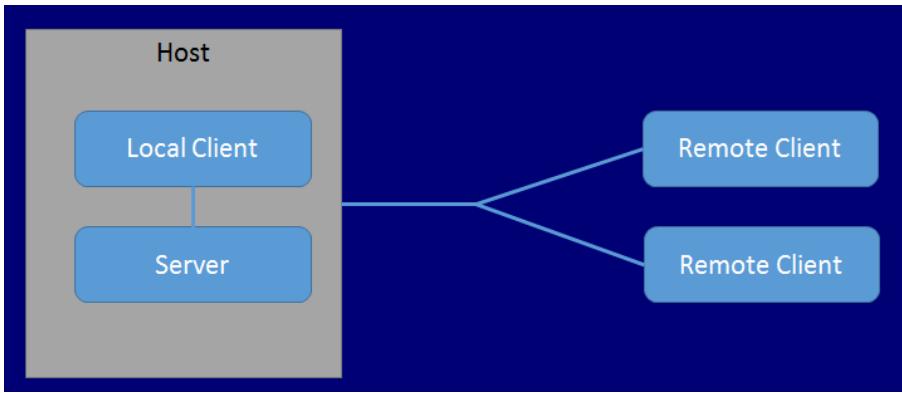


FIGURA 38. CLIENTE-SERVIDOR EN UNET

El anfitrión es un servidor y un cliente en el mismo proceso. Una ventaja que tiene este sistema de red es que el código para los anfitriones y los clientes es el mismo, da tal manera que los desarrolladores solo tienen que pensar acerca de un tipo de cliente la mayoría de veces.

Este sistema de red cuenta con una muy amplia documentación en la misma página y foro de Unity, además de muchos tutoriales en todo internet.

UNET es gratuito solo hasta 20 jugadores simultáneos, y permite muy poco ancho de banda (4k/s + 2 minutos iniciales de ancho de banda libre).

5.1.6.2 PUN (Photon Unity Network)



FIGURA 39. PHOTON UNITY NETWORK

Sistema de red alternativo UNET y ajeno a Unity, que tiene como objetivo facilitar la implementación de un sistema de red en un videojuego evitando trabajar a “bajo nivel”. Este sistema cuenta con funciones altamente similares a las funciones de UNET, de tal

manera que los usuarios familiarizados con el sistema UNET se encuentren cómodos a la hora de utilizar PUN.

Este sistema de red utiliza un sistema de habitaciones o *rooms*.

PUN permite utilizar su código multijugador en modos de juego offline. Por otra parte, éste cuenta con documentación bastante detallada desde su web y, siendo bastante usado, son muchos los usuarios que han publicado tutoriales por toda la red.

PUN es gratuito solo hasta 20 jugadores simultáneos.

5.1.6.3 Photon Bolt



FIGURA 40. LOGO PHOTON BOLT

Sistema de red muy potente pero complejo. Las conexiones no son fáciles de establecer y es más arriesgado para videojuegos de móviles (si el anfitrión cierra el juego, se acabó). Éste incluye una gran cantidad de mejoras con respecto a PUN y PUN+ (versión de pago de PUN) como por ejemplo una gran compatibilidad con la plataforma digital Steam.

Es totalmente de pago, aunque están trabajando en una versión gratuita.

5.1.6.4 uLink

Aparentemente abandonado por sus desarrolladores desde el año 2014. Librería construida para videojuegos creados en Unity. La API de uLink está diseñada para ser muy similar a UNET, escrito en lenguaje C#.

5.1.6.5 Forge Networking

Librerías de red desarrolladas como un plugin para Unity, las cuales se pueden utilizar una vez se haya comprado dicho plugin. No tiene límite de jugadores, y el precio es bastante caro.

5.1.6.6 TCP / UDP Sockets

La opción menos recomendable desde la aparición de las herramientas anteriores (a menos que se tenga gran experiencia), debido a su complejidad teniendo que programar a bajo nivel, incrementando así el tiempo de desarrollo del videojuego.

5.1.6.7 Herramienta seleccionada: UNET

De todas las herramientas anteriores las dos que más me convencieron fue UNET y PUN, gracias a la gran cantidad de documentación existente, además de contar ambos con versiones totalmente gratuitas que apenas limitan el objetivo de mi proyecto. Finalmente me decanté por **UNET** debido a que considero que existe una mayor documentación en internet que en el caso de PUN, aunque ambos me valdrían para realizar mi proyecto.

5.1.7 Plataformas de videojuegos digitales

En los últimos años han surgido una serie de plataformas de distribución de videojuegos en versión digital. Muchas de estas plataformas se consideran redes sociales debido a sus múltiples funciones que permiten formar comunidades y otras opciones sociales entre jugadores (como invitar a partidas o chatear, todo a través de perfiles de usuario).

Decidí aprovechar algunas de las características sociales de estas plataformas para mi videojuego, empezando por investigar cuales de éstas cuentan con una API para desarrolladores.

5.1.7.1 Steam



FIGURA 41. STEAM

Es una plataforma de distribución digital, gestión digital de derechos, comunicaciones y servicios multijugador desarrollada por Valve Corporation. Esta plataforma es utilizada tanto por pequeños desarrolladores independientes como grandes corporaciones de software para la distribución de videojuegos y material multimedia relacionado.

Steam, como parte de una red social, permite a los usuarios identificar amigos y unirse a grupos a través de la Comunidad Steam. Los usuarios pueden intercambiar mensajes de texto, identificar a qué es a lo que están jugando sus amigos y otros miembros de sus grupos y, para videojuegos que utilicen la API de **Steamworks**, unirse e invitar a sus amigos a partidas multijugador.

Steamworks es una API que cuenta con herramientas gratuitas que pueden complementar el multijugador de un videojuego otorgando una serie de opciones como Matchmaking, servicio de inventario de Steam, o almacenamiento en la nube. Esta API es totalmente compatible con el motor gráfico Unity.

La API Steamworks es compatible de forma nativa con los motores gráficos Unity, Unreal Engine, GameMaker Studio 2, Cryengine, Source 2013, Visionaire Studio, y Leadwerks Game Engine.



FIGURA 42. LOGO DE STEAMWORKS

5.1.7.2 Origin



FIGURA 43. LOGO DE ORIGIN

Origin es una plataforma de videojuegos digital desarrollada por Electronic Arts que permite gestionar los videojuegos comprados a dicha compañía a través de una cuenta de usuario en versión digital. Se trata de un sistema similar a Steam.

En esta plataforma el usuario cuenta con un registro de actividad y un listado de sus amigos, pudiendo ver la actividad de éstos.

Origin no cuenta con API para desarrolladores.

5.1.7.3 Google Play



FIGURA 44. LOGO DE GOOGLEPLAY

Plataforma de distribución digital de aplicaciones móviles para los dispositivos con sistema operativo Android. En relación con los videojuegos, esta plataforma cuenta con un servicio que recibe el nombre de **Play Juegos**, compatible con Android, iOS y web, la cual añade la opción de multijugador en tiempo real, logros, tabla de posiciones y de guardar información en la nube de los videojuegos que sean compatibles con este servicio.



Pizacore

FIGURA 45. LOGO DE PLAY JUEGOS

Esta plataforma cuenta con una API para desarrolladores, pero no tiene lugar en mi proyecto puesto que éste se trata de un videojuego de PC.

5.1.7.4 Uplay



FIGURA 46. LOGO DE UPLAY

Plataforma de distribución digital, gestión de derechos digitales, multijugador y comunicaciones desarrollado por Ubisoft Massive. Esta plataforma otorga una serie de servicios que permite proporcionar de logros a los videojuegos, crear amistades entre los jugadores, y una gestión en general de los videojuegos exclusivos de Ubisoft.

Esta plataforma con cuenta con API para desarrolladores.

5.1.7.5 Herramienta seleccionada: Steam

Después de este breve análisis sobre algunas de las plataformas de videojuegos digitales más conocidas, mi única opción es la de utilizar la API de **Steamworks**, puesto que es la única que puedo utilizar, aunque desde luego no me puedo quejar, pues la plataforma de **Steam** es la más utilizada en la actualidad y con una enormísima diferencia frente al resto.

5.1.8 Resumen de herramientas que se van a utilizar

El siguiente listado resume el conjunto de herramientas que voy a emplear para la elaboración de este proyecto:

- **Motor de videojuegos:** Unity. Aprender a utilizar este motor gráfico forma parte de mis objetivos con este proyecto.
- **Software de modelado:** Autodesk 3ds max. Es el programa de modelado con el que tengo más práctica.
- **Software de texturizado:** Autodesk 3ds max. Igual que el punto anterior.
- **Programas de dibujo y edición gráfica:** Adobe Photoshop y PaintTool SAI. Para logos, menús y HUD el primero, y el segundo para dibujar con tableta digital. Son los dos programas con los que más experiencia tengo para este tipo de tareas.
- **Sistema de red para multijugador:** UNET. Sistema de red que cuenta con más documentación en internet.
- **Plataformas de videojuegos digitales:** Steam. Única plataforma que cuenta con una API de desarrollo, además de ser la plataforma más utilizada actualmente, con mucha diferencia frente al resto.

5.2 Documento de diseño del Juego (GDD)

El documento de diseño del juego (*Game Design Document*, en inglés) es un documento descriptivo donde se comentan todas las características de un videojuego, desde su historia y personajes, hasta sus mecánicas y jugabilidad.

5.2.1 Introducción

5.2.1.1 Título

El nombre elegido para el videojuego es **Islands of Crafts**. Se trata de un videojuego de supervivencia en el cual cada jugador se reencarnará en un muñeco de papel, y tendrán que intentar sobrevivir el máximo número de días posibles en una isla donde todo estará formado por materiales típicos de papelería como cartón, papel o plástico.



FIGURA 47. LOGO DEL VIDEOJUEGO

5.2.1.2 Desarrolladores

Videojuego desarrollado de manera individual por Aitor Vidal Arnau, alumno del grado de Ingeniería Multimedia en la Universidad de Alicante.

El desarrollo del videojuego se ha llevado a cabo en una máquina con las siguientes características:

- Sistema Operativo: Windows 10 Pro
- Procesador: Intel® Core™ i7-7700HQ CPU @ 2.80GHz
- Memoria RAM: 16,0GB
- Tarjeta gráfica: NVIDIA GeForce GTX 1060

5.2.1.3 Argumento

Somos un muñeco de papel en mitad de una isla desierta formada por materiales típicos de papelería, tales como papel o cartón. Tendremos que prestar atención al estado de salud de nuestro personaje, puesto que éste contará con el típico indicador de vida, además de un indicador de *nivel de color* (fácilmente relacionable con el indicador típico de *hambre* en videojuegos del mismo género), teniendo que alimentarse para mantener este último en condiciones, y sobrevivir finalmente.

Nuestro objetivo será el de sobrevivir el máximo número de días posibles en dicha isla, recolectando y administrando bien nuestros recursos y construyendo elementos que nos ayudarán en la supervivencia. Sin embargo, esto no será tarea fácil puesto que determinados sucesos tratarán de impedirlo.

5.2.1.4 Características principales

Las características más destacables de este videojuego son las siguientes:

- Videojuego de aspecto *cartoon*, muy colorido, aunque con un toque ligeramente realista en las texturas gracias a los mapas de normales.
- Desarrollado con Unity 3D.

- Perfectamente explorable, el jugador podrá acceder a cualquier parte de la isla con total libertad, sin la posibilidad de salir de esta.
- Mapas totalmente aleatorios en cada una de las partidas.
- Posibilidad de jugar solo o con amigos.
- Mecánicas sencillas.

5.2.1.5 Género

Islands of Crafts es un videojuego que pertenece al género de videojuegos de supervivencia o *survival* en inglés.

El objetivo primordial de este tipo de videojuegos es uno: no morir. Generalmente, en un videojuego de supervivencia nos encontraremos en un entorno hostil, donde tenemos que buscar objetos que nos permitan fabricar elementos que nos ayuden a defendernos, alimentarnos, refugiarnos...

Algunos de los videojuegos de este mismo género son *Don't Starve*, *Rust*, o el modo supervivencia de *Minecraft*.

5.2.1.6 Plataforma

Videojuego desarrollado en Unity 3D exclusivo para PC.

5.2.1.7 Público

El videojuego va dirigido principalmente a aquellos que encuentren atractivo este género de videojuegos, ya que, de lo contrario, se podrían aburrir rápidamente.

No hay un rango de edad específico; se podría decir que este videojuego es para todos los públicos, ya que éste no contiene elementos sensibles a ningún colectivo.

5.2.1.8 Clasificación PEGI



FIGURA 48. LOGO DE CLASIFICACIÓN PEGI

PEGI son las siglas de *Pan European Game Information* (información paneuropea sobre videojuegos, en español). Este es un sistema europeo para clasificar el contenido de los videojuegos y otro tipo de software de entretenimiento.

Existen dos formas de clasificación para cualquier software; una de edad sugerida y otra sobre seis descripciones de contenido, tales como el uso de lenguaje soez, violencia o miedo.

Sobre la edad sugerida del videojuego que trata este proyecto, según este sistema de clasificación, sería **a partir de 3 años**.



FIGURA 49. ETIQUETA PEGI 3

Y en cuanto a las descripciones de contenido, el único que se correspondería con este videojuego es el de la etiqueta de **Online, pudiendo jugar hasta un máximo de 4 jugadores de manera simultánea**.



FIGURA 50. ETIQUETA PEGI ONLINE

5.2.1.9 Objetivo

Como ya se ha mencionado anteriormente, el objetivo de este videojuego será el de sobrevivir el máximo número de días posibles en una isla hostil.

Administrar bien nuestros alimentos y construir diferentes elementos útiles será la clave de la supervivencia. Sin embargo, habrá una serie de sucesos y elementos que tratarán de impedir la supervivencia de los jugadores, tales como eventos atmosféricos o la misma fauna de la isla.

5.2.1.10 Apariencia

Uno de los objetivos de este proyecto es el de aportar una apariencia de tipo *cartoon* al videojuego.

Esta apariencia caricaturesca tendrá un toque original; irá acompañada de un estilo ligeramente realista debido a que tratará simular materiales de típicos de manualidades (papel, cartón...) con todos y cada uno de los elementos del videojuego y, a su vez, todo se verá especialmente colorido.

Mi principal referencia para lograr este tipo de gráficos ha sido la saga de videojuegos *Paper Mario*, de Nintendo.



FIGURA 51. IMAGEN DE PAPER MARIO: COLOR SPLASH

5.2.2 Mecánicas y jugabilidad

5.2.2.1 Jugabilidad

La jugabilidad será similar a la de todos los videojuegos de supervivencia de este género. Los jugadores cuentan con dos parámetros: **vida** y **nivel de color** (el **nivel de color** será equiparable al **hambre** en videojuegos de su mismo género), representados por una barra cada uno, la cuales serán visibles sobre los avatares de los jugadores.

Por otra parte, cada uno de los jugadores contará con un inventario de un tamaño máximo de 9 objetos. Los objetos serán apilables, excepto los objetos fabricables (armas, armaduras, y refugios).

Así pues, los jugadores podrán explorar sin límite la isla desierta en busca de recursos que les permitan **alimentarse, crear un refugio, o crear un equipo con el que poder defenderse** de diferentes elementos hostiles, como la fauna o fenómenos atmosféricos.

5.2.2.2 Personajes

En este apartado voy a describir cada uno de los personajes del videojuego, pudiéndolos diferenciar entre personaje jugable (personaje principal o protagonista) y personajes no jugables (la fauna de la isla). Cabe destacar que los animales que componen la fauna de la isla serán de colores aleatorios, alternando entre rojo, celeste, verde, azul, amarillo, y rosa.

➤ **Muñeco de papel.** Es el personaje principal, y cada uno de los jugadores será representado por uno de estos. Cada muñeco de papel será de un color distinto y nunca habrá más de un jugador del mismo color. Cada jugador podrá elegir el color de su personaje antes de empezar la partida, durante el menú de creación de la partida. Solo se podrá elegir entre seis colores: rojo, celeste, verde, azul, amarillo, y rosa.



FIGURA 52. EJEMPLO DE MUÑECO DE PAPEL

- **Conejo de papel.** Animal inofensivo, forma parte de la fauna de la isla. Habrá conejos de diferentes colores. Estos animales simularán estar hechos mediante el arte del *origami* o papiroflexia. Al eliminarlos soltarán papel de su color.



FIGURA 53. EJEMPLO DE CONEJO DE PAPEL

- **Pollito de cartón.** Animal inofensivo, forma parte de la fauna de la isla. Habrá pollitos de diferentes colores. Estos animales simularán estar hechos con hueveras de cartón. Al eliminarlos soltarán cartón de su color.



FIGURA 54. EJEMPLO DE POLLITOS DE CARTÓN

- **Cerdo de plástico.** Animal inofensivo, forma parte de la fauna de la isla. Habrá cerdos de diferentes colores. Estos animales simularán estar hechos con botellas de plástico. Al eliminarlos soltarán plástico de su color.



FIGURA 55. EJEMPLO DE CERDO DE PLÁSTICO

- **Tijeras de plástico.** Forman parte de la fauna de la isla, se trata de un objeto animado con carácter agresivo. Habrá tijeras de diferentes colores. Al eliminarlos soltarán plástico de su color.

5.2.2.3 Objetos

En el videojuego habrá una serie de objetos que ayudarán a la supervivencia de los jugadores. Parte de estos objetos (únicamente los fabricables) cuentan con un parámetro fundamental: **el nivel de color. El nivel de color será el equivalente a la resistencia del objeto**, además determinará el color de los objetos (rojo, celeste, verde, azul, amarillo, y rosa). El **nivel de color** disminuirá con el agua de la lluvia, al sufrir daño (armaduras), o con un uso continuado del objeto (armas). **Cuando el nivel de color llegue a 0, el objeto será destruido.** Los objetos que cuentan con el parámetro de **nivel de color** serán fabricables por los propios jugadores a través de los diferentes recursos que recolecten, siendo estos recursos, también, de diferentes colores. Además, **sólo se podrán juntar recursos de un mismo color para crear un objeto concreto.**

Podemos distinguir **tres tipos de recursos**. Cada uno de estos recursos tiene una **calidad** diferente, influyendo de manera directa sobre el **nivel de color del objeto creado.**

- **Papel.** Es el recurso más básico y de más baja calidad. Es el más fácil de conseguir debido a la abundancia de vegetación de papel que existe, pero a su vez se necesitan grandes cantidades para fabricar objetos de este material. Por otra parte, el papel es **el único recurso que los jugadores podrán ingerir**. **El papel se puede conseguir eliminando conejos, talando pinos y palmeras, y recolectando flores**. Los objetos creados con **papel** tendrán un **nivel de color bajo**.
- **Cartón.** Recurso de calidad media. Es más complicado de conseguir que el papel, debido a que hay menos vegetación de este material. **El cartón se puede conseguir eliminando pollitos, y talando fresnos y arbustos**. Los objetos creados con **cartón** tendrán un **nivel de color medio**.
- **Plástico.** Recurso de calidad alta. Es el más complicado de conseguir, puesto que no existe vegetación con este material. **Solo se puede conseguir eliminando cerdos y tijeras**. Los objetos creados con **plástico** tendrán un **nivel de color alto**.

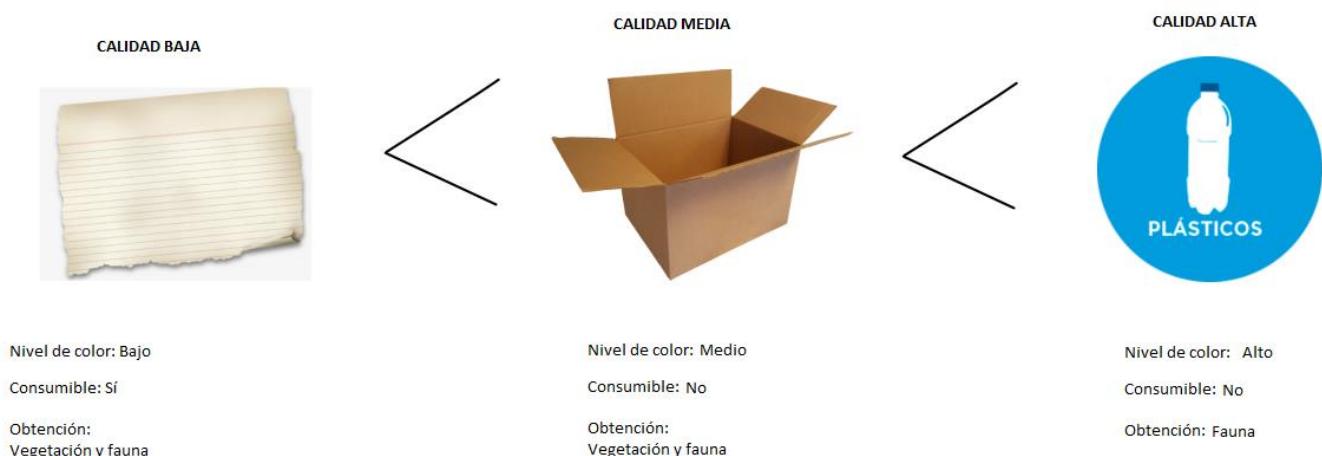


FIGURA 56. DIFERENCIAS ENTRE MATERIALES

Mediante los recursos mencionados anteriormente será posible fabricar los siguientes **objetos**, los cuales tendrán un **nivel de color** determinado en función de la **calidad del recurso** con el que se crearon:

- **Espada:** Arma con la que se pueden talar árboles o eliminar animales rápidamente. El **nivel de color** del objeto se verá **disminuido al atacar**. Como

característica adicional, cuanto mejor sea la calidad del objeto, más daño causará a árboles y animales.

- **Armadura:** Los jugadores se podrán equipar con una armadura, de esta manera el jugador sufrirá menos daño a golpes de enemigos. El **nivel de color** de la armadura se verá **disminuida** cada vez que el **jugador sufra daños**.
- **Refugio:** Los jugadores serán capaces de fabricar refugios con los que podrán resguardarse de la lluvia. El **nivel de color** de los refugios disminuirá de forma progresiva con la lluvia. En el momento en el que un refugio tenga un nivel de color igual a 0, éste será destruido.

A parte de los objetos mencionados anteriormente, existen otros objetos que **no serán fabricables** y la única manera de obtenerlos será encontrándolos de forma aleatoria por la isla, aunque serán raros de encontrar. Se trata de objetos consumibles de un solo uso:

- **Bote de pintura:** Rellena el nivel de color de un jugador al máximo. La **probabilidad** de encontrar este objeto es **baja**.
- **Cinta adhesiva:** Rellena la vida de un jugador al máximo. La **probabilidad** de encontrar este objeto es **muy baja**.
- **Bote de pintura fosforecente:** Otorga iluminación temporal al cuerpo del muñeco de papel, útil para ver mejor por la noche. La **probabilidad** de encontrar este objeto es **mayor que los dos anteriores**.

5.2.2.4 Acciones del jugador

- Moverse libremente a lo largo de toda la isla, sin ningún tipo de límite.
- Golpear y eliminar cualquier elemento de la fauna y flora, y así recolectar sus recursos.
- Saltar.
- Fabricar objetos mediante los recursos.
- Arrojar al suelo objetos del inventario.
- Equiparse objetos.
- Usar objetos consumibles.
- Invitar amigos de *Steam* a la partida (antes de que ésta comience).

- Elegir el color del personaje con el que se va a jugar la partida.

5.2.2.5 Controles

- **WASD** para moverse. W y S para avanzar hacia adelante y hacia atrás, y A y D para avanzar hacia izquierda y derecha.
- **El movimiento del ratón** permitirá moverla cámara en todas direcciones.
- **Las teclas 1, 2, 3, 4, 5, 6, 7, 8, y 9** permiten equipar **en mano** cualquier objeto del inventario.
- Al presionar **clic derecho del ratón** con un objeto equipado en mano, se realizará la acción permita el objeto, por ejemplo, si es un papel se consumirá, si es una armadura se equipará sobre el cuerpo del personaje, etc.
- Si se presiona **clic izquierdo** el personaje atacará con los puños; en el caso de tener una espada equipada, golpeará con ella y causará más daño.
- **Con un objeto equipado en mano**, si se presiona la **tecla Q** se dejará caer el objeto al suelo.
- Con el **clic de la rueda del ratón**, si se tiene equipado algún objeto, este se desequipará.
- **Mantener Shift** para mostrar el ratón.
- Mientras se muestre el ratón, **clic izquierdo** sobre el ícono de un objeto del menú de fabricación para fabricar un objeto.
- **Tecla espacio** para saltar.

5.2.2.6 Progresión

Al tratarse de un videojuego no lineal y de mundo abierto, la progresión de la partida viene marcada por el paso de los días, y el constante intento de sobrevivir por parte de los jugadores consiguiendo recursos que les ayuden a ello.

5.2.2.7 Mecánicas

- **Destruir elementos** (vegetación o fauna). Al destruir estos elementos, **soltarán recursos** que caerán al suelo directamente. El jugador podrá **recoger estos recursos** simplemente al colisionar con ellos.
- **Fabricar objetos** mediante los recursos del inventario. Cada objeto tendrá una **cantidad requerida** de un recurso concreto **para fabricarlo**. El *sprite* de los objetos que se puedan fabricar aparecerán en el HUD de forma opaca, mientras que el *sprite* de aquellos objetos que no se puedan fabricar todavía aparecerán con transparencia.
- **Equipar objetos.** Al presionar las teclas correspondientes para equipar un objeto, éste **se equipará siempre en mano**. De esta forma, el objeto aparecerá sobre la mano del personaje y podrá usarlo, desequiparlo o arrojarlo al suelo.
- **Golpear con los puños.**
- **Saltar.**
- **Arrojar al suelo el objeto que tengas equipado en mano.** Es importante destacar que si se arroja un ítem fabricable (espada, armadura, o refugio) éste se romperá y, en el suelo, aparecerá una parte de los recursos empleados para su fabricación. Es una forma de reciclar un ítem antes de que se termine de destruir.
- **Vida del personaje. Cuando la vida se vacíe por completo, es decir, llegue a 0, el personaje morirá.** Si el personaje **muere** se **termina la partida**. En el caso de estar jugando con **amigos**, **si uno de los jugadores muere, la partida terminará también**, siendo obligatoria la supervivencia de todos los miembros del grupo, incentivando la cooperación de todos. **La vida disminuirá al sufrir daño por parte de un animal**, o si el **nivel de color** está a 0; en este último caso la vida irá disminuyendo de forma progresiva mientras no haya nivel de

color. En el momento en el que el jugador recupere algo de nivel de color, la pérdida progresiva de vida se detendrá. Será posible **recuperar toda la vida mediante el objeto cinta adhesiva, o una pequeña parte de la vida consumiendo papel.**

- **Nivel de color del personaje.** El nivel de color es como si fuese **el hambre** del personaje. El **nivel de color** irá disminuyendo a lo largo de la partida de forma progresiva (disminuirá más rápido si entra en contacto con la lluvia) y, en el momento en el que el **nivel de color llegue a 0, el personaje comenzará a perder vida rápidamente**. El jugador podrá **restaurar** parte de su **nivel de color** si se **alimenta de papel de su mismo color, o todo el nivel de color al consumir un objeto bote de pintura**. Si el papel que consuma **no se corresponde con su color**, se le **penalizará** disminuyendo **una pequeña parte de su nivel de color, pero recuperará algo de vida igualmente**.
- **Ciclo día-noche.** Un **día completo** en el juego **durará 5 minutos aproximadamente (288 segundos exactamente)**. A lo largo del día todo transcurrirá con normalidad, sin embargo, conforme se acerque la **noche** (sobre las 19:00 de la tarde), empezará a oscurecer, hasta el punto de verse todo completamente oscuro; además, entre las 00:00 y las 06:00 **todos los animales de la fauna se volverán agresivos**, siendo ese período de tiempo el más peligroso del día.
- **Otorgar iluminación.** Al consumir el objeto bote de pintura fosforescente, el cuerpo del personaje emitirá luz, otorgando así iluminación temporal ideal para ver mejor en la oscuridad de la noche.
- **Lluvia.** De forma totalmente **aleatoria**, podrá llover en el juego. Solo podrá llover una vez al día, y la lluvia podrá alargarse durante varias horas. La lluvia será **nociva para los jugadores**, haciendo que su **nivel de color disminuya de**

forma más rápida. Para **protegerse** de la lluvia, lo ideal será fabricar un **refugio**.

5.2.2.8 Mapa o escenario

El videojuego tiene lugar en una isla desierta, rodeada por un mar.

Los mapas del videojuego serán **totalmente aleatorios**, concretamente procedurales. Entendemos por generación procedural aquella donde los contenidos no están diseñados de antemano ni vienen almacenados en el propio juego, sino que **se crean de manera aleatoria** en base a una serie de algoritmos definidos en la programación del videojuego.

De esta manera, la forma del terreno, los puntos de *spawn* de animales y vegetación, la localización de objetos esparcidos por el mapa, e incluso el texturizado del terreno se generará de manera procedural. El texturizado del mapa se realizará en función de la altura del terreno, pudiendo distinguir entre 3 tipos de texturas, siendo arena la parte más baja, hierba la parte media, y tierra/piedra la parte alta.

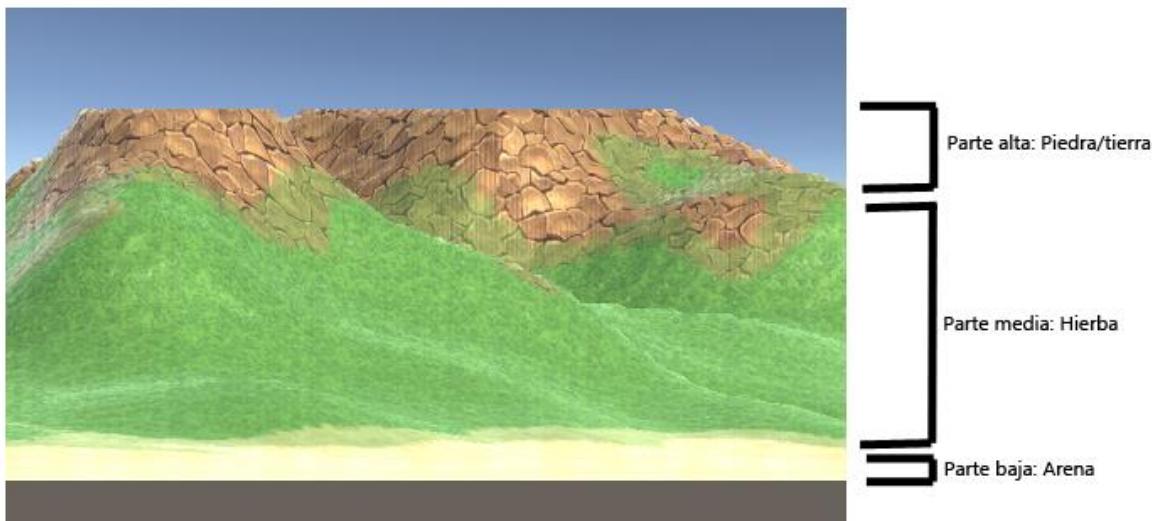


FIGURA 57. TEXTURIZADO EN FUNCIÓN DE LA ALTURA DEL TERRENO

Para la generación aleatoria de la localización de los animales, vegetación, y objetos se tendrá en cuenta la altura del terreno, de tal forma que algunos elementos se encontrarán únicamente en determinadas alturas del terreno. De esta misma forma, el texturizado del terreno irá en función de la altura del terreno, siendo las zonas más bajas de tierra/arena, las zonas intermedias de césped, y las zonas más altas de piedra o nieve.

Los mapas contarán con una vegetación variada, y de colores aleatorios comprendidos entre rojo, verde, azul, celeste, amarillo, y rosa.

- **Pinos.** Al destruirlos sueltan papel de su color. Se encuentran en zonas de altura media.



FIGURA 58. EJEMPLO PINO DE PAPEL

- **Palmeras.** Al destruirlas sueltan papel de su color. Se encuentran en zonas de altura baja.



FIGURA 59. EJEMPLO DE PALMERAS DE PAPEL

- **Flores.** Al destruirlas sueltan papel de su color. Se encuentran en zonas de altura media.



FIGURA 60. EJEMPLO FLORES DE PAPEL

- **Fresnos.** Al destruirlos sueltan cartón de su color. Se encuentran en zonas de altura alta.



FIGURA 61. EJEMPLO FRESNO DE CARTÓN

- **Arbustos.** Al destruirlos sueltan cartón de su color. Se encuentran en zonas de altura media y alta.



FIGURA 62. EJEMPLO ARBUSTO DE CARTÓN

Como se mencionó anteriormente, el recurso **plástico** no es obtenible directamente de la vegetación.

5.2.2.9 Eventos

Los diferentes eventos que pueden dar lugar en este videojuego son los ya mencionados anteriormente: El **ciclo de día-noche** y la **lluvia**.

5.2.2.10 Flujo del juego

El flujo del juego funcionaría de la siguiente manera:

1. Jugador crea la partida (o se une a una partida que esté en proceso de creación).
2. Se elige el color con el que se desea jugar.
3. Se presiona el botón de *Ready* para indicar que el jugador está listo.
4. En el momento en el que todos los jugadores se encuentren listos, comienza una cuenta atrás para dar comienzo a la partida.
5. Una vez dentro, son las 09:00 de la mañana, los jugadores buscan recursos por toda la isla.
6. Con los recursos fabrican espadas, refugios, y armaduras, además guardan en su inventario papel para consumir en caso de que sea necesario.
7. Al llegar la noche, los jugadores consumen los botes de pintura fosforescente para ver iluminar sus cuerpos en la oscuridad, y poder evitar entrar en contacto con animales (los cuales son agresivos durante la noche).
8. Una vez amanece, se repite el flujo hasta que alguno de los jugadores muera.

5.2.3 Interfaz

5.2.3.1 Menú principal

En el menú principal se encontrarán **dos opciones**, la primera para crear la partida (se tiene la opción de escribir el nombre de la sala o partida), y la segunda opción será para unirte a otra partida que se encuentre en proceso de creación.

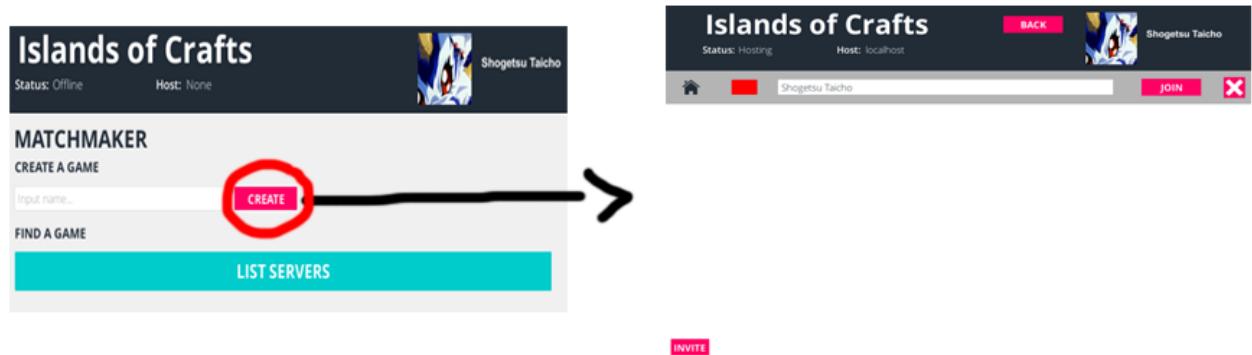


FIGURA 63. TRANSICIÓN CREAR PARTIDA

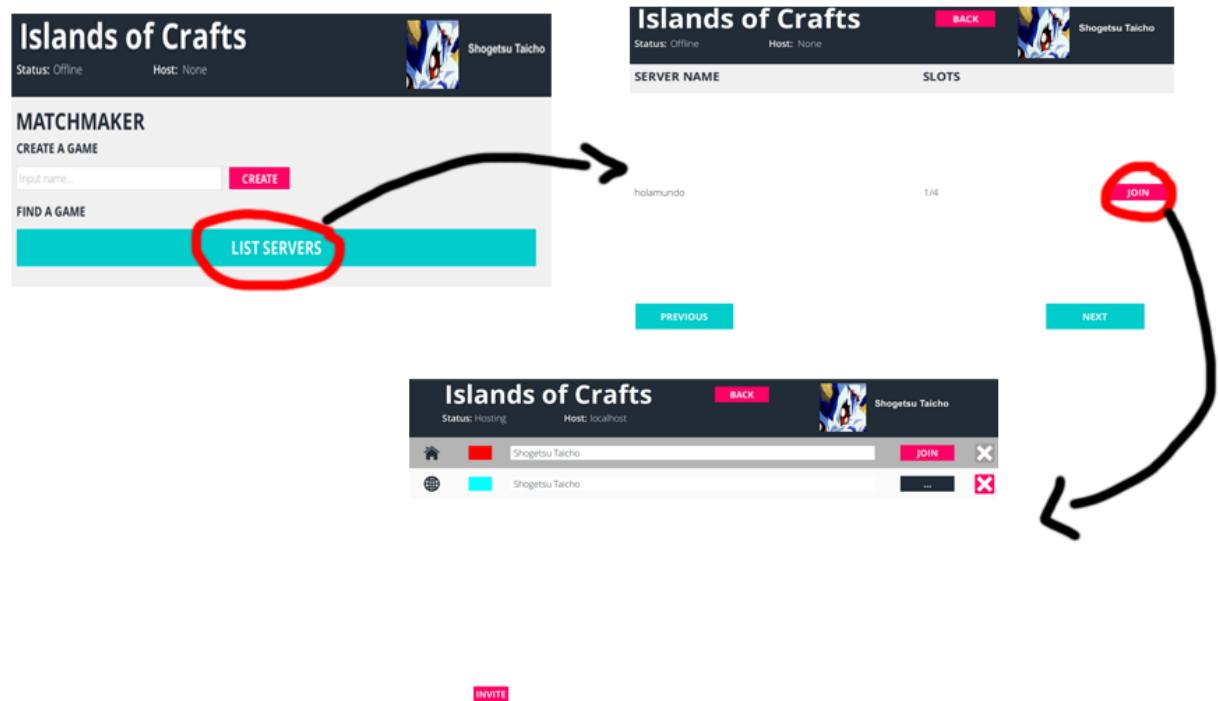


FIGURA 64. TRANSICIÓN UNIRSE A PARTIDA

5.2.3.2 Menú Lobby

En el menú de Lobby se da la opción de cambiar el color que representará tu personaje durante la partida. Además, un jugador podrá abandonar la sala o partida en creación.

Por otro lado, en la parte inferior, se encontrará un botón que sirve para invitar a los amigos a la partida.

5.2.3.3 Menú buscar partida

En el menú de buscar partida, aparecerá un listado con las partidas que se encuentran en creación (o que ya se están jugando). A las salas donde la partida ya comenzó el jugador no podrá unirse.

En el listado mencionado, aparecerá el nombre de la sala, además del número de jugadores en ella, y un botón para unirte a la sala.

5.2.3.4 Menú invitar amigo

Al presionar sobre el botón *Invite* que se encuentra en el lobby, se abrirá una ventana emergente de **Steam** permitiendo invitar a los amigos que se encuentren en tu lista de amigos, siempre y cuando éstos tengan el videojuego instalado, de lo contrario no podrán unirse, a pesar que acepten la invitación.

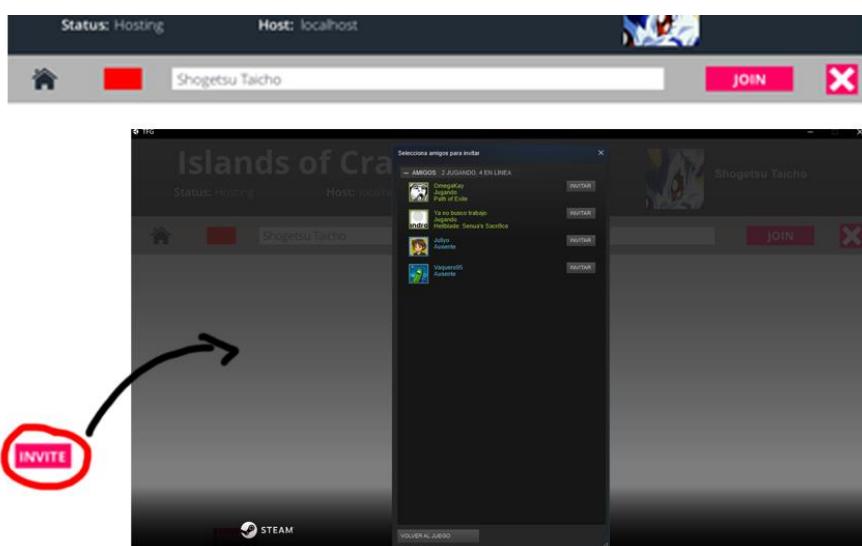


FIGURA 65. TRANSICIÓN INVITAR AMIGO DE STEAM

5.2.3.5 HUD

El HUD que se mostrará durante la partida estará compuesto por:

- **Un inventario.** El inventario se encontrará en la parte inferior de la pantalla. Estará formado por 9 casillas. En cada una de estas aparecerán los sprites de los ítems que el jugador tenga en su posesión. Cuando el ítem se encuentra equipado en mano, aparecerá un marco rojo indicando qué ítem se encuentra equipado. Cuando el ítem es una armadura, y se encuentra equipada en el cuerpo (se usó con clic derecho) aparecerá un marco verde indicando que esa armadura se encuentra equipada. Conforme la espada o la armadura se vayan desgastando (el nivel de color de estos ítems disminuye por su uso), el *sprite* de estos ítems se irá volviendo cada vez más oscuro, hasta llegar al punto de romperse (y desaparecer del inventario).
- **Menú de fabricación.** En la parte superior izquierda se encuentra el menú de fabricación, donde los *sprites* de los ítems, que en ese instante sean fabricables, aparecerán de manera opaca (de lo contrario, se mostrarán con transparencia). Junto a cada uno de esos *sprites*, se encuentran 6 botones que se corresponden a los 6 colores que ya se mencionaron anteriormente. Estos botones funcionan para fabricar un ítem del color que se deseé, siempre y cuando se tengan en el inventario los recursos necesarios. Cuando un ítem sea fabricable de un color determinado, el botón de dicho color se mostrará activo, pudiendo hacer clic sobre él.
- **Hora del día, y el día.** En la esquina superior derecha de la pantalla se mostrará la hora del día exacta, además del día en el que nos encontramos.
- **Vida, nivel de color y nombre del jugador.** Sobre los personajes se encontrará el nivel de color del personaje actual, además de su vida y su **nombre de Steam**.

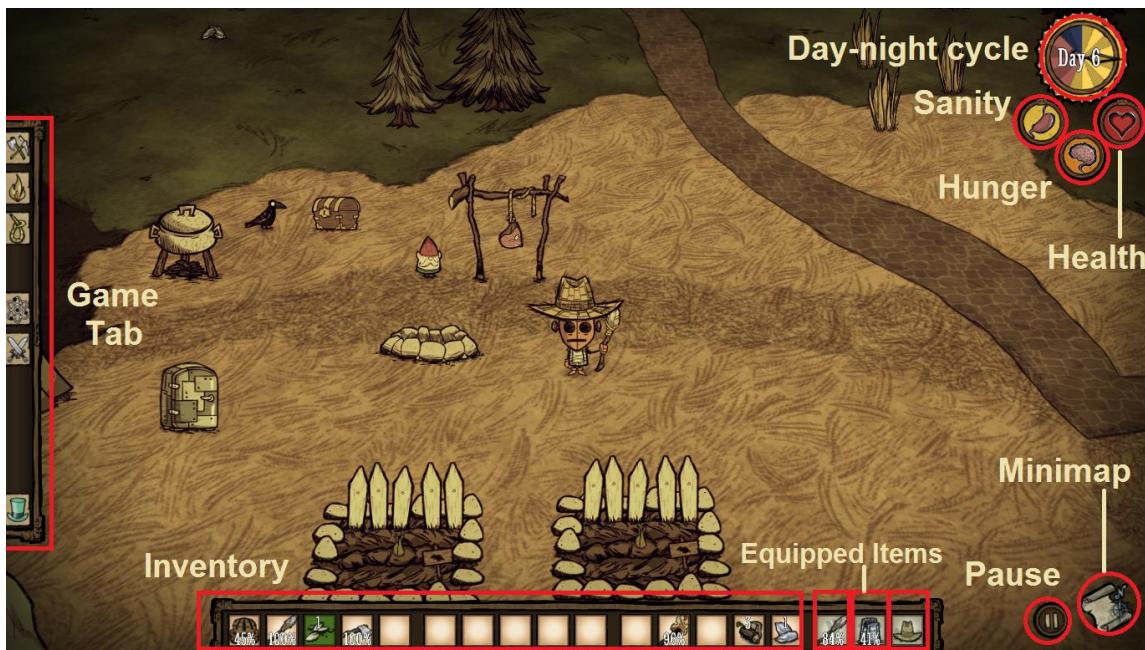


FIGURA 66. EJEMPLO DE HUD MUY SIMILAR

5.2.3.6 Menú partida terminada

Al terminar la partida, se mostrará una ventana indicando el número de días sobrevividos, además de un botón para volver al menú principal.

5.2.4 Cámara

La cámara será en tercera persona, situándose detrás del jugador.

Se podrá mover en todas direcciones casi sin límite. Los límites de la cámara vendrán establecidos con el hecho de no poder acercar la cámara hasta el punto de atravesar al jugador (al bajar la cámara y hacer que colisione con el suelo), y no poder dar un giro completo al jugador desde arriba.

Por otro lado, la cámara tendrá efectos de postprocesado, tales como *Blurr*, para hacer el videojuego más vistoso.

5.2.5 Inteligencia Artificial

La inteligencia artificial de los animales será muy sencilla, puesto que este no es el objetivo principal de este proyecto.

Los animales podrán moverse de manera aleatoria dentro de un radio a su alrededor.

Cuando un animal inofensivo (todos excepto las tijeras) se sienta atacado, comenzará a huir de su atacante hasta perderlo de vista.

Cuando llega la noche (entre las 00:00 y las 06:00), todos los animales obtienen un carácter agresivo. Cuando un animal está agresivo, comenzará a perseguir y atacar al primer jugador que aviste. Los animales tienen un rango determinado de visión. Las tijeras son el único animal agresivo las 24h.

5.2.6 Música y efectos de sonido

En cuanto a la música, se escuchará una melodía alegre durante el día, otra algo más lenta durante la tarde, y otra más lenta y relajada durante la tarde-noche. Una vez sean las 00:00 solo se escucharán efectos de sonido ambientales, tales como grillos, hasta las 06:00.



FIGURA 67. MOMENTOS DEL DÍA

Por otra parte, el juego también tendrá efectos de sonido al caminar, golpear, recolectar recursos, y al llover.

5.2.7 Lenguaje

Este videojuego no cuenta con apenas textos; sin embargo, las pocas palabras que se muestran se aparecerán en **inglés**.

5.3 Desarrollo e implementación del videojuego

Para el desarrollo e implementación del proyecto, se ha establecido una serie de clases y objetos que componen la arquitectura del videojuego.

Al tratarse de un videojuego online, se han creado dos escenas en Unity, una **escena offline** y una **escena online**, de tal forma que en cada una de las escenas se han implementado clases y objetos diferentes.

En primer lugar, en cuanto a la **escena offline**, los objetos que encontramos son los siguientes:

- **LobbyManager**: Objeto encargado de la gestión del **Lobby**, para la creación de partidas y un sistema de buscar y/o unirse a estas. También, tiene un listado de objetos que puede generarse (*spawnable objects*) en el servidor. Además, **LobbyManager** genera los *prefab* de **Player** en el momento en el que empieza la partida.
- **SteamManager**: Este objeto se encarga de añadir funciones de la API de Steam al **Lobby**. Se ha utilizado para implementar el **sistema de invitación** de amigos de Steam, y mostrar el **nombre/avatar del jugador** en el **Lobby**.
- **EventSystem**: Objeto necesario para dar la capacidad al ratón de interaccionar con los distintos botones del **Lobby**.
- **CanvasSteamProfile**: Objeto que se utiliza simplemente para mostrar el nombre y el avatar de Steam del jugador, en el menú principal.

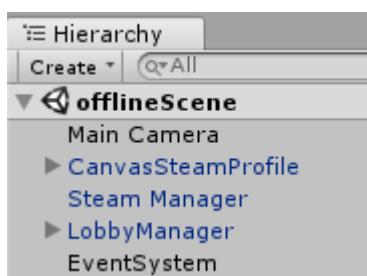


FIGURA 68. OBJETOS DE LA ESCENA OFFLINE

Por otro lado, en cuanto a la **escena online**, los objetos implicados en aquí son los que se muestran en el listado a continuación:

- **GameManager:** Este objeto es el encargado del flujo del juego, concretamente de la gestión del ciclo día-noche, de incrementar el número de días sobrevividos por los jugadores, además del sistema de lluvia.
- **AudioManager:** Objeto encargado de la gestión de los efectos de sonidos y música que se ejecutan a lo largo del juego.
- **EventManager:** Objeto que se encarga de hacer posible la conexión entre el inventario y el menú de fabricación.
- **MapManager:** Su trabajo es el de generar el terreno procedural, y el *spawn* de todos los elementos (vegetación, fauna...), además de prepararlo para la IA. Los ***prefab* de la fauna y la vegetación** (*RabbitPrefab*, *Flower*, etc) contienen *scripts* que se encargan de gestionar la vida, IA, *drop*, entre otras características.
- **Canvas:** Se trata del HUD, el cual está compuesto por el menú de *crafeo*, el inventario, el número de días transcurridos, y la hora del día actual.
- **Sea:** Objeto que se encarga de generar el mar y animarlo.
- **DirectionalLight:** Luz direccional de la escena. El objeto **GameManager**, mediante un *script*, se encarga de manipularlo para la gestión del ciclo día-noche.

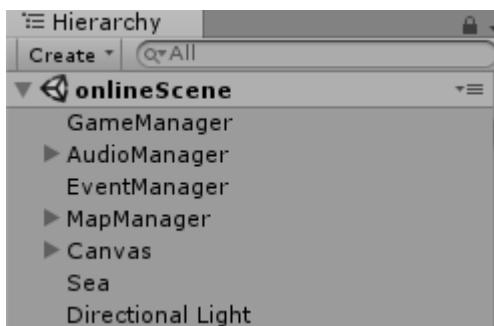


FIGURA 69. OBJETOS DE LA ESCENA ONLINE

Como he mencionado anteriormente, en el momento en el que comienza la partida se generan todos los *GameObject* de **Player** y de los elementos **NPC** (fauna y vegetación), a los cuáles se les añadirá la palabra (**Clone**) al nombre de sus correspondientes *prefab*.

```

► Player(Clone)
  TerrainPrefab(Clone)
► RabbitPrefab(Clone)
► flower(Clone)
► Palm(Clone)
► Palm(Clone)
► RabbitPrefab(Clone)
► flower(Clone)
► flower(Clone)
► Palm(Clone)
► RabbitPrefab(Clone)
► flower(Clone)
► RabbitPrefab(Clone)
► flower(Clone)
► flower(Clone)
► Palm(Clone)
► flower(Clone)
► flower(Clone)
► RabbitPrefab(Clone)
► RabbitPrefab(Clone)
► Palm(Clone)
► RabbitPrefab(Clone)
► flower(Clone)

```

FIGURA 70. GAMEOBJECT GENERADOS DE MANERA AUTOMÁTICA

La arquitectura que mantienen estos elementos es la siguiente:

- **Player.**

```

▼ Player(Clone)
  ThirdPersonCamera 1
  pjRigSkin2facesPrefab
    ► Character1_Reference
    Object001
    Object002
  ▼ PlayerInfoCanvas
    ► HealthBar
    ► ColorLevelBar
    ► PlayerName
    PlayerImage
    PointLight
    ► Rain

```

FIGURA 71. JERARQUÍA DE OBJETOS DENTRO DEL OBJETO PLAYER

- *ThirdPersonCamera*: Se trata de la cámara en tercera persona del jugador.
- *pjRigSkins2facesPrefab*: Contiene el esqueleto y la maya que componen al jugador.
- *PlayerInfoCanvas*: Muestra información en el HUD sobre el jugador, como las barras de vida y nivel de color, y su nombre de Steam.

- *PointLight*: Luz que emiten los jugadores al consumir un determinado ítem.
- *Rain*: Lluvia que envuelve a los jugadores cuando llueve en el juego.
- **NPC.** Como *NPCs* podemos distinguir dos tipos, de fauna y de vegetación.
 - *NPC de vegetación*: En su arquitectura de objetos encontramos todos aquellos elementos pertenecientes a la maya del modelo. Cabe destacar que, mediante un *script*, durante la generación de cada partida se modifica el material de todos estos objetos pertenecientes a la maya, para *randomizar* los colores de éstos.

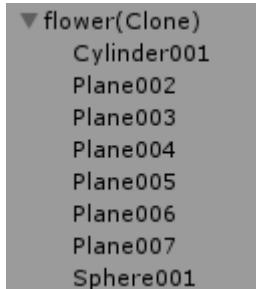


FIGURA 72. EJEMPLO DE JERARQUÍA DE OBJETOS EN UN NPC DE VEGETACIÓN.

- *NPC de fauna*: En su arquitectura de objetos encontramos elementos pertenecientes a la maya del modelo. Al igual que la vegetación, se modifica el material de estos elementos durante la generación de la partida. Destacar también que los *GameObject* pertenecientes a la maya son hijos de un *GameObject* padre, el cuál posee una componente *Animator* para generar las animaciones del NPC en cuestión.

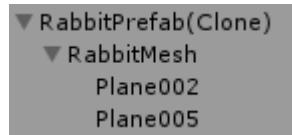


FIGURA 73. EJEMPLO DE JERARQUÍA DE OBJETOS EN UN NPC DE FAUNA.

También, cabe mencionar la arquitectura del *GameObject Canvas* mencionado anteriormente, el cuál es el encargado de la interfaz gráfica del videojuego durante la partida.



FIGURA 74. ARQUITECTURA GAMEOBJECT CANVAS

En el **Canvas** podemos distinguir una serie de objetos:

- **Inventory:** Se encarga de mostrar todos los elementos del inventario en pantalla, los cuales van gestionados mediante un *script* que posee el *GameObject* de **Player**. *Inventory* posee dos objetos principales, *Title* el cuál muestra el texto “Inventory” sobre el inventario, y *ItemsParent*, siendo este el encargado principal de mostrar objetos del inventario, organizado por casillas, representadas cada una de estas por un objeto *InventorySlot*. En este objeto distinguimos 4 objetos diferentes: *ItemButton*, el cuál simplemente contiene el *sprite* del objeto que corresponda con esa casilla, *EquipArmor*, siendo este un marco verde que rodea a la armadura equipada actual, *Equip*, el cuál es un *sprite* con un marco rojo que rodea el ítem equipado en mano, y un texto, *Quantity*, el cuál muestra la cantidad de objetos que se tiene de ese mismo tipo en esos momentos en el inventario.

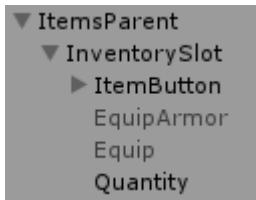


FIGURA 75. CASILLAS DEL INVENTARIO

- **CraftingMenu:** Muestra un menú con todos los objetos que se pueden fabricar durante la partida. Igual que en el inventario, tenemos *Title*, y *ItemsParent*. Este último está también organizado mediante casillas, que reciben el nombre de *CraftSlot*. Estas casillas cuentan con un objeto *ItemButton*, el cual contiene dos objetos hijos, *Colors* y *Icon*. El primero otorga un panel de botones de colores (los 6 colores del videojuego) los cuales se iluminarán y se podrán pulsar siempre y cuando el jugador tenga los objetos necesarios en el inventario. Por otro lado, *Icon* es, simplemente, el ícono *sprite* del objeto a fabricar, el cual se mostrará transparente en caso de no tener los objetos requeridos en el inventario para su creación.



FIGURA 76. CASILLAS DEL MENÚ DE FABRICACIÓN

- **TimeText:** Objeto encargado de mostrar en pantalla la hora del juego actual.
- **DaysText:** Muestra el número de días transcurridos en pantalla.
- **GameOver:** Objeto encargado de mostrar en pantalla la ventana de *GameOver* cuando se termina al juego. Este contiene 3 objetos: *Title*, el cual muestra el texto **GAMEOVER**, *Title(1)*, muestra un texto indicando el número de días que han sobrevivido los jugadores, y *BackButton*, el cual genera un botón con la palabra **BACK** escrita, sirve para volver al menú principal.



FIGURA 77. OBJETOS DE GAMEOVER

Volviendo a la **escena offline**, gran parte de los objetos que la componen en gran parte se encargan de mostrar en pantalla elementos que forman la interfaz del menú principal del juego.

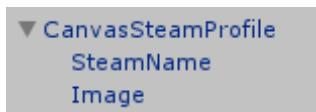


FIGURA 78. JERARQUÍA DE OBJETOS DE CANVASSTEAMPROFILE

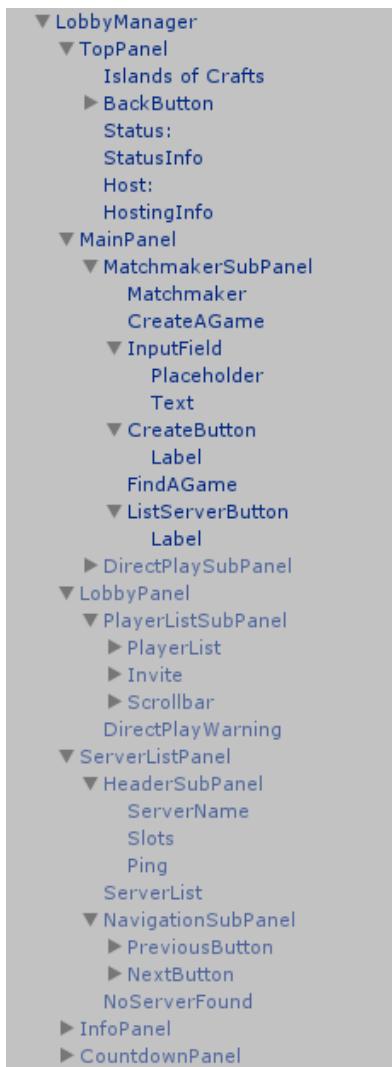


FIGURA 79. JERARQUÍA COMPLETA DE OBJETOS QUE COMPONEN LOBBYMANAGER

- Dentro de **CanvasSteamProfile** encontramos *SteamName* y *Image*, los cuales se encargan de mostrar en el menú principal el nombre y el avatar del jugador perteneciente a su perfil de Steam.
- Dentro de **LobbyManager** encontramos la estructura importante que compone la interfaz de todo el menú principal. Empezamos por **TopPanel** que se encarga de mostrar el nombre del juego, información sobre la red (Si se trata del host, si está offline/online, donde se aloja el servidor (localhost...)), y además contiene un botón de *back* para volver a la pantalla principal en caso de estar en la pantalla de creación de la partida. A continuación, tenemos el **MainPanel**, dentro del cual se encuentra **MatchmakerSubPanel**, siendo este el panel principal donde se muestra los botones y textos para crear partida o buscar una partida ya existente. Una vez se pulsa sobre el botón de crear partida, se habilita el panel de **LobbyPanel**, el cuál muestra un listado de los jugadores en la sala (con **PlayerList**) y, además, aparece el botón **Invite** para invitar amigos de Steam a la partida. En caso de querer unirse a otra partida, al pulsar sobre el botón de **ListServerButton** en el **MainPanel**, se habilitaría el panel de **ServerListPanel**, mostrando un listado de las partidas actuales, con su nombre (**ServerName**) y los huecos disponibles (**Slots**); además, en la parte inferior de la pantalla se habilita un panel (**NavigationSubPanel**) con botones para navegar en esta ventana, en caso de que haya muchas partidas, ordenándolas por páginas. Por otra parte, el panel de **InfoPanel** se encarga de hacer aparecer una pequeña ventana de espera indicando cuándo te estás conectando o uniendo a la sala de una partida. Finalmente, **CountdownPanel** es un panel que se activa en el momento en el que una partida va a dar comienzo, apareciendo una pequeña cuenta atrás (con unos segundos, que se pueden configurar desde el script de **LobbyManager**) indicando a los jugadores el tiempo restante para dar comienzo a la partida.

Al pasar de la **escena online** a la **escena offline** los objetos no destruidos de esta última, aparecerán en la jerarquía de objetos de la **escena online** como **DontDestroyOnLoad**, siendo estos los mencionados anteriormente: **LobbyManager** y **SteamManager**.

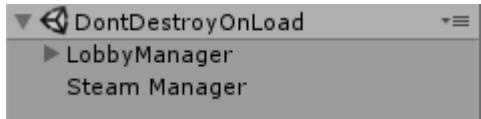


FIGURA 80. OBJETOS QUE PERDURAN EN EL CAMBIO DE ESCENA DE OFFLINE A ONLINE

Resumiendo todo lo dicho anteriormente, los diferentes objetos que componen las dos escenas, online y offline, son los que se muestran en la tabla a continuación.

Escena offline	Escena online
CanvasSteamProfile	GameManager
EventManager	AudioManager
	EventManager
	MapManager
	Canvas
	Sea
	DirectionalLight
SteamManager	
LobbyManager	

FIGURA 81. TABLA RESUMEN DE LOS OBJETOS QUE APARECEN EN LA ESCENA OFFLINE Y ONLINE

Y dicho esto y teniendo en cuenta que los dos puntos fuertes de este videojuego son el modo multijugador y la generación procedural del mundo, decidí que lo primero que debía de implementar eran ambas características antes que cualquier otra.

5.3.1 Multijugador

5.3.1.1 Unity Network

Como mencioné en apartados anteriores, decidí implementar el multijugador con el sistema de red propio de Unity, UNET.

Nada más crear un proyecto, lo primero es activar el modo multijugador de Unity. Antes que nada, es importante estar registrado y tener una cuenta de Unity.

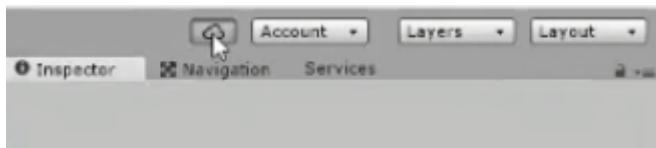


FIGURA 82. ACCESO A LOS SERVICIOS ONLINE DE UNITY

Para ello es necesario vincular el proyecto que se está creando a la cuenta de Unity.

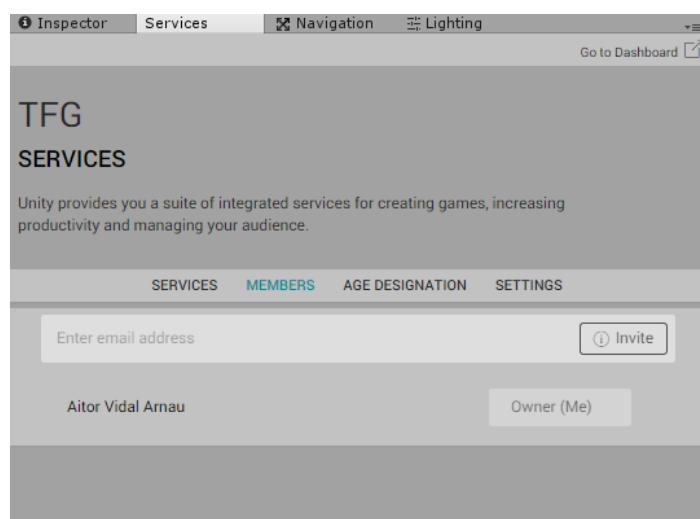


FIGURA 83. VINCULACIÓN DE LA CUENTA DE UNITY CON EL PROYECTO

De esta manera nuestro proyecto se creará en la web de Unity, y nos aparecerán una serie de servicios en nuestro panel de usuario.

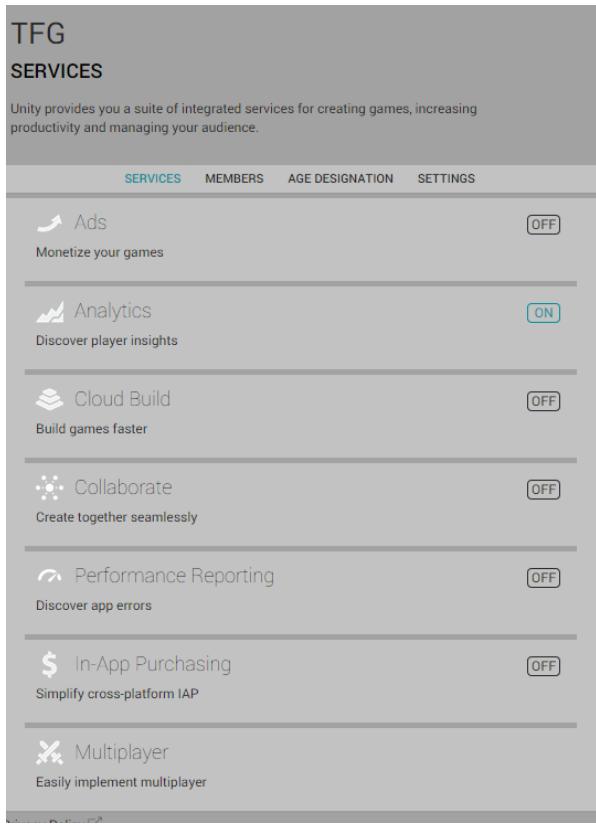


FIGURA 84. SERVICIOS DE UNITY NETWORK

Al hacer clic sobre *Multiplayer* aparecerá lo siguiente:

The screenshot shows the 'MULTIPLAYER' configuration overview. It includes:

- CONFIGURATION OVERVIEW**: Streamlined software and hardware to implement multiplayer features.
- A link to configure Multiplayer settings: "Please go to this link to configure the Multiplayer settings."
- A "Go to dashboard" button.
- Current configuration** table:

SUBSCRIPTION PLAN	personal
GLOBAL CCU AVAILABLE	20
TOTAL CCU USED	0
CCU USED BY THIS PROJECT	0
MAX PLAYERS	4

- A "Refresh Configuration" button.
- Supported Platforms** section: iOS, Android, WebPlayer, PC, Mac, Linux, Xbox One, PS4.

FIGURA 85. OPCIONES DEL APARTADO *MULTIPLAYER* EN EL PANEL DE SERVICIOS DE UNET

Y al hacer clic sobre *Go to dashboard* se abrirá una ventana de navegador con un panel de configuración.

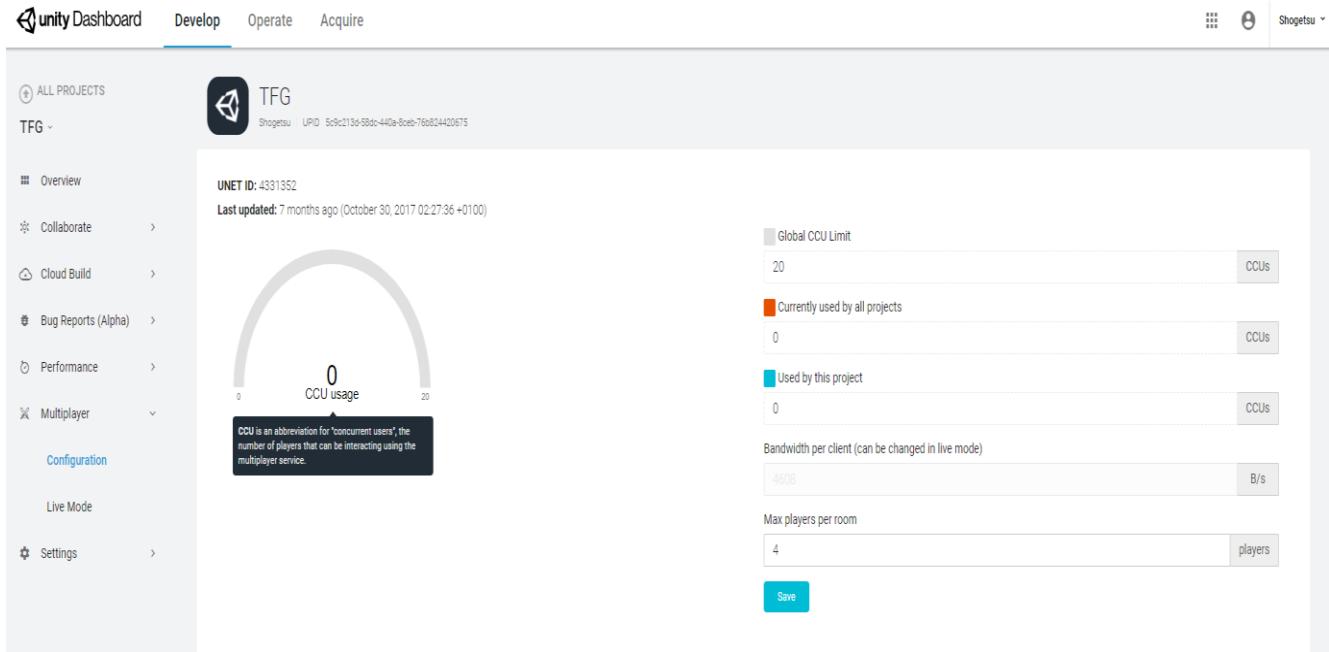


FIGURA 86. UNITY DASHBOARD

En este panel de configuración se ofrecen una serie de opciones como el número máximo de jugadores concurrentes que estén jugando (*Global CCU Limit*) y el número máximo de jugadores por sala (o partida), en mi caso configuro este último para que tenga un límite de 4 jugadores.



FIGURA 87. MAX PLAYERS PER ROOM

Si accedemos al apartado de *Live Mode* podemos acceder a una serie de servicios muy personalizables, pero todos de pago.

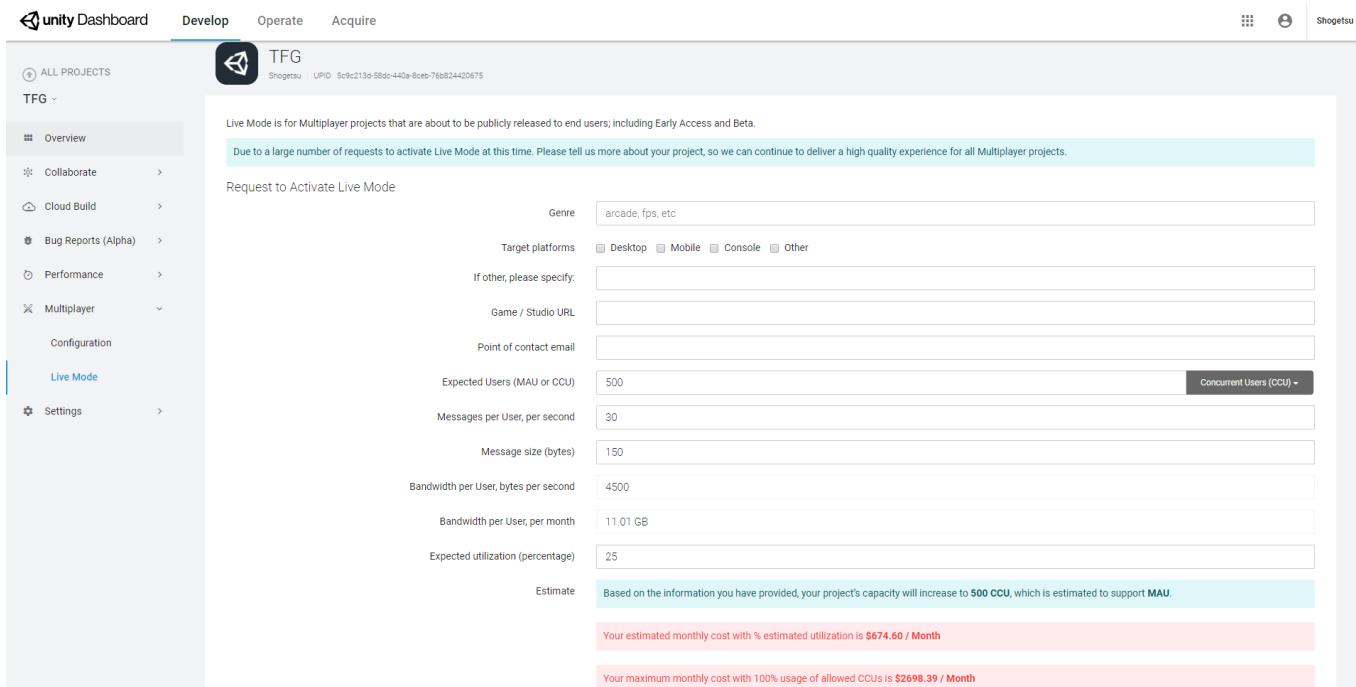


FIGURA 88. UNITY NETWORK LIVE MODE

Llegados a este punto, el proyecto ya tendría configurado el modo multijugador online, pero para hacer que todo esto tenga algún sentido, será necesario crear un *GameObject* que actuará como gestor del sistema de *network*, el cuál yo he llamado *LobbyManager*.

5.3.1.2 *Lobby Manager*

Unity permite configurar de manera sencilla el *GameObject* mencionado anteriormente mediante un script que recibe el nombre de *NetworkManager*, sin embargo, decidí descargar un *asset* oficial gratuito de Unity (*Network Lobby*) el cual nos aporta las mismas opciones que el script mencionado, pero añade un *Lobby* mucho más vistoso y configurable.

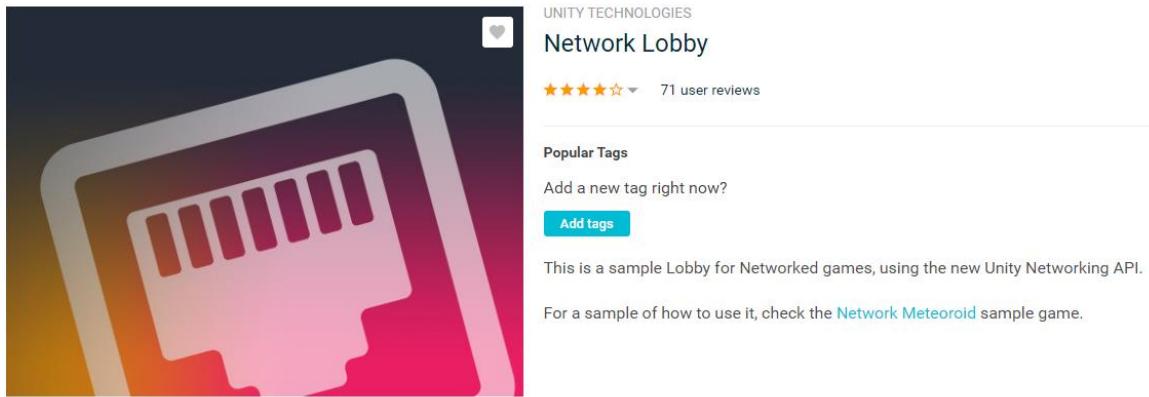


FIGURA 89. ASSET NETWORK LOBBY

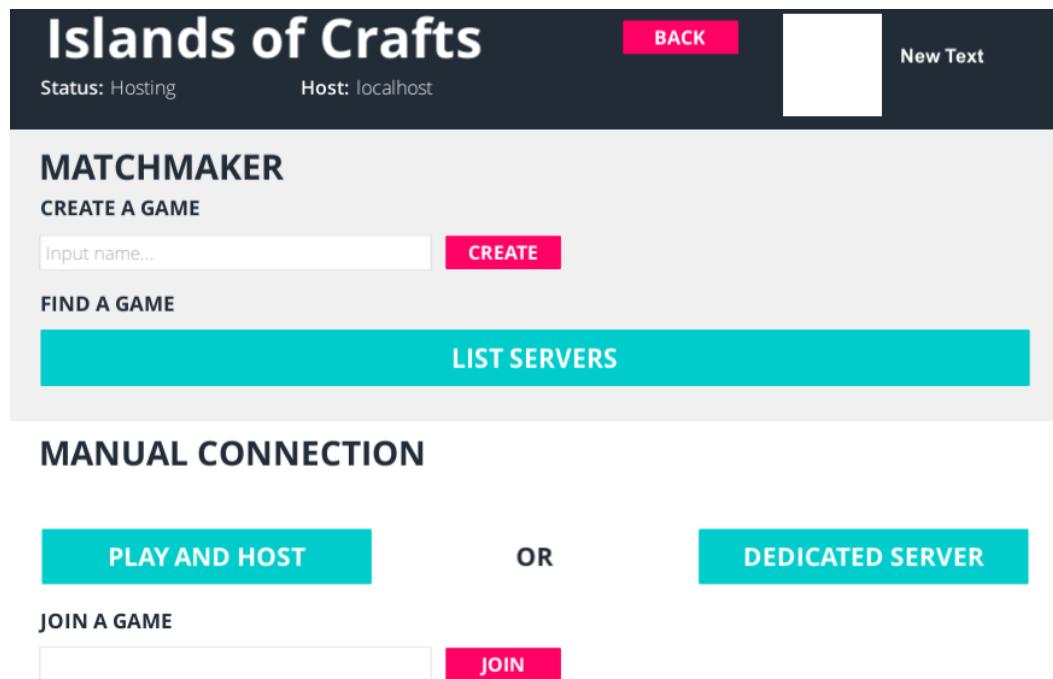


FIGURA 90. MENÚ PRINCIPAL DE NETWORK LOBBY

Este asset contiene el script *LobbyManager*.

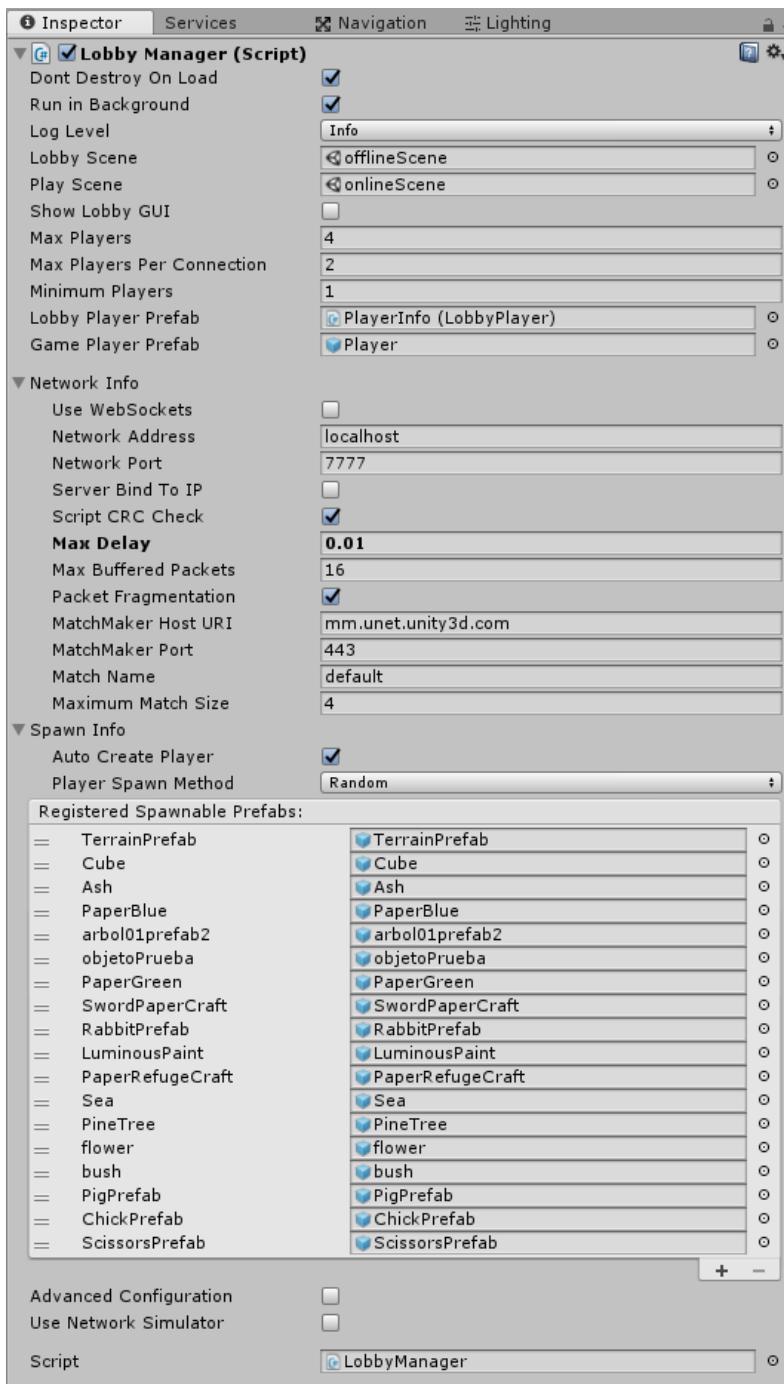


FIGURA 91. SCRIPT LOBBYMANAGER

Sobre este script, permite una amplia configuración de la red del videojuego, pero yo voy a destacar las siguientes características:

- **Run in background:** Es obligatorio seleccionar esta opción, de lo contrario el juego se pausará cuando la ventana esté minimizada. En el caso de que sea el jugador *host* quien minimice la pantalla, al resto de jugadores se les quedaría la pantalla congelada.

- **Lobby Scene:** Es la escena que pertenece al lobby.
- **Play Scene:** Escena que pertenece a la partida.
- **Lobby Player Prefab:** Se generará una réplica de este *prefab* en cada uno de los clientes. Éste se encarga de gestionar la información de cada jugador (color, nombre, *ready*, etc) de manera individual **mientras se encuentren en el Lobby**.

De esto se encarga el script *LobbyPlayer*.

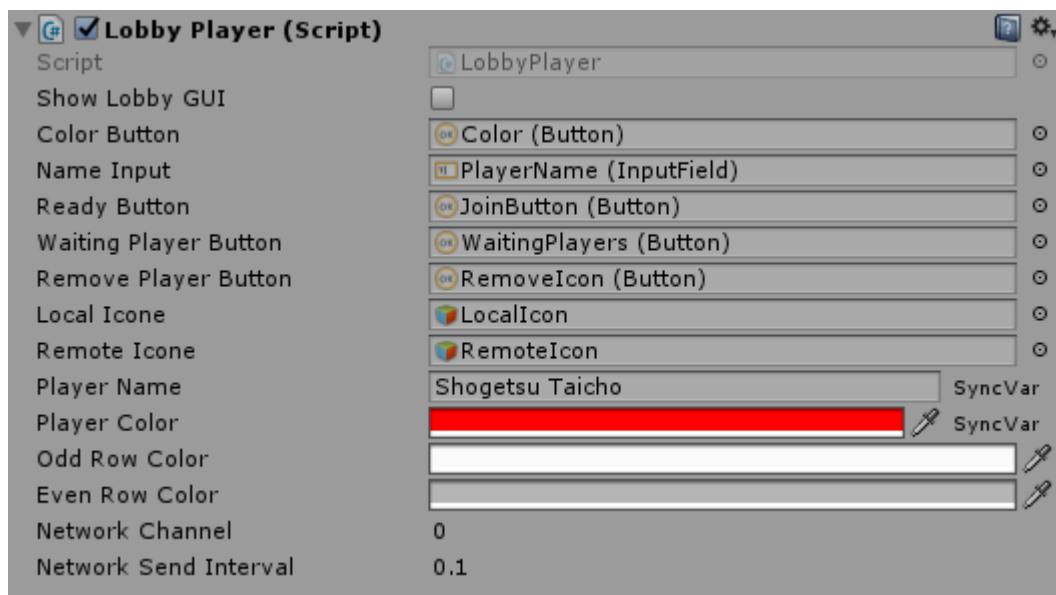


FIGURA 92. SCRIPT LOBBYPLAYER

- **Game Player Prefab:** Se generará una réplica de este *prefab* en cada uno de los clientes **durante la partida**. Es el *prefab* de cada uno de los jugadores.
- **Spawn Info:** Aquí se encuentra un listado con todos los *prefab* que el servidor podrá invocar durante la partida. Este listado ayuda a que los jugadores no hagan trampas generando objetos por su cuenta.

Volviendo al menú principal, el jugador podrá escribir el nombre de su partida y crearla mediante el botón *CREATE*, o unirse a una partida ya existente (y en proceso de creación) pulsando sobre *LIST SERVERS*.

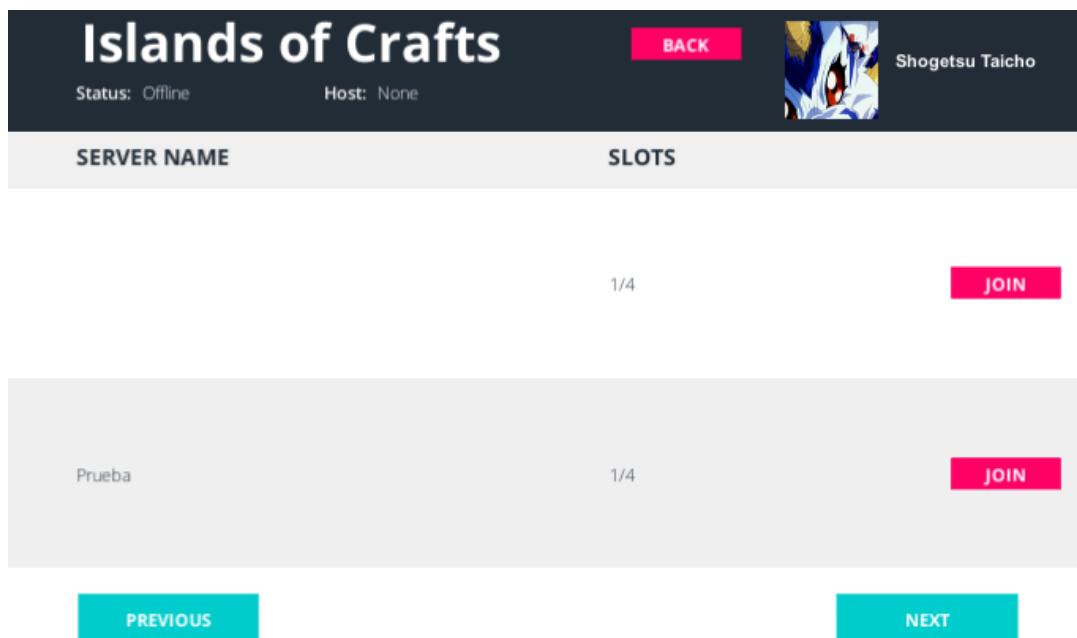


FIGURA 93. LISTADO DE PARTIDAS EN EL LOBBY

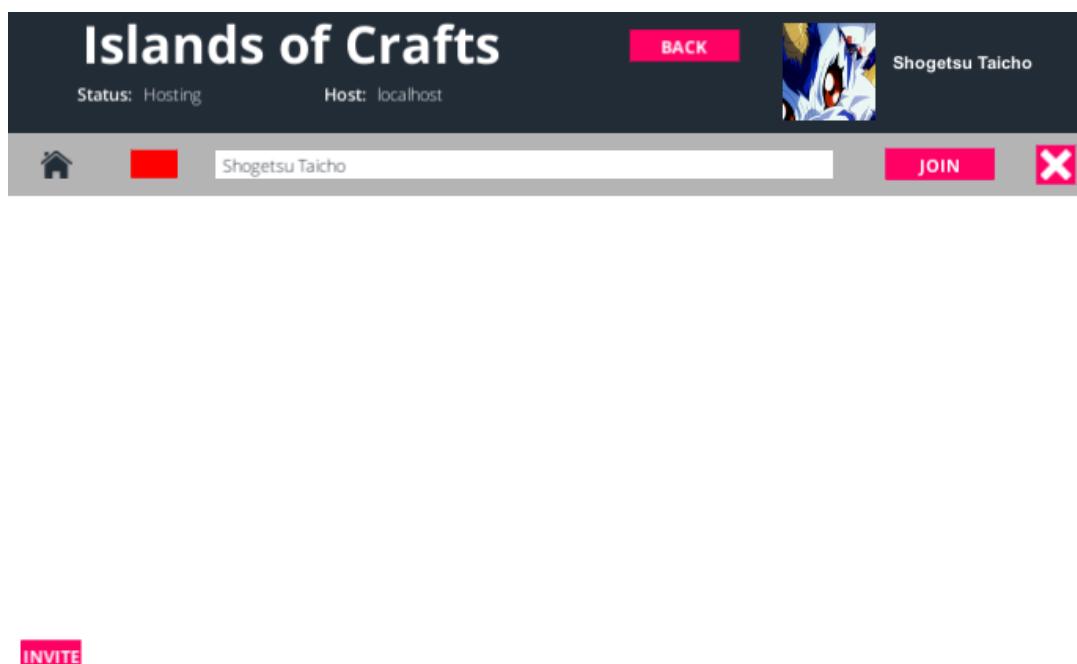


FIGURA 94. MENÚ DE CREACIÓN DE PARTIDA

En el menú de creación de partida, será posible invitar a los amigos que el jugador tenga agregados en su lista de amigos de *Steam* (esto pasará a explicarlo más adelante). Además, es posible cambiar el color del jugador presionando sobre el rectángulo de color que aparece a la izquierda de su *Nick*.

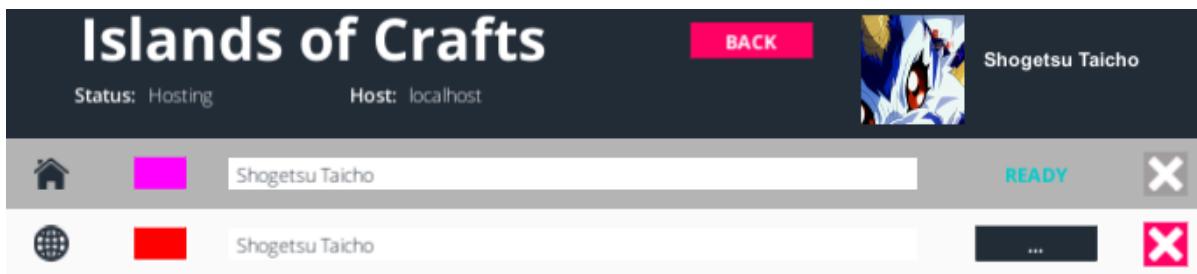


FIGURA 95. EJEMPLO DE CAMBIO DE COLORES Y JUGADOR PREPARADO EN EL LOBBY

Una vez todos los jugadores se encuentran listos (al presionar el botón *JOIN*), aparece una cuenta atrás y, al terminar esta cuenta atrás, se hace paso a la escena de la partida (escena online) mencionada anteriormente.

Llegados a este punto, es muy importante señalar una serie de premisas que se deberán de tener en cuenta a la hora del desarrollo de nuestro videojuego:

- Es importante que cada uno de los *GameObject* que se generen en el servidor tengan la componente *Network Identity*. Esto permite que todos los objetos implicados en el videojuego queden sincronizados entre todos los jugadores, ya que se está otorgando un identificador a cada uno de los distintos objetos para que un cliente sepa que ese objeto es único. Si ese objeto es el jugador, importante seleccionar *Local Player Authority* puesto que ese será el objeto que moverá el jugador; esto simplemente permite al sistema saber cuál es tu personaje, y cuál no es.



FIGURA 96. COMPONENTE NETWORK IDENTITY

- Existen dos tipos de funciones clave para la comunicación entre el servidor y los clientes. Estas funciones se conocen como *Command* y *ClientRpc*. Los clientes son los que se encargan de ordenar al servidor para que éste ejecute una función *Command*, de tal manera que el contenido de este tipo de funciones sólo se ejecutará en el servidor, pudiendo usar los datos del cliente. Estas funciones

se utilizan sobre todo para sincronizar variables (SyncVar). Por otro lado, las funciones *ClientRpc* son ejecutadas por el servidor, de tal forma que el servidor obliga a que todos los clientes ejecuten el contenido de esta función, pudiendo usar los datos del servidor. Este tipo de funciones se utilizan sobre todo para actualizar variables en los clientes, o elementos gráficos como el HUD.

```
[SyncVar]  
int vit;
```

FIGURA 97. EJEMPLO DE SYNCVAR

```
[Command]  
public void CmdHeal(int heal)  
{  
    vit = vit + heal;  
    RpcUpdateHealthBar();  
}
```

FIGURA 98. EJEMPLO DE FUNCIÓN COMMAND

```
[ClientRpc]  
void RpcUpdateHealthBar()  
{  
    healthBar.sizeDelta = new Vector2(vit, healthBar.sizeDelta.y);  
}
```

FIGURA 99. EJEMPLO DE FUNCIÓN RPCCLIENT

Importante destacar que estas funciones pueden recibir parámetros como cualquier otra función normal, sin embargo, el tipo de estos parámetros está limitado al listado siguiente:

- basic types (byte, int, float, string, UInt64, etc)
- arrays of basic types
- structs containing allowable types
- built-in unity math types (Vector3, Quaternion, etc)
- NetworkIdentity
- NetworkInstanceId
- NetworkHash128
- GameObject with a NetworkIdentity component attached

- Para sincronizar correctamente las transformaciones de los objetos entre todos los clientes, es necesario añadir en estos la componente *Network Transform*. En caso de no añadir esta componente, las transformaciones que sufra un objeto afectarían a todos los objetos del mismo tipo.

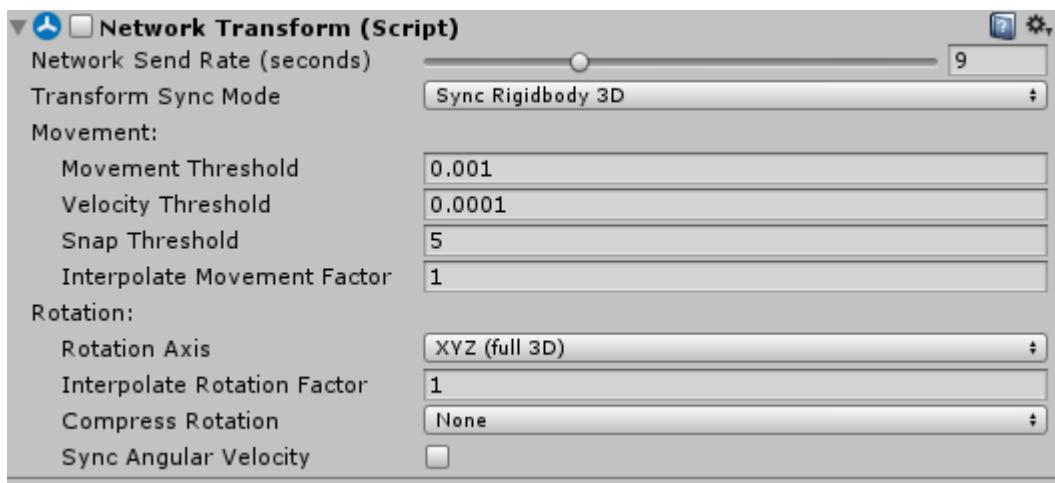


FIGURA 100. COMPONENTE NETWORK TRANSFORM

Sincronizar posición, rotación, y escalado son 9 *floats*.

Si un jugador está completamente quieto, esos 9 *floats* no varían, y no tiene mucho sentido que estén constantemente enviándose.

Para evitar esto, el sistema que usa Unity para diferenciar datos, se llama Sistema Diferencial, es decir, de esta manera que no se mandan los valores de los *floats* mencionados, sino que se envían la diferencia de los *floats* con respecto al *frame* anterior, de tal forma que cuando el personaje está quieto y ninguna de las 3 (posición, rotación, y escalado) varía, el tráfico es igual a 0. Unity, al emplear esta técnica, reduce bastante el ancho de banda. Cabe destacar que se emplea esta técnica de forma automática, pero en versiones más antiguas esto no era así. Ahora, gracias a esto, es posible incluso ahorrar hasta un 80% del coste del servidor por tráfico. Otro consejo muy útil para ahorrar en los costes por tráfico es mantener al usuario el máximo tiempo posible sin conexión de datos porque todo el tiempo que esté el jugador inactivo sin consumir ancho de banda, eso que se ahorra el servidor. Esto se aplicaría sobre todo a partes como en el Lobby o menú de inicio de un juego online, donde el jugador no esté

haciendo nada. Antes, en versiones antiguas, todo esto había que controlarlo, pero actualmente el propio componente (*Network Manager* o, en mi caso, *LobbyManager*) de Unity es capaz de detectar cuándo hay una escena online y cuándo hay una escena offline.

En cuanto a los parámetros del *NetworkTransform*, es importante tener en cuenta la tasa de envíos de mensajes por segundo (el primer parámetro), ya que un número muy alto puede ocasionar problemas graves en el servidor.

Para el jugador siempre es conveniente tenerlo al máximo (29), pero para otros objetos con dejarlo a 9 sería suficiente. Hay casos especiales, como, por ejemplo, en el caso de un proyectil, en los que se podría poner la tasa de envíos de mensajes por segundo a 0, ya que, en este caso, con saber la posición inicial del proyectil, en cada cliente se ejecutará el script correspondiente que moverá el proyectil por el escenario.

- Para sincronizar las animaciones de los objetos entre los clientes bastará con añadir la componente *Network Animator*.

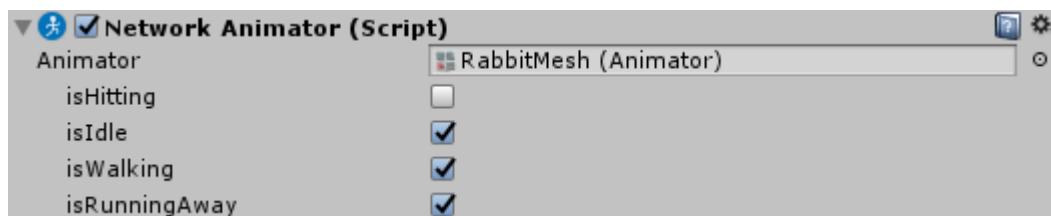


FIGURA 101. COMPONENTE NETWORK ANIMATOR

Una vez se le añade la componente *Animator* que corresponda con el *GameObject* en cuestión, aparecerán los distintos estados de animación con los que cuenta ese objeto. Solo aquellos estados seleccionados serán los que se sincronicen. No sincronizar algunos estados pueden ahorrar ancho de banda.

- Para que a la hora de compilar el videojuego funcione correctamente, es necesario establecer las escenas que se van a ejecutar y el orden en que se van a ejecutar (primero la escena offline, y luego la escena online). Para ello hay que acceder a *File>Build Settings* y marcar las dos escenas de esta forma y **en el mismo orden**.

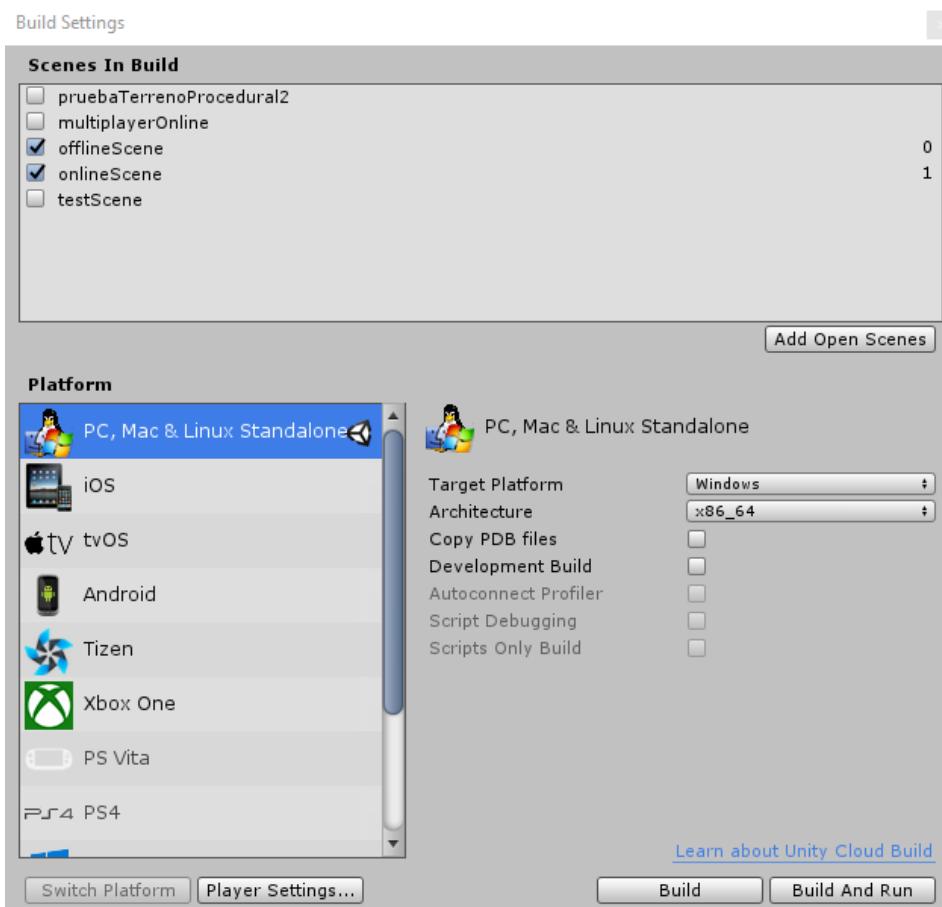


FIGURA 102. BUILD SETTINGS EN UNITY

5.3.1.3 Steamworks

Como mencioné anteriormente, este videojuego hará uso de la API de Steam para algunas funciones sociales.

Para poder utilizar esta API, **Steamworks**, es necesario descargar un paquete de scripts de la propia página oficial de *Steamworks*.

Una vez hecho esto, se crea un *GameObject* que recibirá el nombre de *SteamManager*. Este objeto contendrá los scripts necesarios para hacer que la API de Steam funcione correctamente en el proyecto.

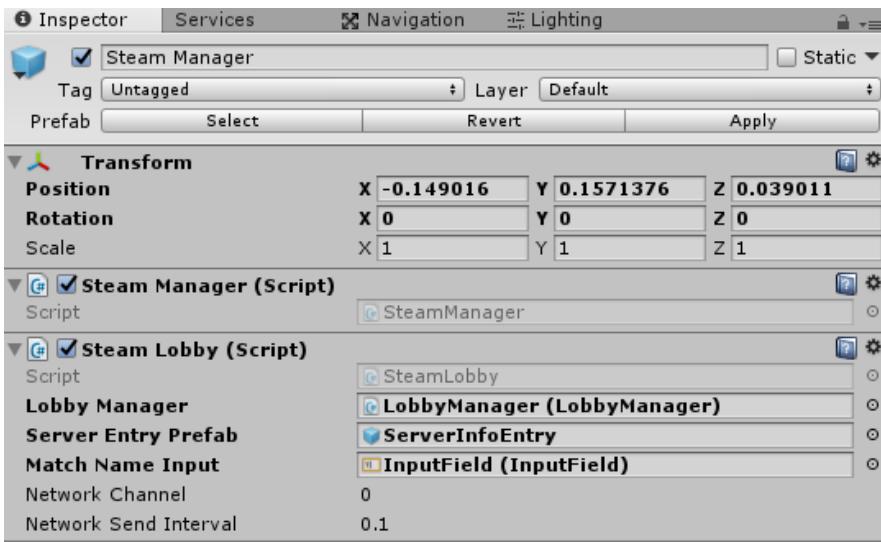


FIGURA 103. SCRIPTS DEL GAMEOBJECT STEAMMANAGER

SteamManager es un script se encarga de inicializar Steam al iniciar el juego. Permite poder utilizar el *overlay* de Steam en nuestro videojuego.

Antes de continuar, hay que tener en cuenta el ID de nuestro videojuego en la base de datos de Steam; evidentemente, Steam no tiene datos sobre nuestro videojuego, por lo que tendremos que utilizar una ID de ejemplo que nos brinda la propia API para que los desarrolladores podamos hacer pruebas antes de publicar ningún juego en Steam. Para ello, hay que acceder a la carpeta raíz del proyecto donde, tras la instalación de la API, habrá aparecido un txt que recibe el nombre de *steam_appid.txt*. Dentro de este txt nos tendremos que asegurar de que se encuentre escrito el **ID 480**, que es el ID de ejemplo para realizar pruebas. Importante tener en cuenta que, si se compila el proyecto y se genera un ejecutable, hay que hacer una copia de este *txt* y ponerlo junto al ejecutable generado, de lo contrario las funciones de Steam de nuestro videojuego en el ejecutable no funcionarán.

Una vez hecho esto, decidí hacer que se mostrase el nombre de Steam del jugador en el menú principal del juego, además de su avatar. Para ello creé el siguiente script:



FIGURA 104. SCRIPT DISPLAYNAME

DisplayName es el *nick* del jugador en cuestión, y **AvatarImage** el avatar del jugador. Los datos son obtenidos de manera sencilla mediante las funciones que nos brinda Steamworks y, a partir de ahí, estos datos se añaden en las componentes respectivas del canvas (Una componente *Text* para el nombre, y una componente *Image* (sprite) para el avatar).

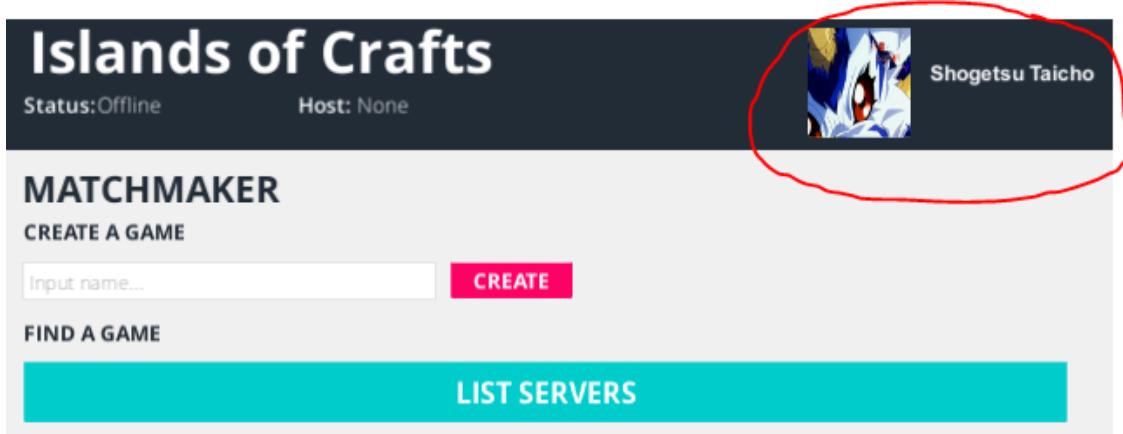


FIGURA 105. EJEMPLO DE NICK DE STEAM EN EL VIDEOJUEGO

Dentro de **SteamManager** se encuentra otro script, que recibe el nombre de **SteamLobby**. Este script lo he utilizado para implementar el sistema de **invitación de amigos** a las partidas, solo durante la creación de éstas.

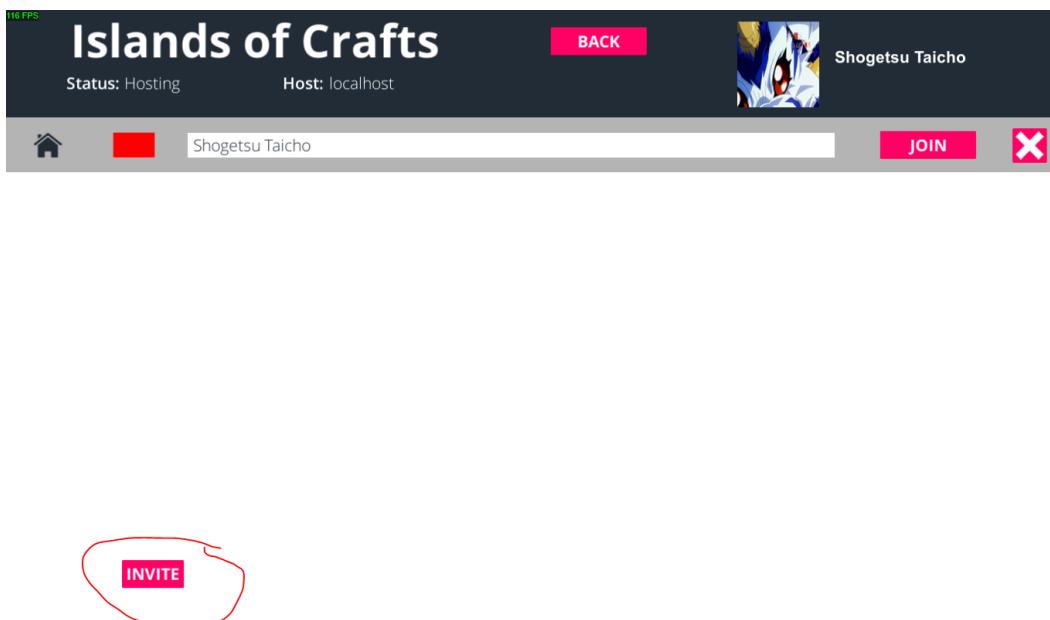


FIGURA 106. BOTÓN PARA INVITAR AMIGOS DE STEAM

Al presionar sobre el botón *Invite* durante la creación de la partida, se abrirá el *overlay* de Steam mostrando tu lista de amigos.

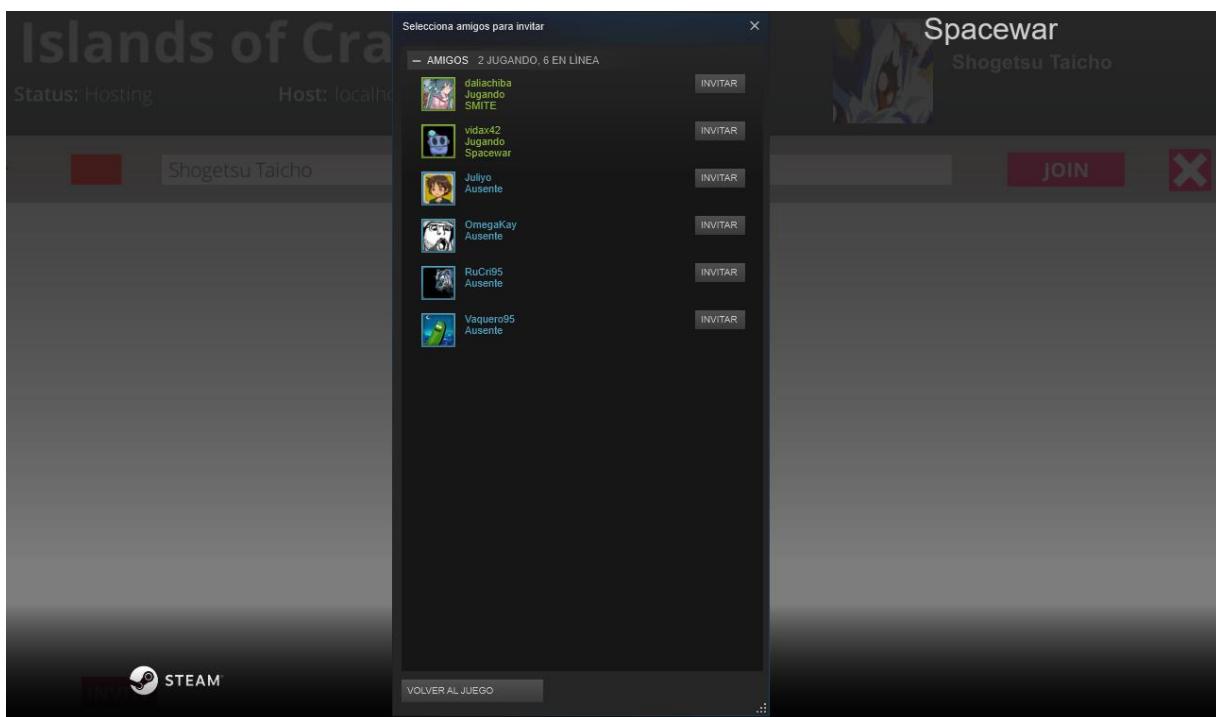


FIGURA 107. OVERLAY DE STEAM PARA INVITAR AMIGOS

En este listado únicamente aparecerán los amigos que se encuentran conectados en estos momentos. Para abrir esta ventana se ha utilizado la función *ActivateGameOverlayInviteDialog((CSteamID)current_lobbyID)* de Steamworks.

Al presionar sobre el botón de **invitar** se enviará, a través de la función mencionada, el ID de la partida actual al amigo en cuestión, de tal manera que, en el momento en el que el usuario invitado reciba la invitación y la acepte, éste recogerá el ID de la partida y se unirá a esa partida en concreto mediante una función encargada para ello.



FIGURA 108. EJEMPLO DE INVITACIÓN ENVIADA

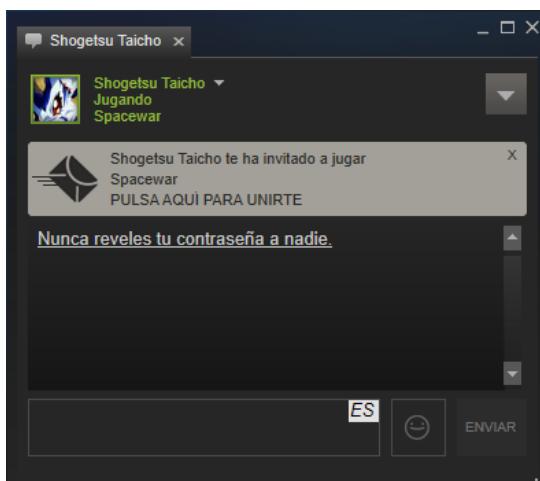


FIGURA 109. EJEMPLO DE INVITACIÓN ENVIADA LISTA PARA ACEPTAR



FIGURA 110. EJEMPLO DE USUARIO INVITADO RECENTEMENTE A LA PARTIDA

Además, cabe destacar que el nombre de usuario del jugador invitado se mostrará sincronizándose la cadena de texto en todos los jugadores.

5.3.2 Generación de un mundo procedural

5.3.2.1 Generación del terreno mediante el algoritmo de *Perlin Noise*

Para la generación procedural del terreno del mundo, decidí utilizar un algoritmo que recibe el nombre de ***Perlin Noise*** o Ruido Perlin en español.

Este algoritmo produce una secuencia naturalmente ordenada (suave) de números pseudoaleatorios, a diferencia de los números aleatorios puros. De esta forma, el algoritmo nos permite crear de una imagen de ruido como la que se muestra a continuación.

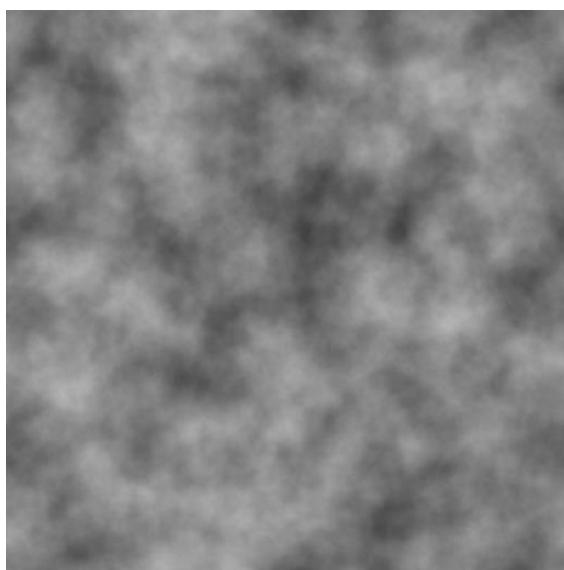


FIGURA 111. TEXTURA DE RUIDO GENERADA CON PERLIN NOISE

La idea fundamental detrás de esto es la de utilizar esta imagen como mapa de alturas del terreno, es decir, las partes más claras de la textura serán las partes más altas del terreno, y las partes más oscuras serán las partes más bajas, permitiendo así la generación procedural de montañas y playas.

En este videojuego las partidas transcurren en una isla, por lo tanto, la idea era que el algoritmo simulase un terreno similar a una isla, generando una imagen como la que se muestra a continuación.

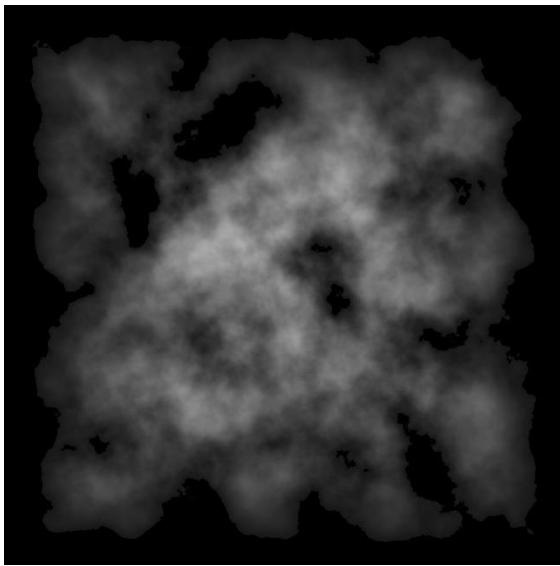


FIGURA 112. TEXTURA DE RUIDO GENERADA CON PERLIN NOISE, CON FORMA CUADRADA

Para lograr esto, se empleó el siguiente código haciendo uso de **PerlinNoise**.

```
float CalculateHeights(int x, int y)
{
    float perlinValue;

    float distance_x = Mathf.Abs(x - width * 0.5f);
    float distance_y = Mathf.Abs(y - height * 0.5f);
    float distance = Mathf.Max(distance_x, distance_y); // mascara cuadrada

    float max_width = width * 0.5f - 10.0f;
    float delta = distance / max_width;
    float gradient = delta * delta;

    float xCoord = (float)x / width * scale + offsetX;
    float yCoord = (float)y / height * scale + offsetY;
    perlinValue = (Mathf.PerlinNoise(xCoord * 4, yCoord * 4) +
                  0.5f * Mathf.PerlinNoise(xCoord * 8, yCoord * 8) +
                  0.25f * Mathf.PerlinNoise(xCoord * 16, yCoord * 16)) *
                  Mathf.Max(0.0f, 1.0f - gradient);

    return perlinValue;
}
```

FIGURA 113. CÓDIGO DEL ALGORITMO PERLIN NOISE

Esta función se llama directamente desde esta otra función.

```

float[,] GenerateHeights()
{
    /*Tenemos una cuadricula de puntos,
    en los cuales cada punto tiene asociado un float que determina su altura,
    usando Perlin Noise */

    heights = new float[width, height];

    //Hacemos un bucle para cada uno de estos puntos
    for (int x=0; x < width; x++)
    {
        for (int y=0; y<height; y++)
        {
            heights[x, y] = Mathf.Pow(Mathf.Round(CalculateHeights(x, y) * 128) / 128, 3);
        }
    }
    return heights; //Para las alturas, este metodo devuelve un array de dos floats correspondientes al ruido
}

```

FIGURA 114. FUNCIÓN GENERATEHEIGHTS() ENCARGADA DE ASIGNAR A LOS VALORES QUE DEVUELVE EL ALGORITMO PERLIN NOISE

```

TerrainData GenerateTerrain(TerrainData terrainData)
{
    terrainData.heightmapResolution = width + 1;
    terrainData.size = new Vector3(width, depth, height);
    terrainData.SetHeights(0, 0, GenerateHeights());
    return terrainData;
}

```

FIGURA 115. FUNCIÓN QUE SE ENCARGA DE GENERAR EL TERRENO FINALMENTE

En esta última función se asignan las alturas a cada uno de los puntos del mapa, siendo estas alturas el resultado obtenido del algoritmo Perlin Noise.

Todas estas funciones están implementadas en un script que se encuentra dentro del *prefab* del terreno.

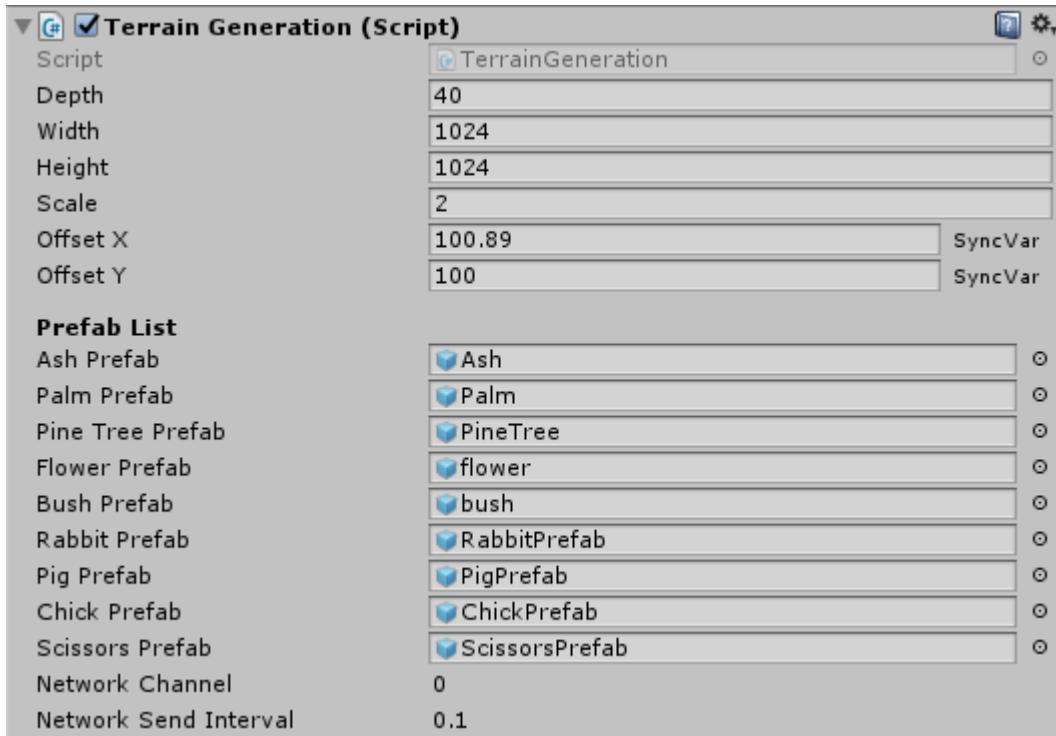


FIGURA 116. SCRIPT TERRAINGENERATION, PARA LA GENERACION DEL TERRENO

Este script permite configurar el terreno que se va a generar, como el tamaño, la profundidad, etc.

Además, cuenta con los valores **OffsetX** y **OffsetY**, los cuales se generan de manera aleatoria y representan a la semilla que hace posible la aleatoriedad del terreno, es **muy importante** que estos dos valores se **sincronicen** (usando **SyncVar**) en **todos los clientes**, ya que, en caso contrario, cada uno de los jugadores vería un terreno completamente distinto.

Por otra parte, hay un listado de *prefabs* siendo estos los que aparecerán de manera aleatoria por toda la isla, como árboles o conejos.

5.3.2.2 Texturizado automático del terreno

En primer lugar, hay que añadir todas las texturas que tendrá el terreno.

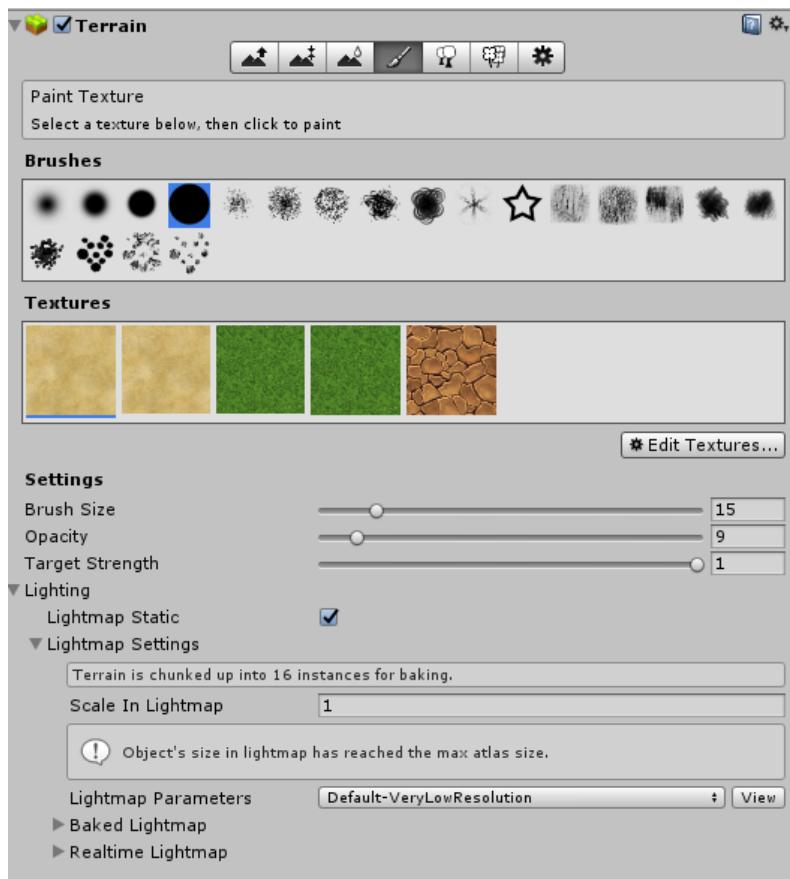


FIGURA 117. MENÚ DE TEXTURAS DEL TERRENO

Cada textura se corresponde a una altura distinta del terreno, en mi caso el terreno tiene 5 alturas distintas, en las cuales las alturas 0 y 1 tendrán la misma textura, y las alturas 2 y 3 también.

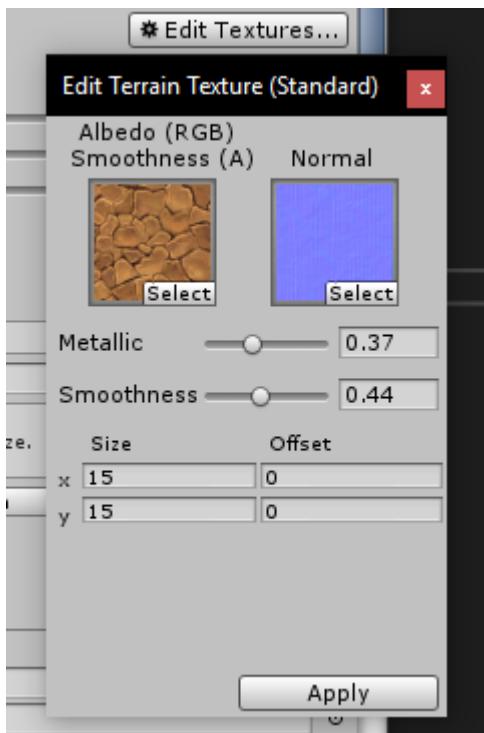


FIGURA 118. EDITAR UNA TEXTURA DEL TERRENO

A cada una de las texturas les añadí un mapa de normales para darles un ligero realismo, además de modificar los valores de Metallic y Smoothness. Los mapas de normales que empleé para las texturas del terreno simulan como si el terreno estuviese hecho de papel o cartón, además de estar dibujado.

El *script* que se muestra a continuación es el encargado de aplicar cada una de las texturas en función de la altura del terreno.

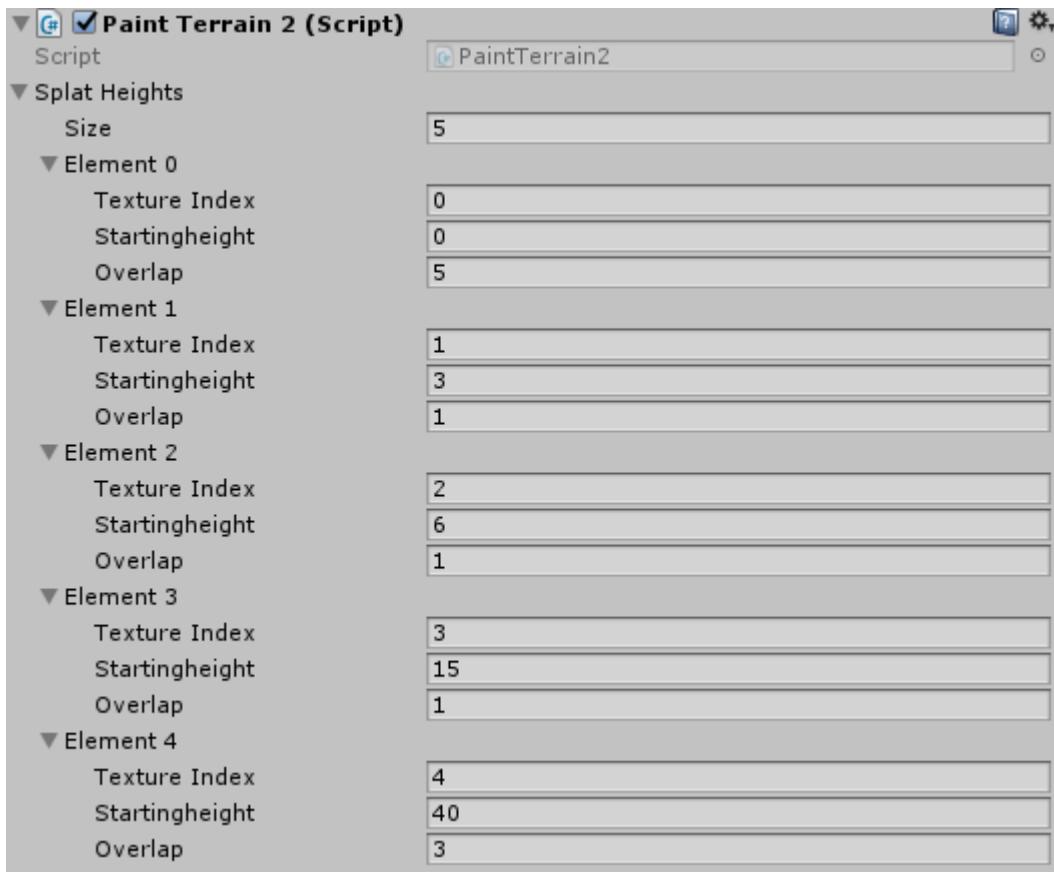


FIGURA 119. SCRIPT PARA PINTAR DE FORMA AUTOMÁTICA EL TERRENO, PAINTTERRAIN2

Cada altura cuenta con 3 variables:

- La primera variable es **TextureIndex**, es simplemente la enumeración dentro del array de SplatHeights.
- La segunda variable es **Startingheight** y es la encargada de asignar la altura inicial donde esa textura comienza a aplicarse en el terreno.
- La tercera variable es **Overlap**, y esta se encarga de aplicar un ligero solapamiento entre las texturas (tanto en la parte más baja como en la parte más alta) para que la transición entre texturas sea de manera suave.

Es importante destacar que en este script no se ha sincronizado ninguna variable entre los clientes, basta con hacer todas las operaciones de forma local, ya que al sincronizar la semilla del terreno generado los cálculos serán los mismos en todos los clientes.

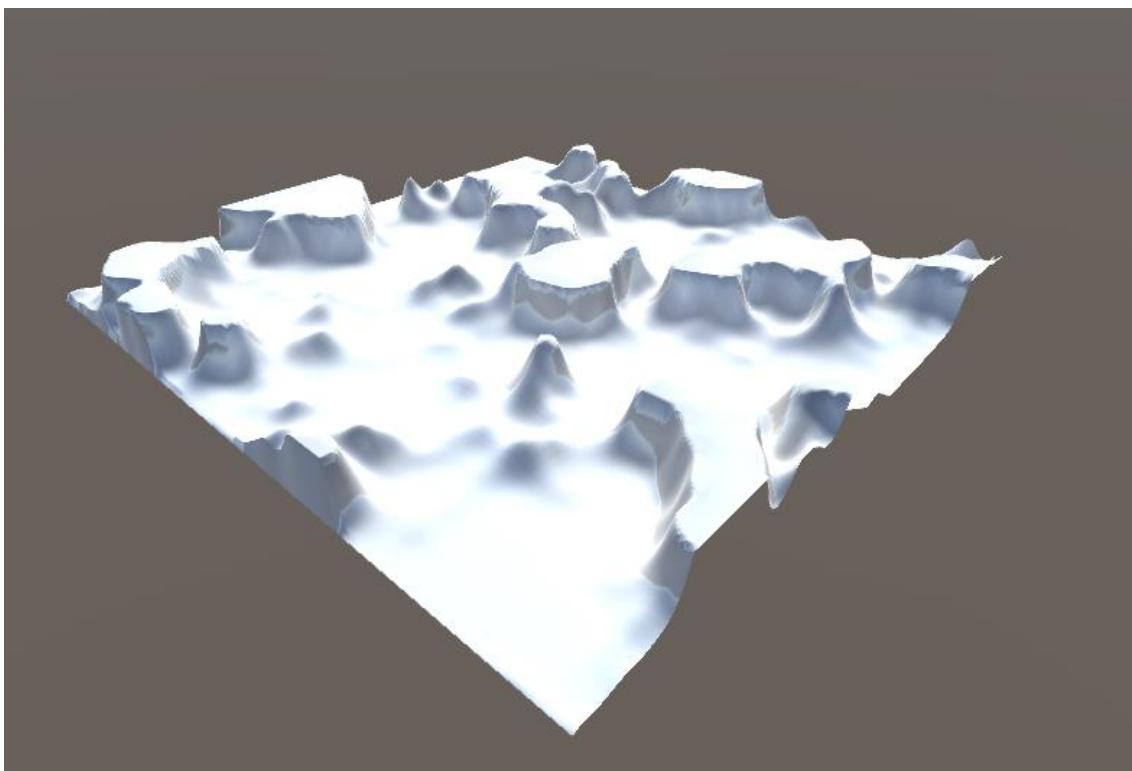


FIGURA 120. INICIOS DE TERRENO PROCEDURAL

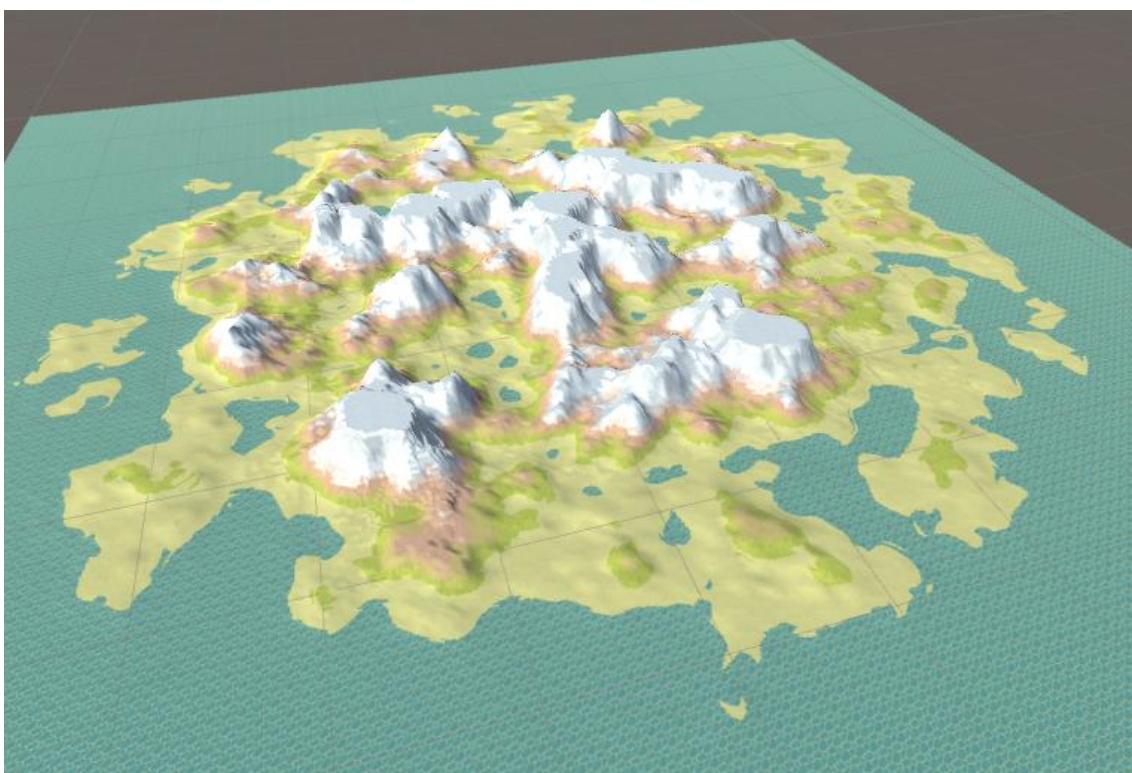


FIGURA 121. TERRENO PROCEDURAL EN FORMA DE ISLA, Y TEXTURIZADO COMPLETAMENTE

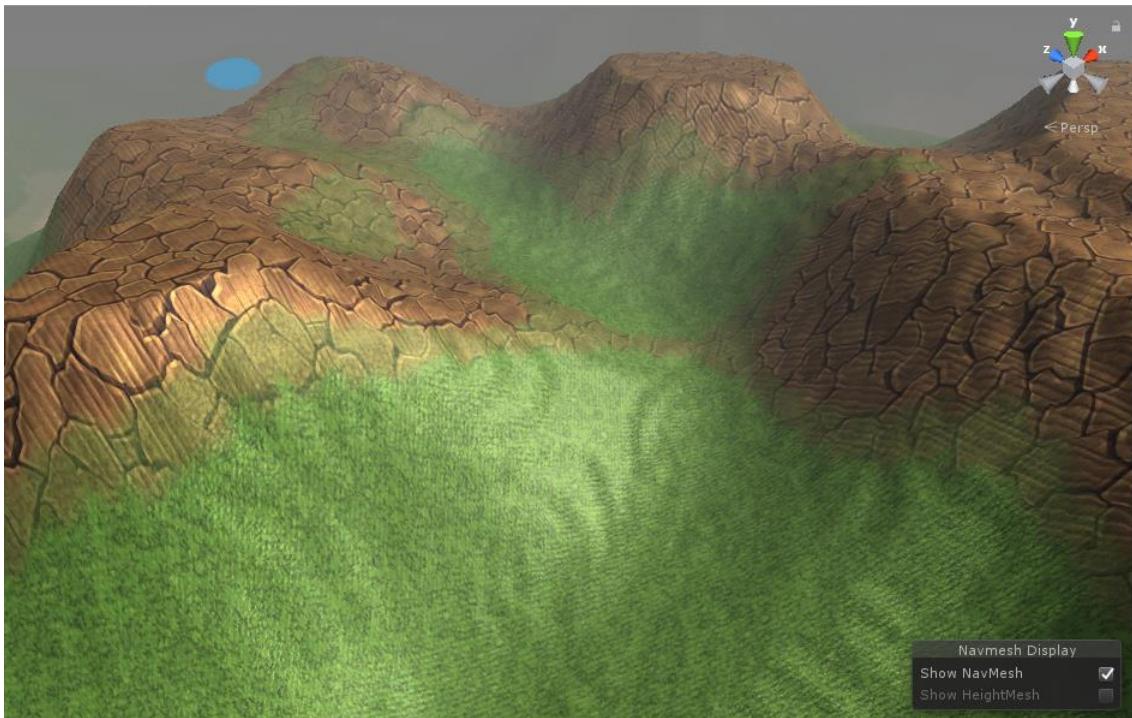


FIGURA 122. TEXTURAS DEL TERRENO CON MAPAS DE NORMALES

5.3.2.3 Generación de fauna, vegetación y otros elementos

Para la generación aleatoria de los distintos elementos a lo largo de la isla lo he implementado de tal manera que se generen por grupos.

En cada una de las 5 alturas, se calculan posiciones aleatorias. Estas posiciones aleatorias representan el centro de un círculo (invisible) con un radio aleatorio. A continuación, se calculan posiciones aleatorias dentro de estos círculos, siendo aquí donde se generan los diferentes elementos correspondientes en función de la altura.

La función encargada de realizar esto se encuentra dentro del script de **TerrainGeneration**, y se ejecuta justo después de haber generado el terreno.

5.3.2.4 Mar

El mar es un plano situado en la altura 0 ($y=0$) el cual, mediante un script, los vértices del plano se desplazan en el eje Y generando oleaje.

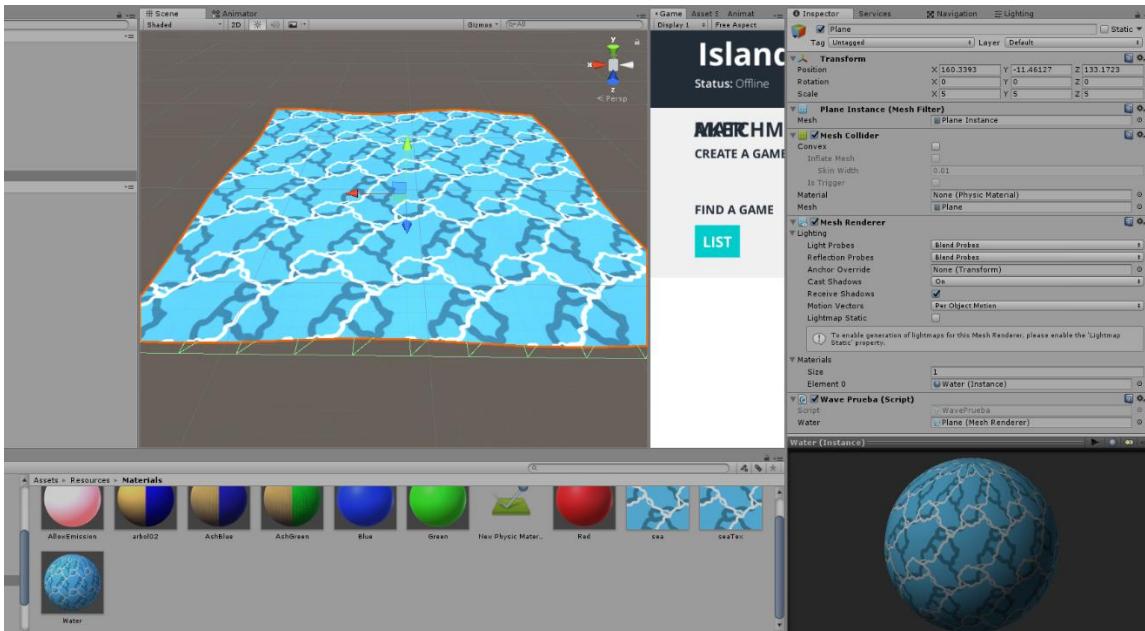


FIGURA 123. CREACIÓN DEL MAR Y SU MATERIAL/TEXTURA

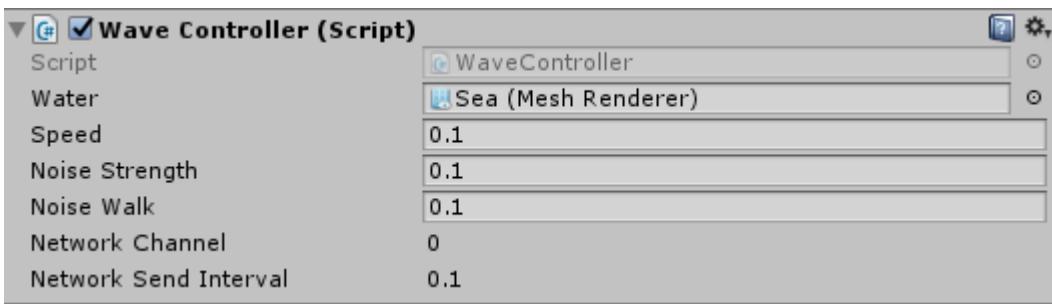


FIGURA 124. SCRIPT WAVECONTROLLER, PARA CONFIGURAR EL OLEAJE DEL MAR

El script permite configurar algunos parámetros como la velocidad en la que se desplazan los vértices. Además, el script está programado para que la textura del mar se desplace ligeramente en diagonal, para dar una mejor sensación de movimiento.

El material del mar es el siguiente:

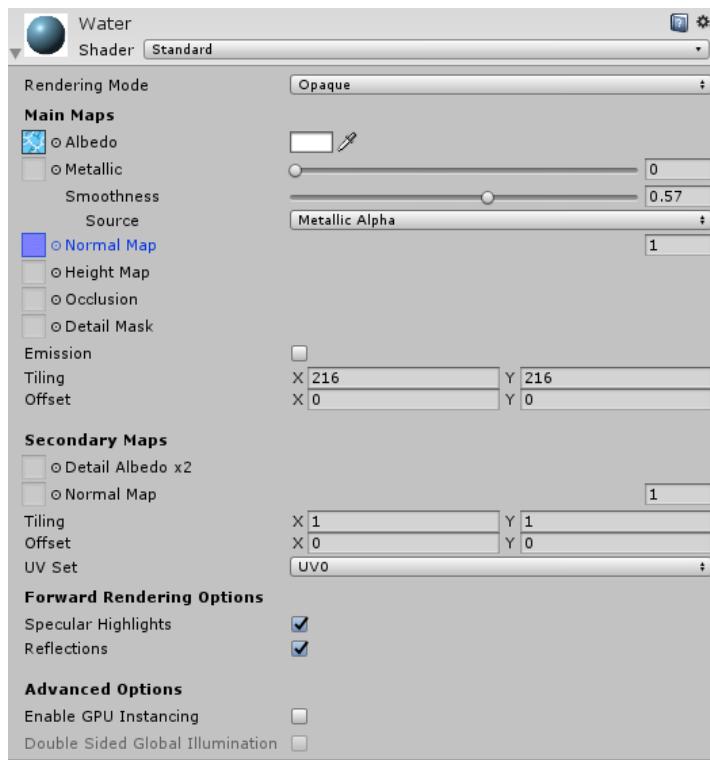


FIGURA 125. MATERIAL DEL MAR

La textura que se ha seleccionado como mapa de normales aporta un efecto como tela de plástico, además el parámetro *Smoothness* en 0.57 permite que la iluminación del Sol ayude a la simulación de este efecto.



FIGURA 126. RESULTADO DEL MAR CON EL MATERIAL CREADO. REFLEXIÓN DE LA LUZ DEL SOL

5.3.3 Personaje principal

El personaje principal de este videojuego es un muñeco de papel.

5.3.3.1 Modelado y texturizado

El muñeco de papel se ha modelado en 3ds max a partir de una esfera.

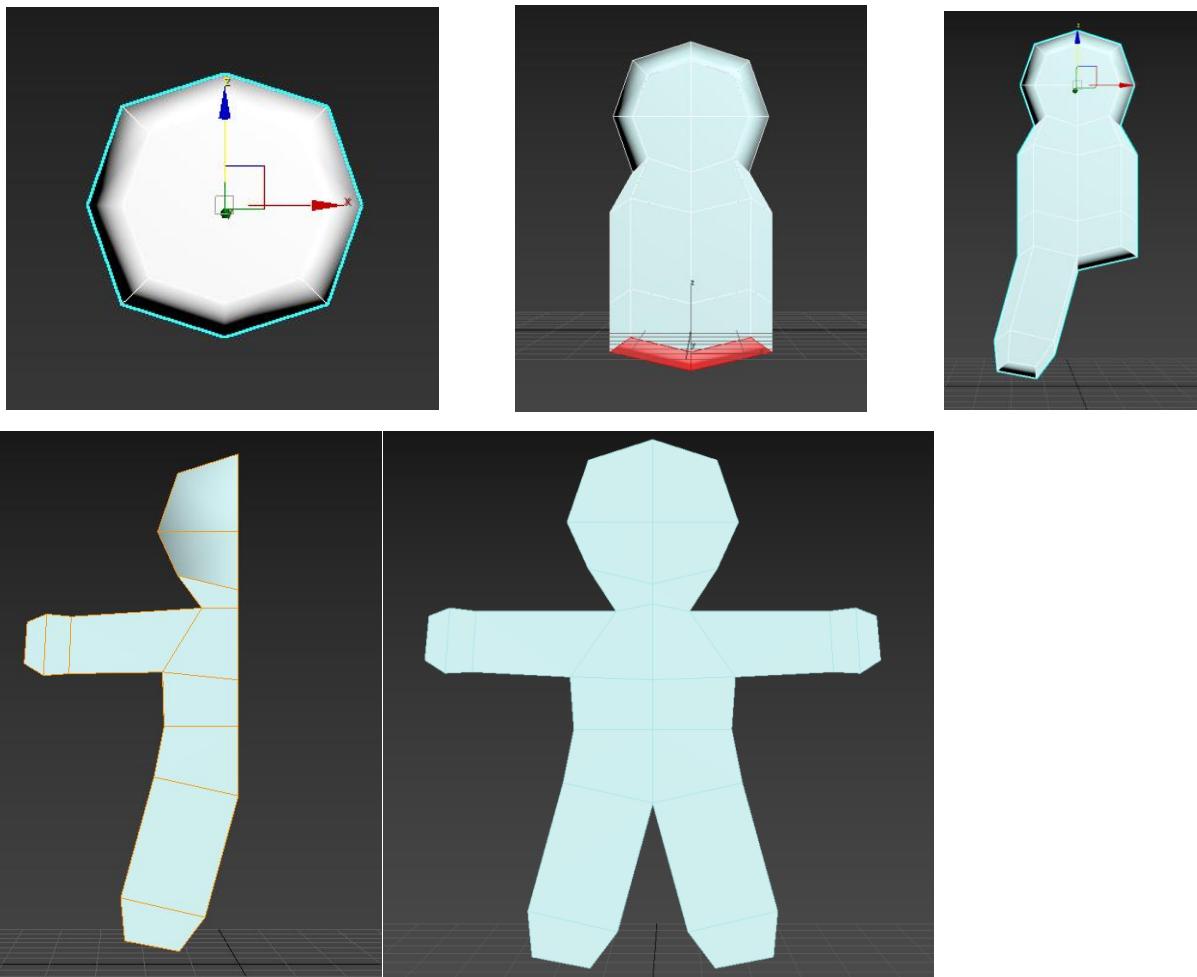


FIGURA 127. IMÁGENES DEL PROCESO DE MODELADO DEL PERSONAJE PRINCIPAL

El texturizado se realiza en Unity. Nada más empezar la partida en función del color del personaje se le asigna un material de un color básico, además de un mapa de normales que simula la textura de papel.

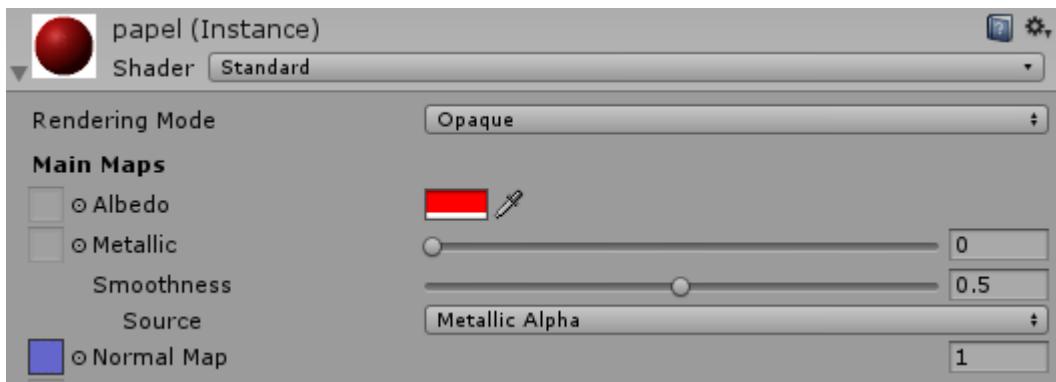


FIGURA 128. PARÁMETROS DEL MATERIAL DEL PERSONAJE PRINCIPAL



FIGURA 129. RESULTADO DEL PERSONAJE PRINCIPAL CON EL MATERIAL CONFIGURADO ANTERIORMENTE

5.3.3.2 Rigging y skinning

El *rigging* y *skinning* se ha realizado en Maya, creando y adaptando un esqueleto humanoide para el personaje, y ajustando el peso de los huesos a la maya.

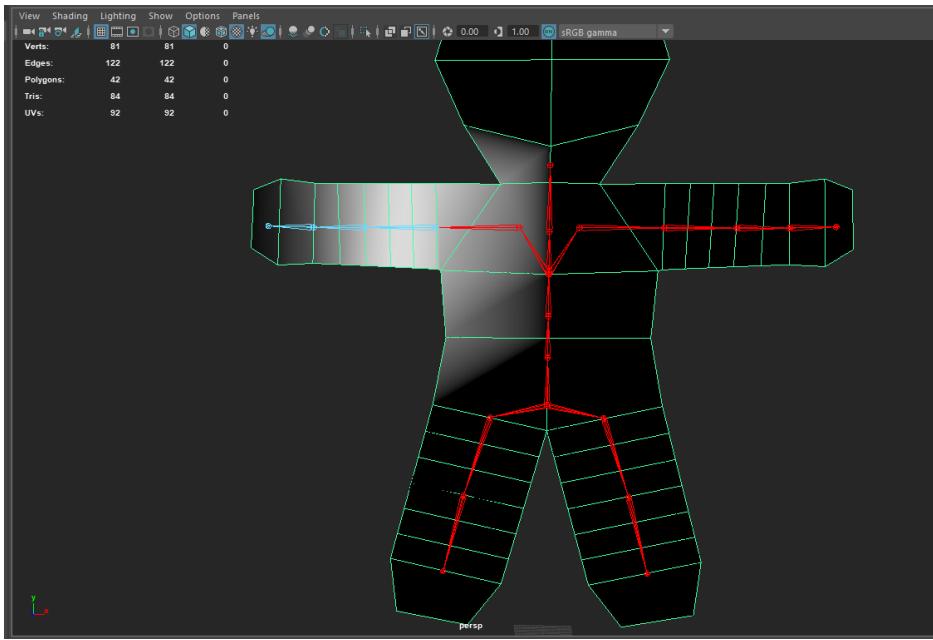


FIGURA 130. SKINNING DEL PERSONAJE PRINCIPAL

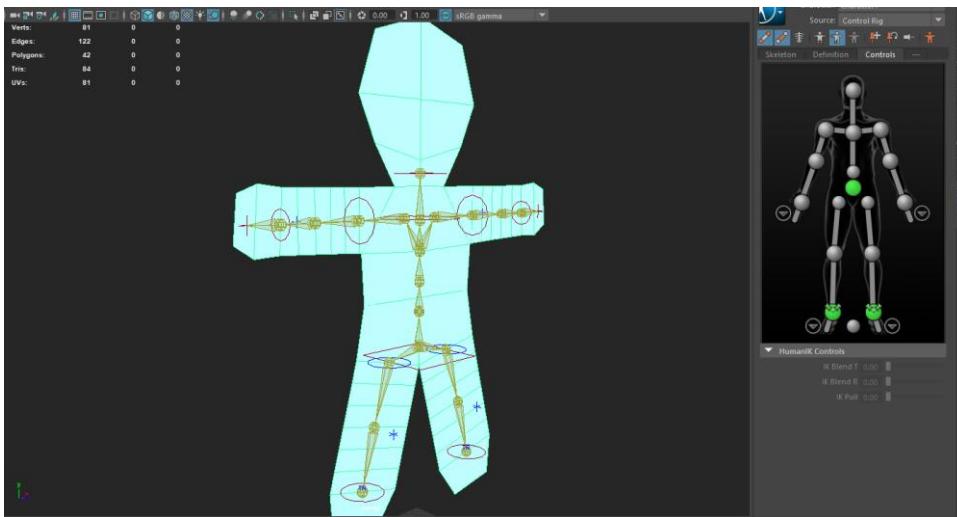


FIGURA 131. RIGGING DEL PERSONAJE PRINCIPAL

5.3.3.4 Controles y animaciones

Las animaciones del personaje se han descargado directamente desde la página de adobe **Mixamo**. De esta manera, se pueden aplicar de manera directa al esqueleto del modelo anterior sin ningún problema. Para ello, utilizaremos el motor de animaciones de Unity, creando una componente *Animator* en el *GameObject* del jugador.

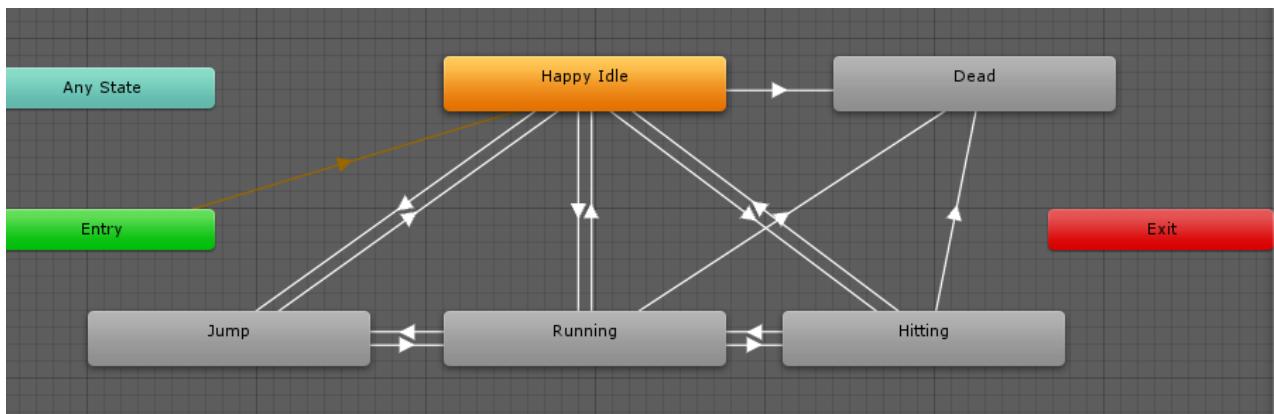


FIGURA 132. TRANSICIÓN DE ANIMACIONES DEL PERSONAJE PRINCIPAL



FIGURA 133. PARÁMETROS DE LAS ANIMACIONES DEL PERSONAJE PRINCIPAL

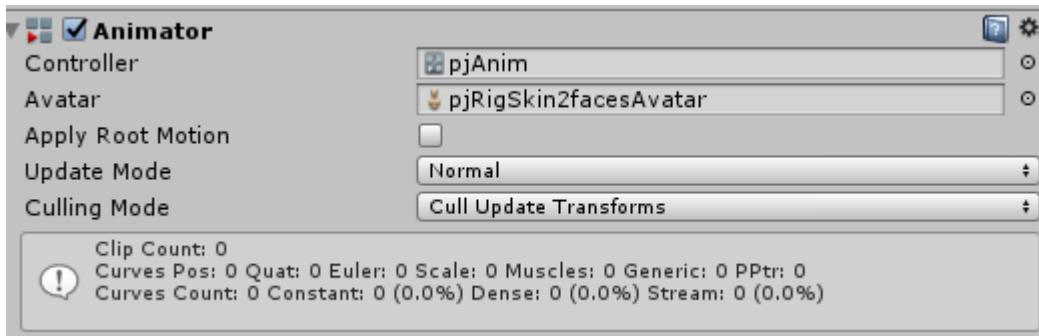


FIGURA 134. COMPONENTE ANIMATOR DEL PERSONAJE PRINCIPAL

Los personajes realizan animaciones al golpear, saltar, correr, morir, y al estar parados.

Para el control del personaje se ha implementado un script que recibe el nombre de **PjControl**.

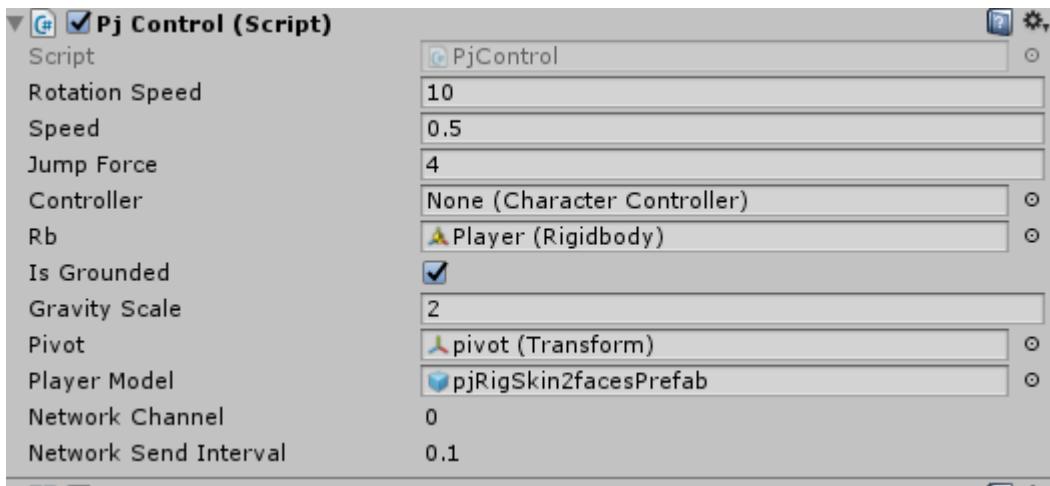


FIGURA 135. SCRIPT PJCONTROL

En él se puede configurar la velocidad a la que se desplaza, fuerza del salto, velocidad de rotación, gravedad, entre otros parámetros. Es en este script donde se realizan las comprobaciones cuando se pulsa una tecla o se presiona algún botón del ratón. Cuando se realiza una acción se ejecuta el trigger o se utiliza una variable del tipo bool para activar las animaciones correspondientes. Por otra parte, es importante detectar cuándo un personaje se encuentra tocando el suelo (isGround).

Para golpear y ocasionar daños los jugadores tienen un gran *collider* en la mano derecha.

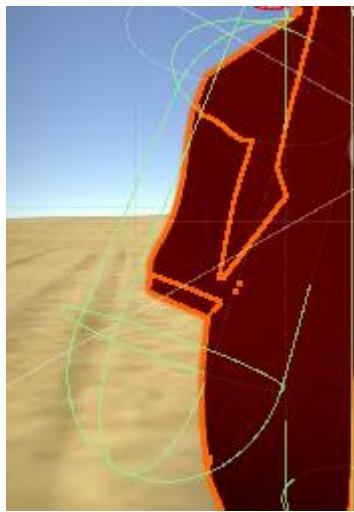


FIGURA 136. COLLIDER EN EL BRAZO DERECHO DEL PERSONAJE PRINCIPAL

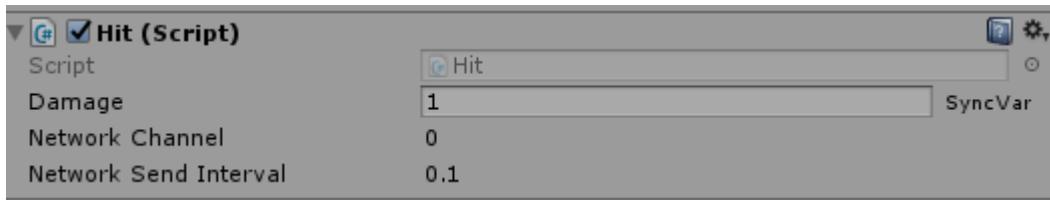


FIGURA 137. SCRIPT HIT

El script **Hit** se encarga de ocasionar daño al colisionar con los *collider* de los demás *GameObject* que tengan un script de **Health**, usando las diferentes funciones para realizar el cálculo de daño.

5.3.3.5 Características del personaje

Los personajes cuentan con dos características importantes, la vida y el nivel de color.



FIGURA 138. PERSONAJE PRINCIPAL CON MEDIDORES DE VIDA Y NIVEL DE COLOR, ADEMÁS DE SU NOMBRE DE STEAM

Y los scripts que se encargan de gestionar ambas características son los siguientes:

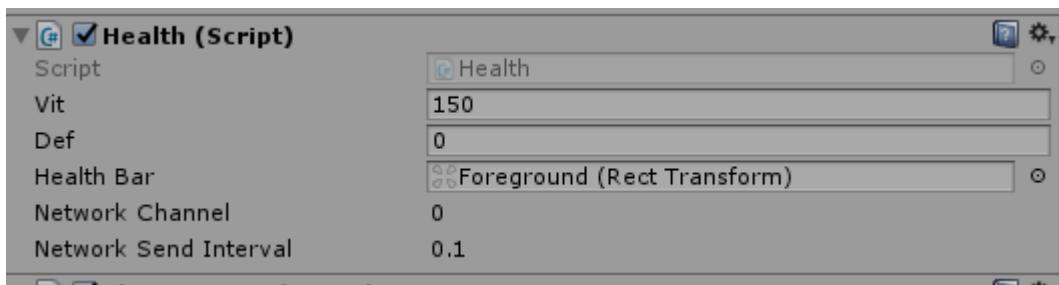


FIGURA 139. SCRIPT HEALTH

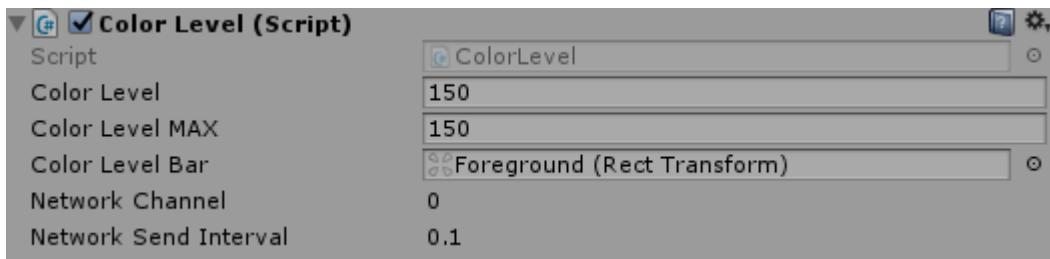


FIGURA 140. SCRIPT COLORLEVEL

La pérdida progresiva tanto de nivel de color como de vida está implementada mediante una co-rutina, donde se sincronizan ambas variables a través de un Command, y se actualizan las barras en todos los clientes con un RpcClient. Si el personaje tuviese defensa (con una armadura equipada) la vida que pierda al sufrir daño por parte de un animal será menor. Además, la pérdida de nivel de color será mucho mayor si el jugador se encuentra en contacto con el mar o la lluvia.

Por otro lado, el siguiente script se encarga de inicializar al inicio de la partida 3 parámetros importantes de cada jugador: el nombre del personaje, el color, y un string del color.

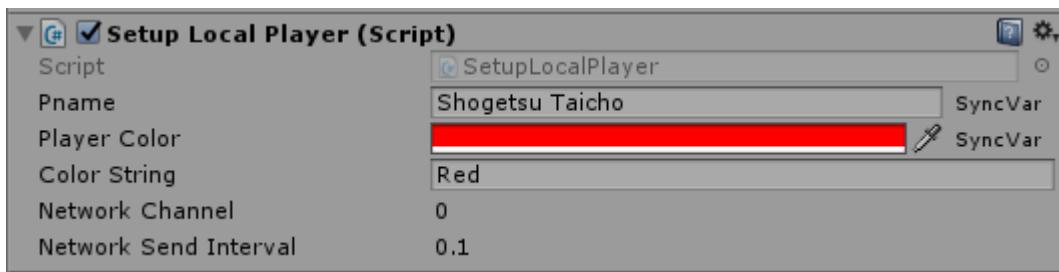


FIGURA 141. SCRIPT SETUPLOCALPLAYER

Finalmente, destacar que el nombre que se muestra sobre el jugador es el mismo nombre que utiliza en la aplicación de Steam.

5.3.4 Otros personajes y elementos del mundo

5.3.4.1 Fauna

La fauna está compuesta por 4 personajes diferentes.



FIGURA 142. RESULTADO DEL MODELADO Y TEXTURIZADO DE LOS ANIMALES

Estos personajes se han modelado todos en 3ds max.

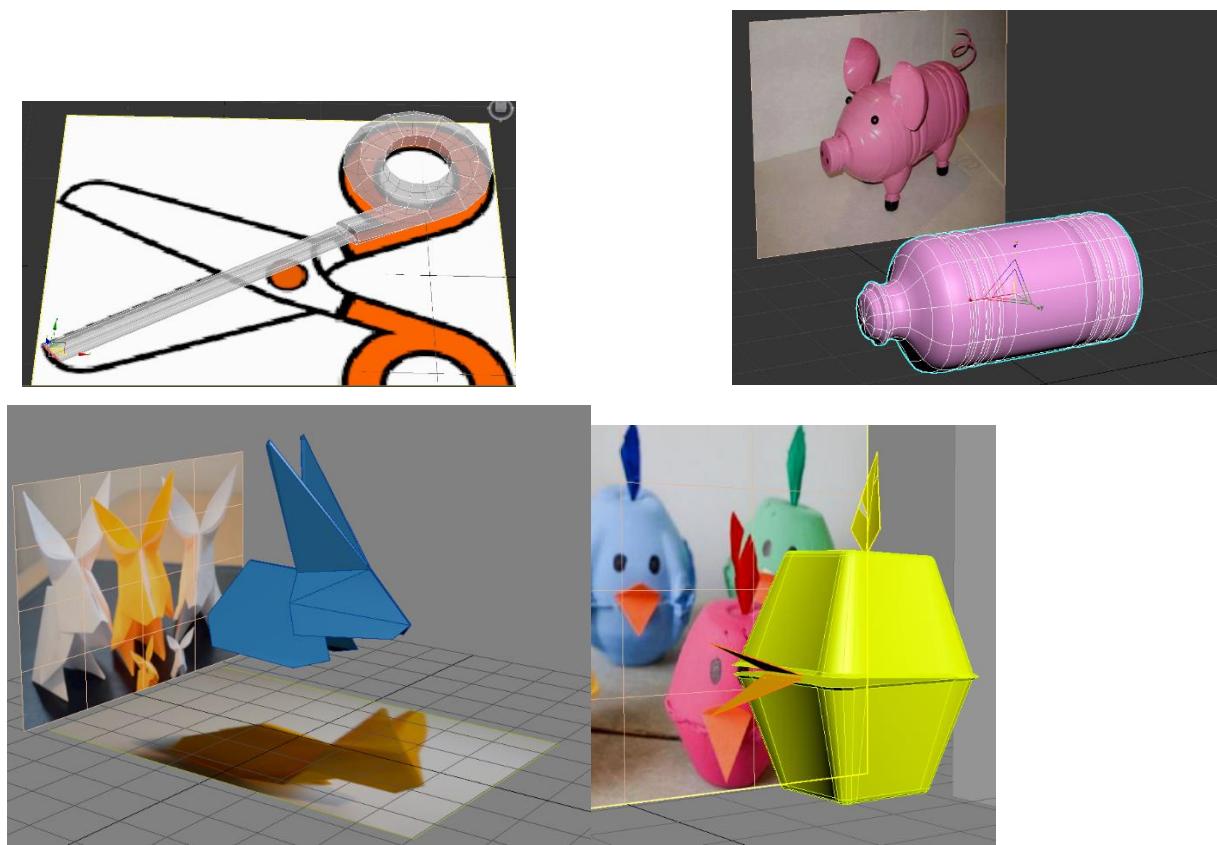


FIGURA 143. ANIMALES EN PROCESO DE MODELADO

Y se han texturizado en 3ds max, con la ayuda de Photoshop, haciendo uso de una técnica de texturizado que recibe el nombre de ***Lazy Unwrapping***, la cual consiste en generar una pequeña textura en Photoshop con varios rectángulos de **colores con degradado**, y luego en el programa de modelado se posicionan los diferentes UV sobre la textura, dando un resultado **simple pero muy vistoso**.

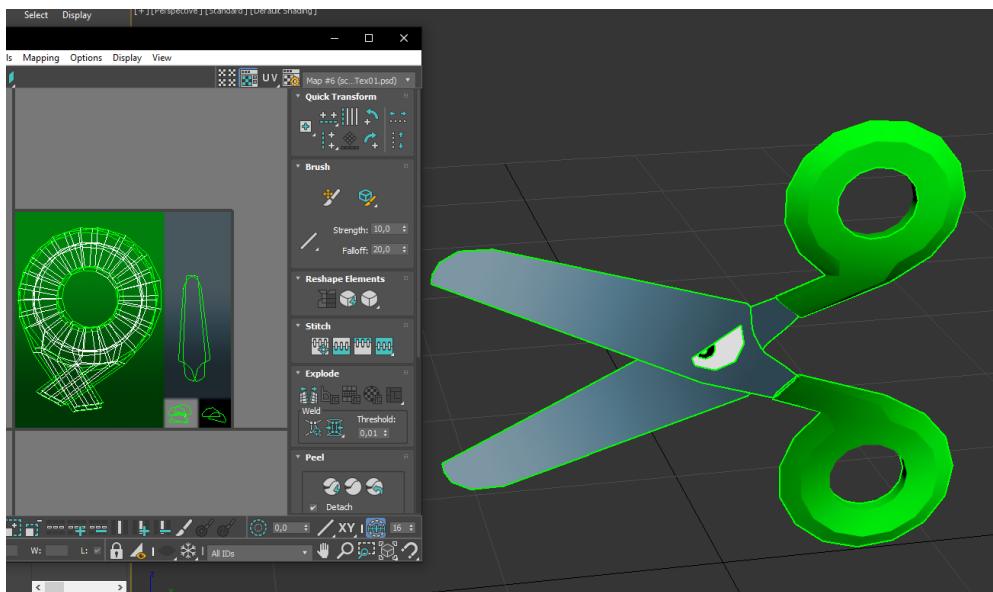


FIGURA 144. EJEMPLO DE TEXTURIZADO. TEXTURIZANDO LAS TIJERAS ANIMADAS

En cuanto a las animaciones, para los personajes de la fauna solamente se han realizado 2 tipos de animaciones, de andar y de atacar. Han sido realizadas directamente sobre el editor de animaciones de Unity, combinando cambios de posición y rotación simples.

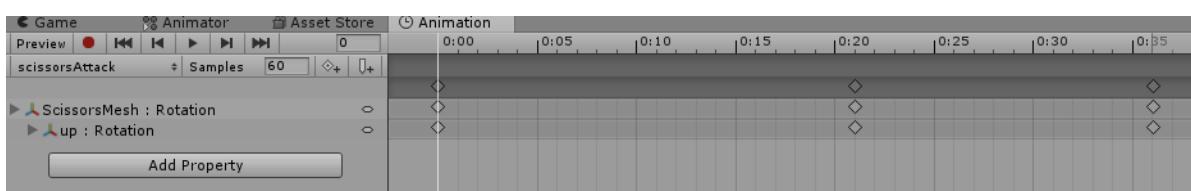


FIGURA 145. EJEMPLO DE CREACIÓN DE ANIMACIONES EN EL EDITOR DE UNITY

Y, posteriormente, las animaciones se añaden sobre una componente *Animator*, pudiendo hacer transiciones suaves entre las animaciones.

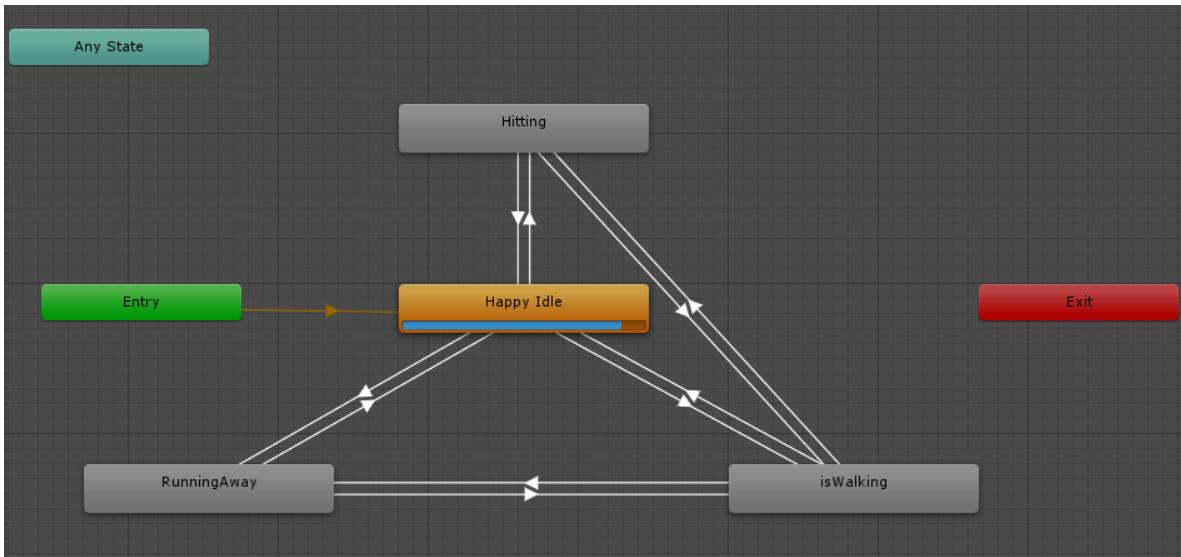


FIGURA 146. ESTADOS Y ANIMACIONES DE LOS ANIMALES

Por otra parte, los personajes de la fauna contienen una IA bastante sencilla, en la cual se realiza una serie de cambios de estados mediante el uso de **boolean**, teniendo únicamente 3 estados: de relax, huyendo, y persiguiendo.

El script encargado de gestionar estos tres estados es **AnimalIA**, y tiene en cuenta también la hora del día (esto se verá más adelante), puesto que esto es determinante para que algunos animales se vuelvan agresivos (o no).

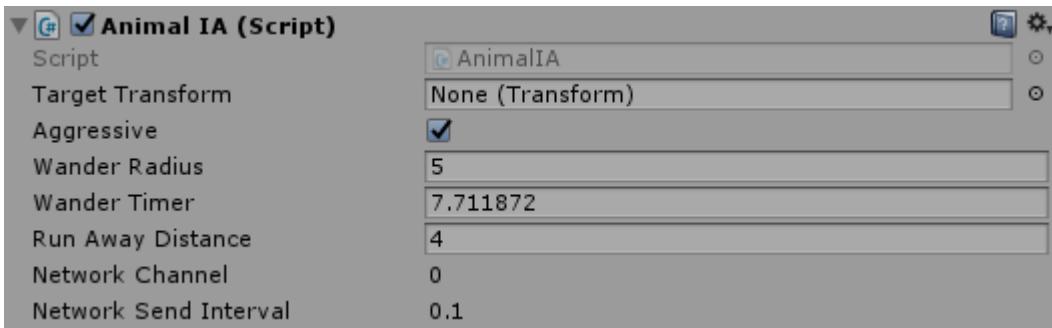


FIGURA 147. SCRIPT ANIMALIA

En él se pueden configurar ciertos parámetros como el radio de visión, un objetivo, o la distancia a la que huyen (cuando estén huyendo). Si la *checkbox* de *Aggressive* se encuentra seleccionada, el animal perseguirá y atacará a un jugador en cuanto se encuentre con uno.

Importante destacar que, para hacer posible la Inteligencia Artificial de los animales, es necesario construir un **navmesh** sobre el terreno. Esto es, una maya invisible sobre la

que la IA será capaz de caminar. La **navmesh** se genera al inicio de la partida, durante la creación del mapa.

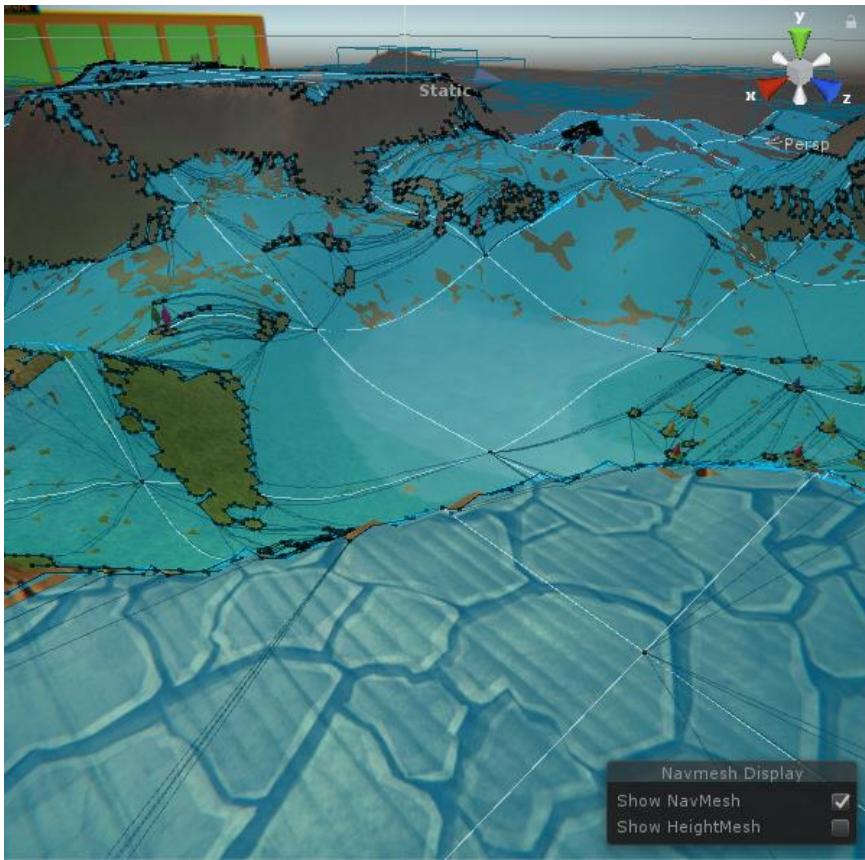


FIGURA 148. MOSTRANDO EL NAVMESH SOBRE EL TERRENO

Para que la IA sea capaz de caminar por el navmesh, es importante añadirle una componente **NavMeshAgent**, pudiendo configurar la velocidad a la que caminan, entre otros muchos parámetros.

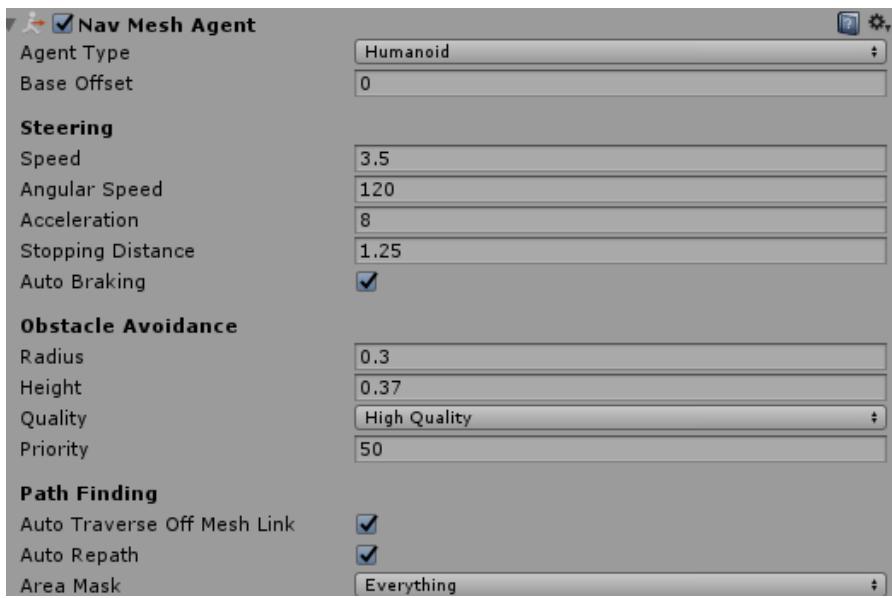


FIGURA 149. COMPONENTE NAVMESHAGENT

Por otro lado, cuando un animal es eliminado suelta ítems. Esto es controlado gracias al script de **Drop**, en el cual se establece una cantidad mínima y máxima de objetos que soltará el animal al morir, y se generará la cantidad de manera aleatoria en función de estos dos valores. Además, en este script se tiene en cuenta el material del animal (Si es papel, cartón, o plástico) además de su color, y soltará ítems acordes a estas dos características.

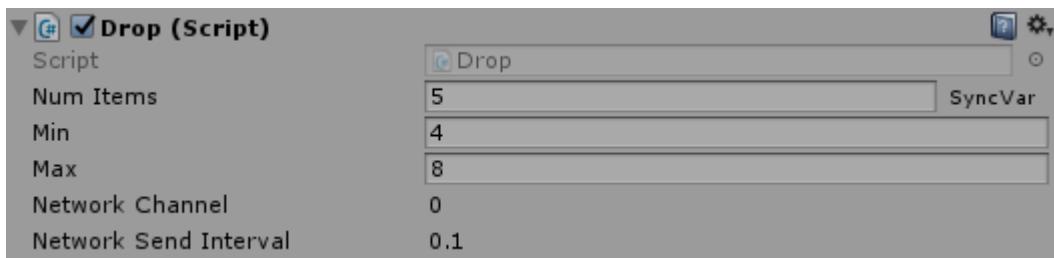


FIGURA 150. SCRIPT DROP

Las características (color y material) se establecen con la ayuda del siguiente script.

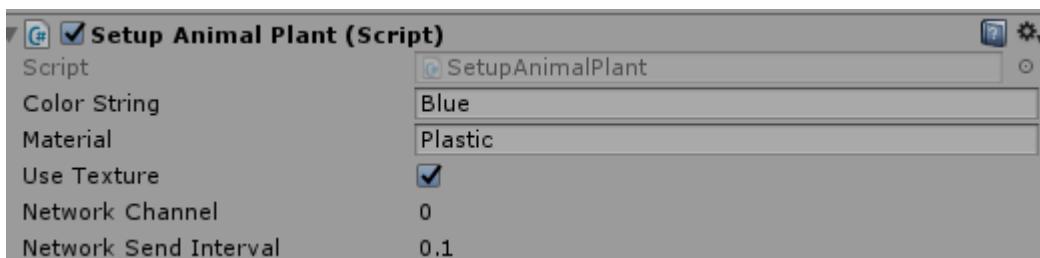


FIGURA 151. SCRIPT SETUPANIMALPLANT

El color se establece de manera aleatoria, al iniciar la partida, entre todos los colores disponibles (Azul, Cian, Magenta, Verde, Amarillo, y Rojo), de forma que el script buscará entre los archivos del juego la textura del color que le corresponda, y se lo asignará al animal. Además, el material no será aleatorio, irá en función del animal correspondiente, siendo esta última característica establecida de forma manual por el programador en forma de *string*, mientras que el script se encargaría de buscar el material correspondiente entre los archivos del juego (el material vendría determinado estéticamente por el normal map).

Finalmente, los animales cuentan con un script que sirve para hacer el cálculo de daño a la hora de golpear, configurar la distancia mínima a la que deben acercarse de su objetivo, y el tiempo que debe transcurrir entre ataque y ataque.

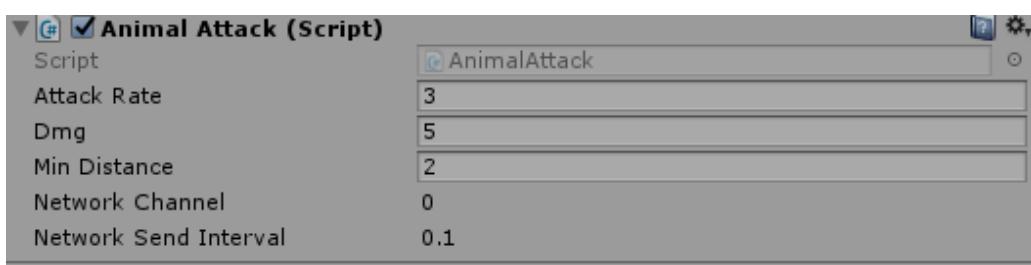


FIGURA 152. SCRIPT ANIMALATTACK

5.3.4.2 Vegetación

La vegetación está compuesta por 5 elementos.



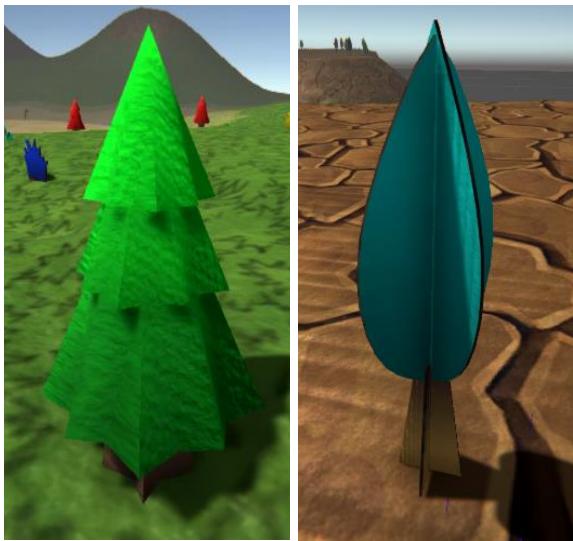
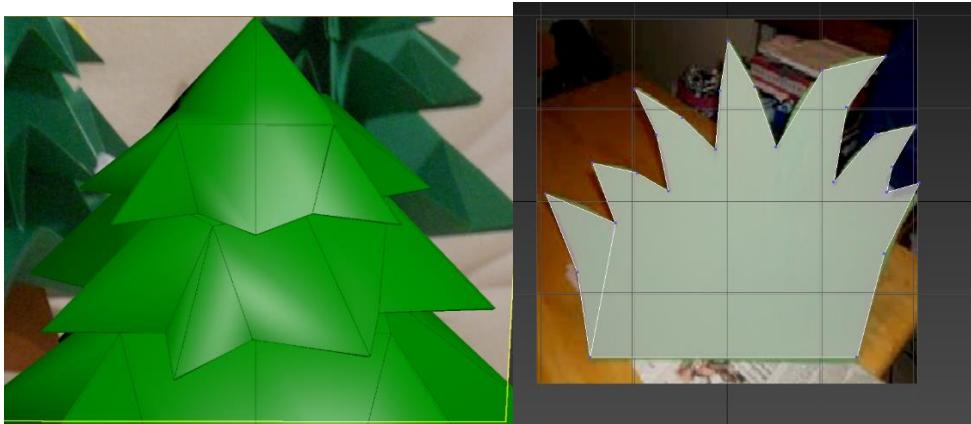


FIGURA 153. RESULTADO DEL MODELADO Y TEXTURIZADO DE LA VEGETACIÓN DENTRO DEL JUEGO

La vegetación se ha modelado y texturizado en 3ds max, al igual que la fauna, excepto las palmeras y los fresnos, que empecé modelando en Maya al inicio del proyecto.



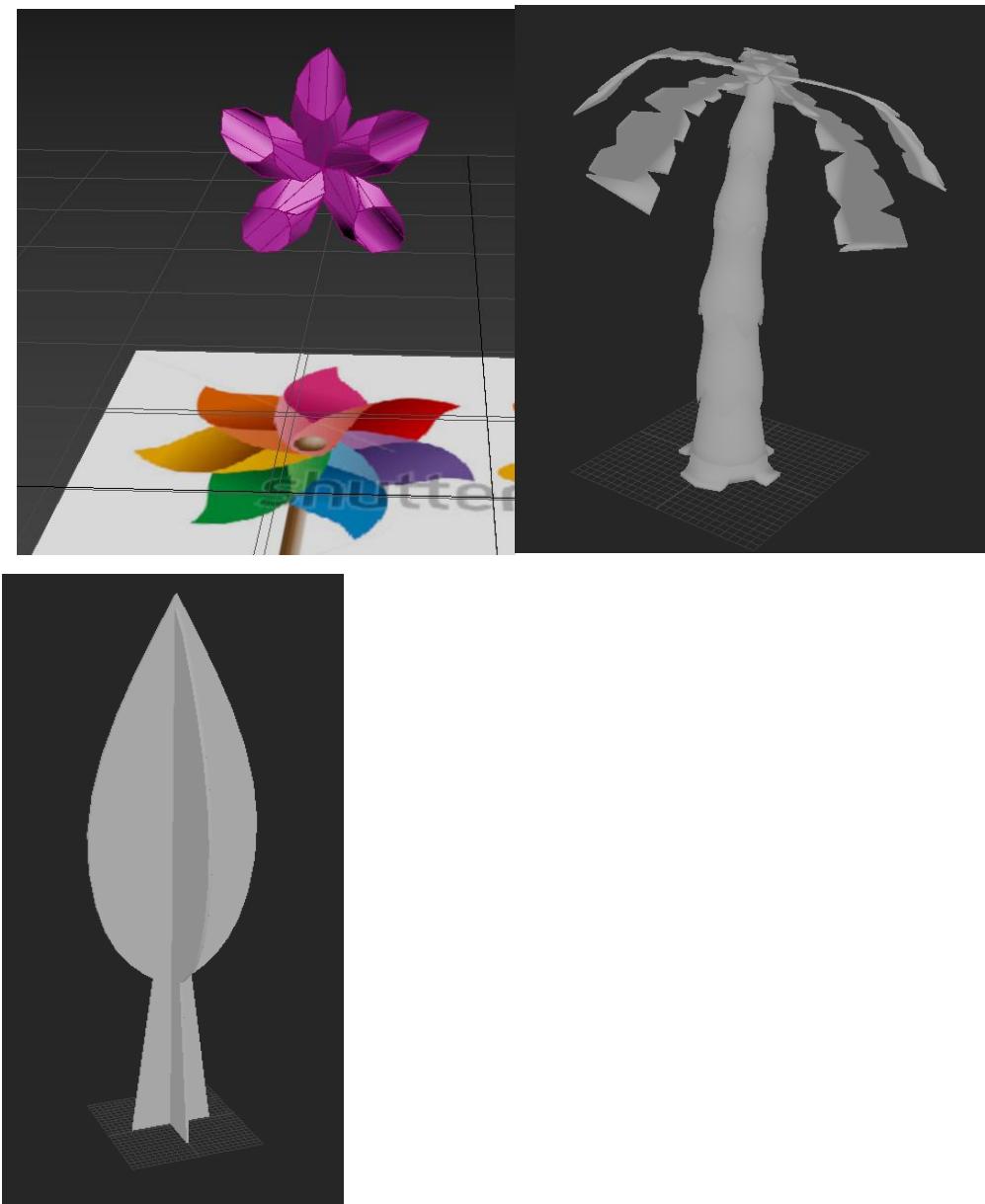


FIGURA 154. MODELADO DE LA VEGETACIÓN EN PROCESO

Por otro lado, la vegetación contiene prácticamente los mismos scripts que la fauna (excepto los de IA y ataque), como Drop para la caída de objetos en función de su color y material, Health para la vida, y SetupAnimalPlant para establecer sus colores y materiales al inicio de la partida.

5.3.5 Inventario y sistema de *crafting*

5.3.5.1 Objetos

Para los objetos del juego he creado assets.

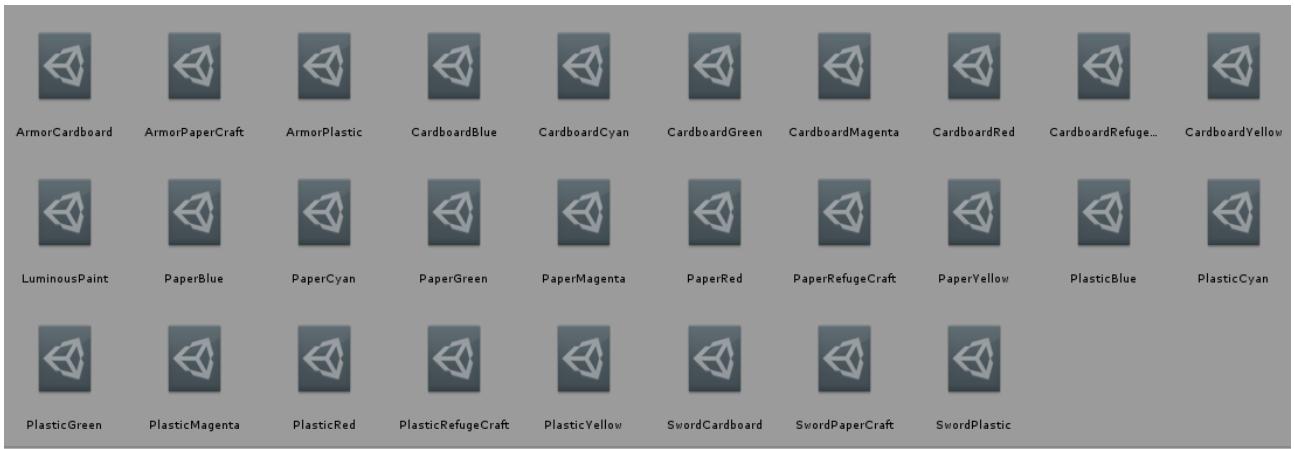


FIGURA 155. ASSETS DE LOS OBJETOS DEL JUEGO

En los cuales se establece una clase (script) diferente según el objeto que se corresponda, en el caso de la siguiente imagen es un *Resource* o recurso.

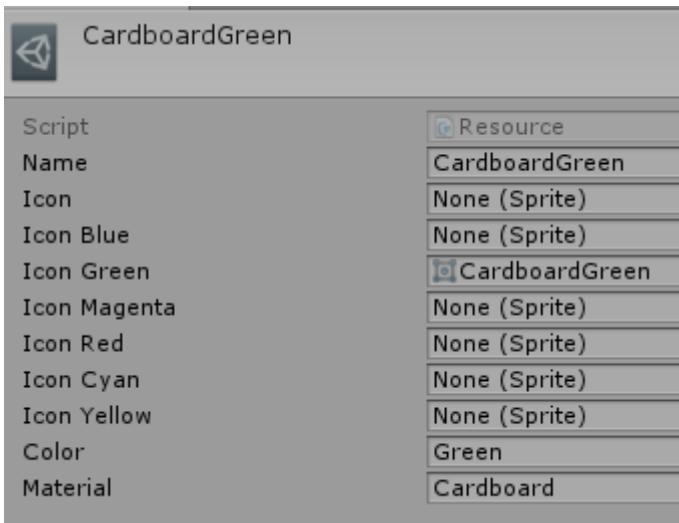


FIGURA 156. EJEMPLO DE LA CONFIGURACIÓN DE UN ASSET DE UN OBJETO DEL JUEGO

El siguiente diagrama de clases refleja la jerarquía de los diferentes tipos de objetos que se han implementado en el videojuego.

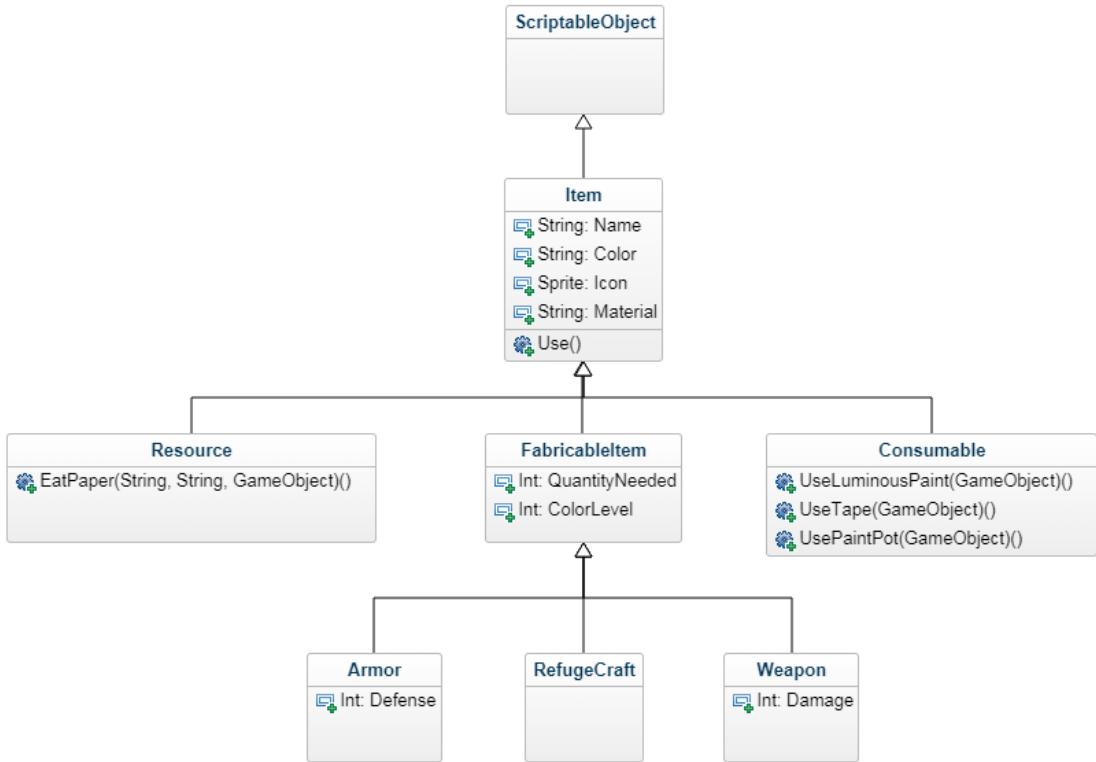


FIGURA 157. DIAGRAMA DE CLASES SOBRE LA JERARQUÍA DE LOS ÍTEMES DEL JUEGO

La clase *Item* hereda de *ScriptableObject* lo que me permite crear los Asset directamente desde el editor de Unity, con clic derecho y pulsar sobre *Create*.

Los Items que sean **Resources** podrán ser comidos por los jugadores, siempre y cuando sean de material Papel, mediante la función *EatPaper*. Todos los resources irán destinados principalmente a la creación de objetos (que se explicará más tarde).

Los **FabricableItem** son ítems fabricables que contienen un entero indicando la cantidad necesaria para su creación, además de un nivel de color que indicará el desgaste de estos. Estos objetos son refugios, que se podrán invocar sobre el terreno, y armaduras y espadas que se podrán equipar mediante el inventario (que se explicará más tarde).

Sobre los ítems **Consumable** son objetos que se consumen directamente al usarse, como los *LuminousPaint* que otorgan iluminación temporal sobre el cuerpo del jugador activando mediante un script un **PointLight** que portará el *GameObject* del jugador, además de cinta adhesiva (*Tape*) que curan por completo la vida del jugador que lo utilice, o *PaintPot* que restaurará por completo el nivel de color del jugador.



FIGURA 158. SCRIPT ACTIVEPOINTLIGHT

El script **ActivePointLight** se encarga de encender la luz temporal del jugador al consumir *LuminousPaint*. En este script se tendrá en cuenta el color del personaje que consuma el objeto, puesto que la luz que emitirá éste será del mismo color que el personaje. Importante destacar que, en este script, se accede a la componente **Renderer** del jugador, y se modifica su material para que emita luz.

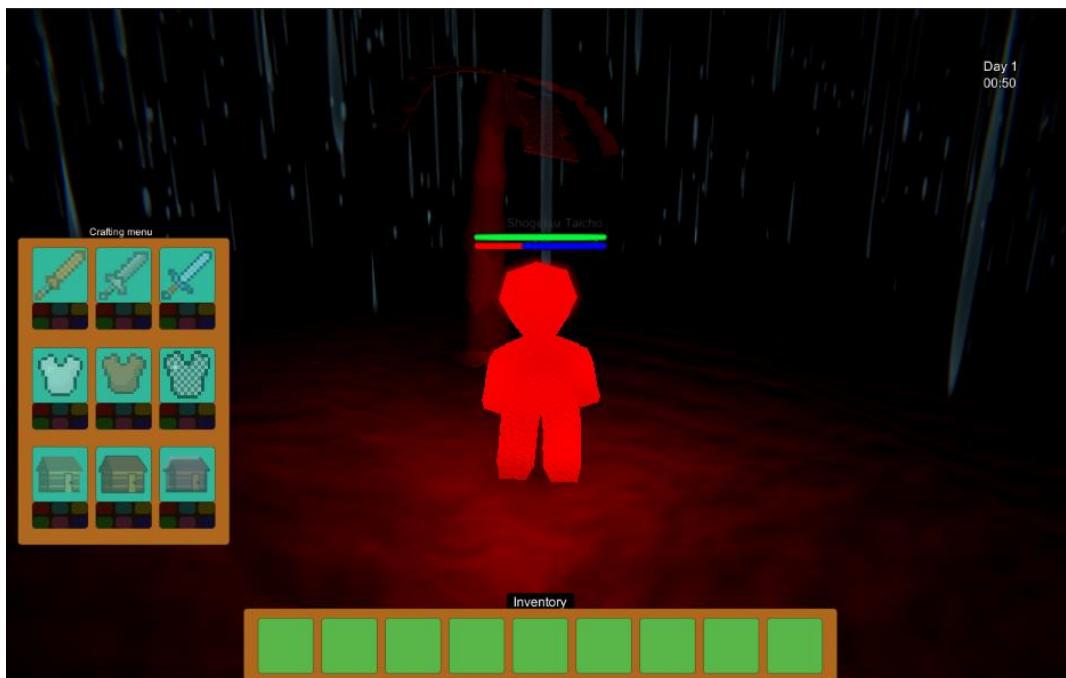


FIGURA 159. RESULTADO DE LA ILUMINACIÓN DEL CUERPO DEL PERSONAJE PRINCIPAL

La armadura, el refugio, y la espada, se han modelado mediante 3ds max.

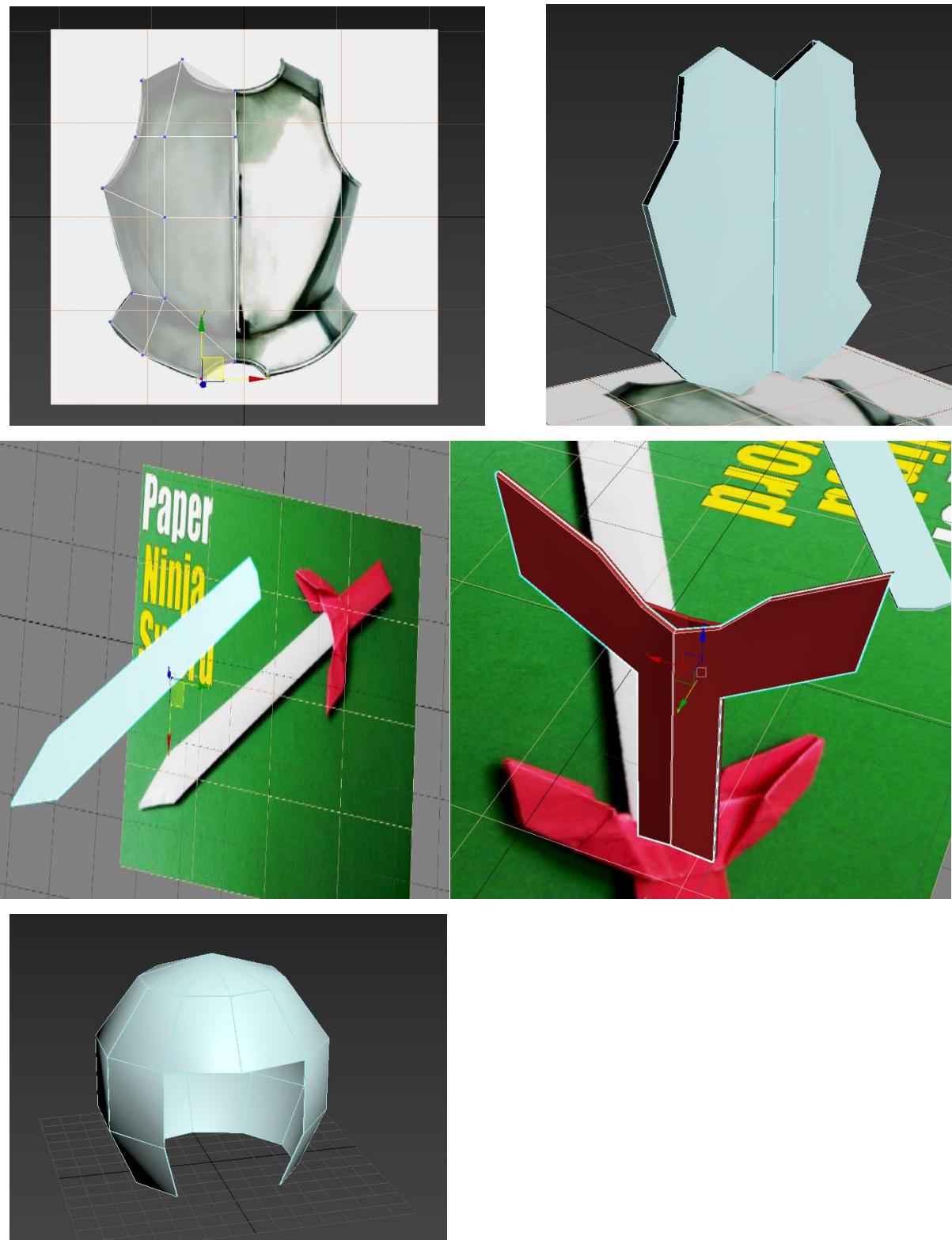


FIGURA 160. OBJETOS EN PROCESO DE MODELADO

Todos los modelos se texturizan en tiempo de ejecución durante la partida, asignando un mapa de normales según su material, y una textura de un color básico según su color.

En el caso de los refugios, tienen un script en el cual se establece su nivel de color en función de su material, en el caso de la imagen 50 es el nivel de color para los refugios de papel. El nivel de color de los refugios disminuye con la lluvia, que se detallará más adelante sobre su implementación.

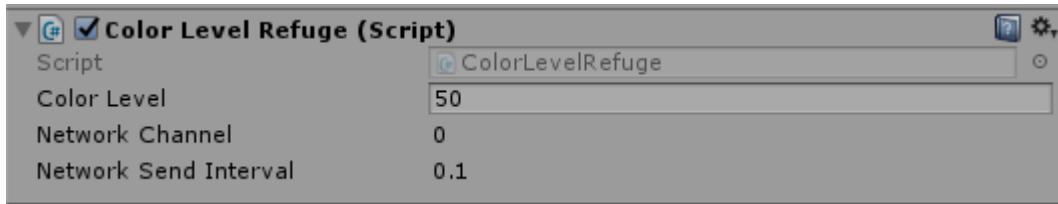


FIGURA 161. SCRIPT COLORLEVELREFUGE

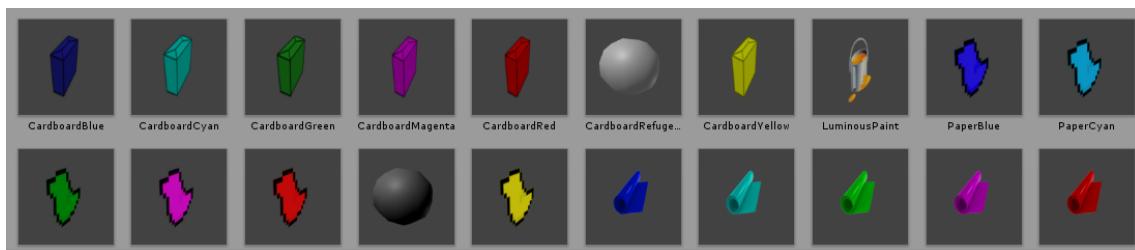


FIGURA 162. PREFABS DE OBJETOS DEL JUEGO

Por último, cabe destacar que los objetos que se pueden recolectar (recursos y consumibles) no tienen un modelo 3D como tal, sino que son simples *sprites* que se encuentran flotando sobre el terreno. Destacar que los *sprites* que se muestran en las imágenes están descargados de internet y no me pertenecen.

5.3.5.2 Inventario

El inventario está compuesto por un panel de 9 casillas que se muestran en el canvas, y viene gestionado por el script **Inventory**, el cual se encuentra dentro del *GameObject* del jugador.

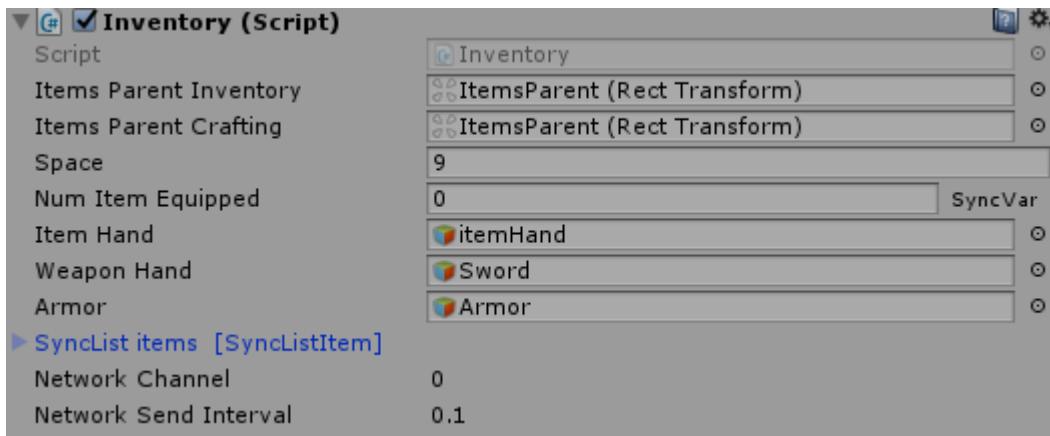


FIGURA 163. SCRIPT INVENTORY

Los usuarios pueden coger objetos (recursos y consumibles) directamente del suelo, simplemente colisionando con ellos. Para la colisión entre el jugador y un objeto que se pueda recoger, hay un script dedicado a ello.

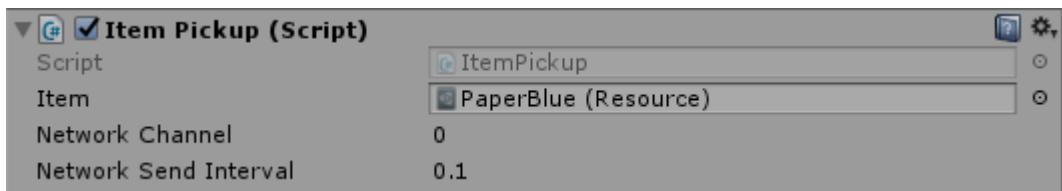


FIGURA 164. SCRIPT ITEM PICKUP

Este script se encuentra dentro del *prefab* del objeto que se puede recoger, y permite dar a conocer al jugador qué ítem es el que está intentando añadir y, finalmente, añade a su inventario, gracias a la variable *Item* del script, la cual posee el *Asset* del objeto que le corresponda al *prefab*.

Una vez el jugador colisiona con el *GameObject* de un objeto que posee el *script* anterior, primero se realiza una comprobación en el script de **Inventory** sobre si el objeto que se intenta añadir realmente está en el juego, y, en caso afirmativo, aparecerá el *sprite* de este ítem en el inventario.

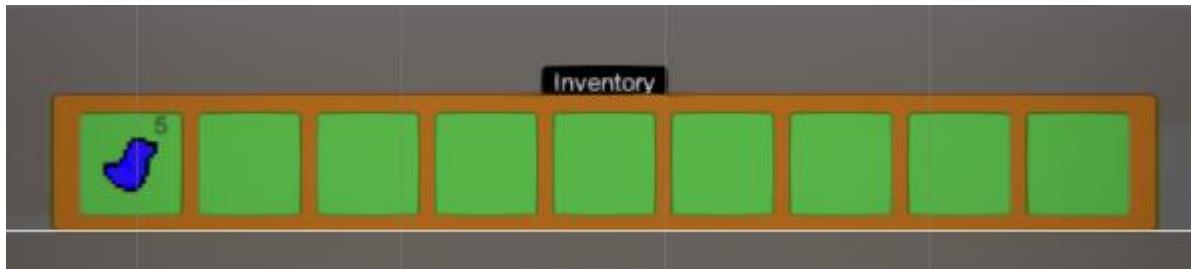


FIGURA 165. INVENTARIO CON 5 RECURSOS IGUALES

Es importante destacar que la sincronización de los objetos del inventario con el servidor es posible debido a la utilización de una **SyncList**, y la correcta utilización de las funciones **Command** y **ClientRpc**.

Los únicos objetos acumulables en el inventario son los recursos (de manera ilimitada), el resto no se acumularán y cada uno consumirá una casilla de espacio.

Mediante las teclas numéricas que se encuentran en la parte superior del teclado, es posible equipar los objetos (solo uno) en mano y, una vez hecho esto, será posible utilizarlos con clic derecho, siempre y cuando éstos tengan alguna utilidad.

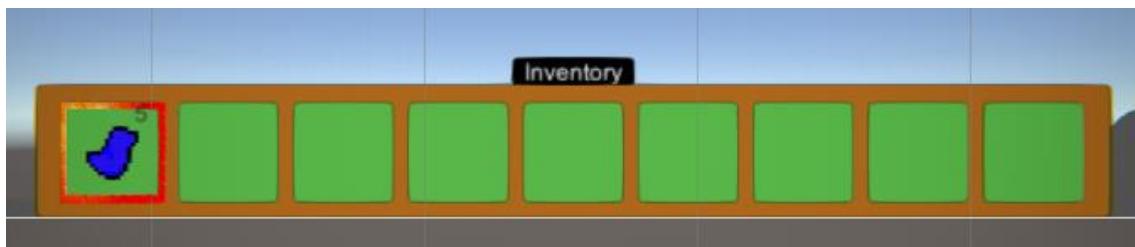


FIGURA 166. INVENTARIO CON UN OBJETO EQUIPADO EN MANO

Al equiparlo, el *sprite* de este ítem aparecerá en la mano del modelo 3D del jugador, y en el inventario aparecerá un marco rodeando el ítem equipado en mano.



FIGURA 167. RESULTADO DE UN OBJETO EQUIPADO EN MANO SOBRE EL MODELO DEL PERSONAJE PRINCIPAL

Para la implementación de este sistema, se ha añadido un *GameObject* que posee una componente *sprite* dentro de la jerarquía del *prefab* del jugador.

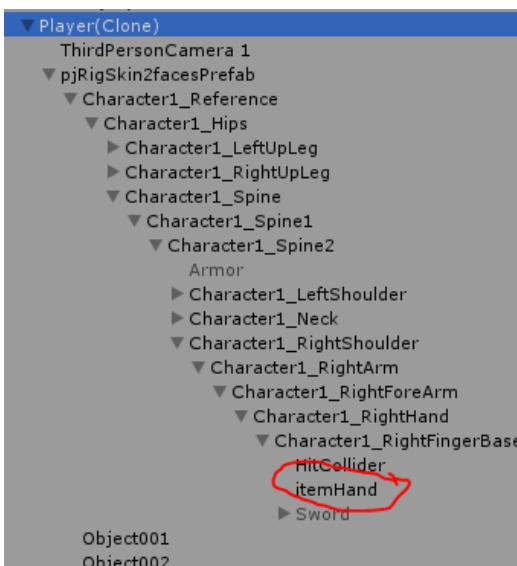


FIGURA 168. GAMEOBJECT ITEMHAND DENTRO DE LA JERARQUÍA DE OBJETOS DEL PREFAB DEL JUGADOR

Cabe destacar la utilización de un *script* para que el *sprite* del objeto equipado en mano siga a la cámara y nunca le dé la espalda, en el cual se juega con la rotación del *sprite* y la posición de la cámara.



FIGURA 169. SCRIPT BILLBOARD

Por otra parte, el jugador podrá equiparse tanto espadas como armaduras sobre su modelo 3D.



FIGURA 170. EJEMPLO DE ESPADA Y ARMADURA EQUIPADA SOBRE EL MODELO DEL JUGADOR

Para adaptar estos modelos sobre la maya del jugador, se ha empleado la misma técnica que el *sprite* del ítem equipado en mano: crear *GameObject* que contengan estos modelos 3D y posicionarlos correctamente dentro de la jerarquía del *prefab* del jugador.

Por otra parte, en el momento en el que, tanto la espada como la armadura, pierdan nivel de color, se verá reflejado sobre los *sprites* de los objetos en cuestión volviéndose más oscuros.

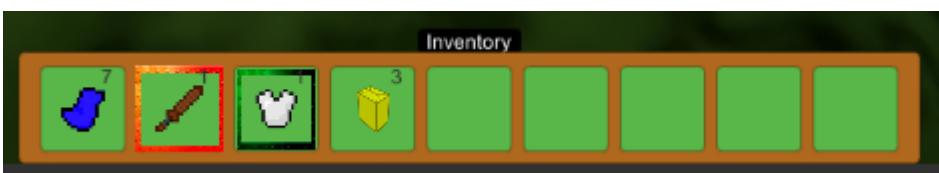


FIGURA 171. EJEMPLO DE ESPADA DETERIORADA, CON MENOS NIVEL DE COLOR QUE EL INICIAL

En cuanto una espada o armadura se rompa, esta desaparecerá del inventario.

También, los jugadores pueden soltar los objetos del inventario en el suelo, presionando la tecla Q, de tal forma que reaparecerán estos objetos delante del jugador que los haya tirado. En caso de echar al suelo cualquier ítem fabricable, éste se romperá y se recuperará una pequeña parte de los recursos empleados en su fabricación. El

cálculo aleatorio de los recursos recuperados se realiza dentro del *script* de **Inventory**, pudiéndose recuperar entre 1, y la cantidad total necesaria entre 1,5 (un 60%).

Finalmente, mencionar que cada una de las casillas del inventario contiene un *script* para gestionar los datos del objeto de cada casilla.

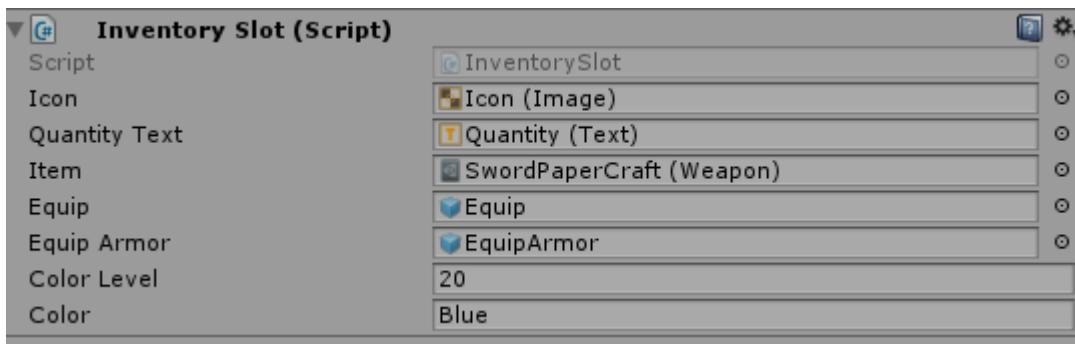


FIGURA 172. SCRIPT INVENTORY SLOT

5.3.5.3 Sistema de fabricación de objetos

Para la fabricación de objetos se ha creado un menú en la parte izquierda de la pantalla.



FIGURA 173. MENÚ DE FABRICACIÓN

El *script* del inventario es el que se encarga de actualizar el menú de fabricación, ya que todos aquellos ítems fabricables aparecerán con sus respectivos *sprites* tal cual, mientras que aquellos ítems que todavía no sean fabricables aparecerán con los *sprites* con transparencia. Por otro lado, debajo de cada uno de los *sprites* aparecen 6 botones, cada uno de un color, siendo estos los 6 colores que se utilizan en el juego. En función

del número de recursos de un determinado color y material se tenga en el inventario, los botones del menú de fabricación se iluminarán indicando que ese objeto de ese color ya es fabricable. En el momento en el que se presione uno de los botones de colores, se realiza un pequeño cálculo en el *script* del inventario, donde se resta la cantidad del recurso necesaria a la cantidad que el jugador tenga en su inventario. Cabe destacar que, para poder hacer clic sobre los botones mencionados, son necesarios dos *Gameobject*, **EventSystem** y **EventManager**.

El primero posee los siguientes *scripts*, necesarios para que los jugadores puedan interactuar con los botones del menú de fabricación.

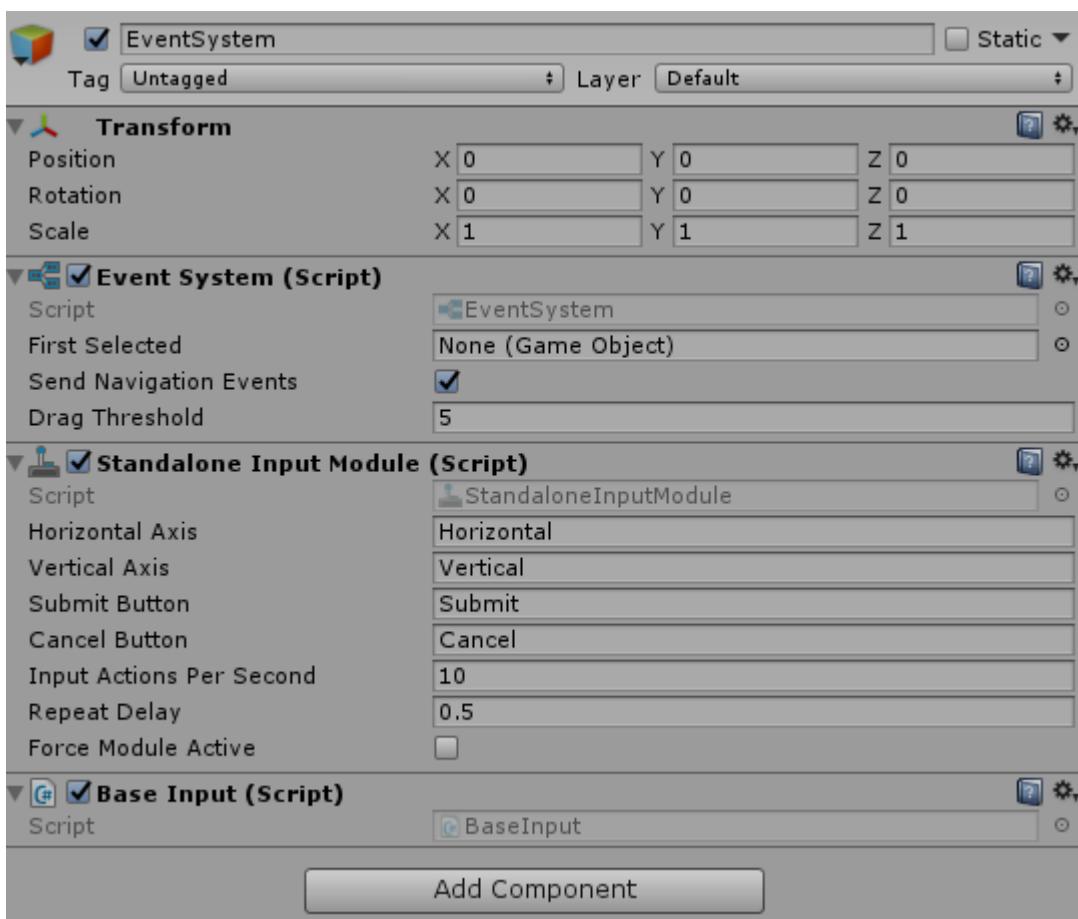


FIGURA 174. GAMEOBJECT EVENTSYSTEM

Por otro lado, **EventManager** se encarga de hacer posible la conexión entre el inventario y el menú de fabricación.

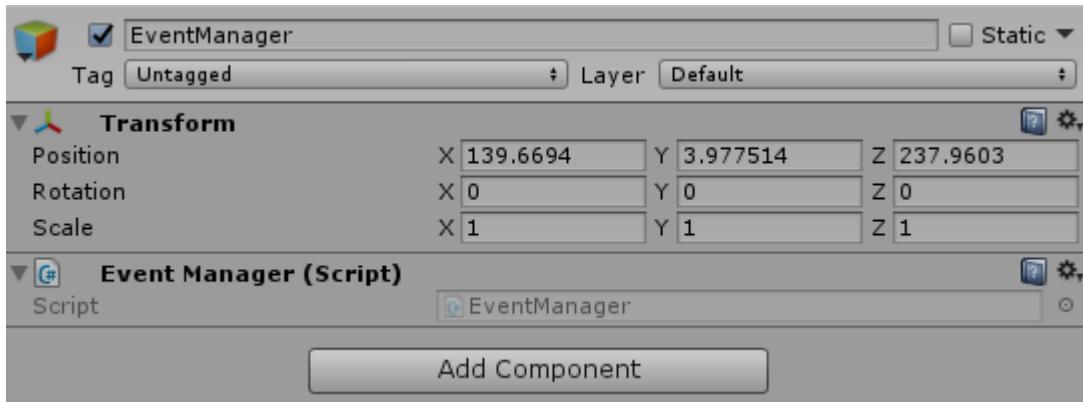


FIGURA 175. GAMEOBJECT EVENTMANAGER

El **script EventManager** contiene una función que hace uso de un *delegate*, un puntero de referencia a un método, que sirve para encadenar eventos. Básicamente, al hacer clic sobre uno de los botones de colores, se llama a un *trigger* de **EventManager** el cuál, gracias al *delegate*, desencadenará un evento en el inventario, donde se produce la creación del objeto y el cálculo de cantidades de recursos.

Mencionar que cada una de las casillas del menú de crafteo contiene un *script* con los siguientes parámetros. Importante el parámetro *Item* con el cuál se podrá saber la cantidad de recursos necesaria y qué tipo de recursos son necesarios para la fabricación de cada uno de los objetos.

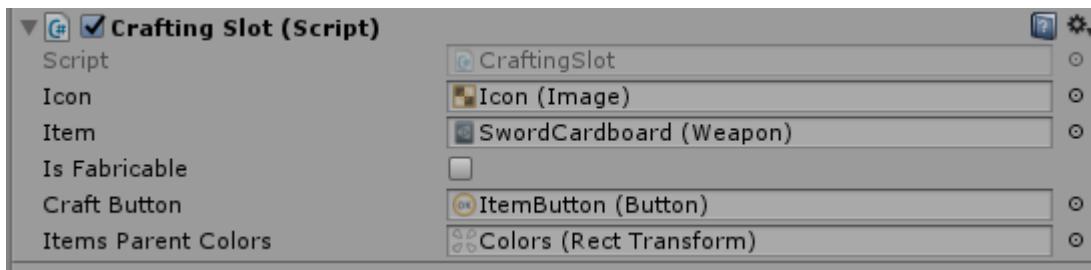


FIGURA 176. SCRIPT CRAFTINGSLOT

5.3.6 Eventos del mundo

5.3.6.1 Ciclo día-noche

El ciclo de día-noche viene configurado con la ayuda de un *prefab* que recibe el nombre de **GameManager**, en el cuál he creado un *script*, **DayNightCycle**.

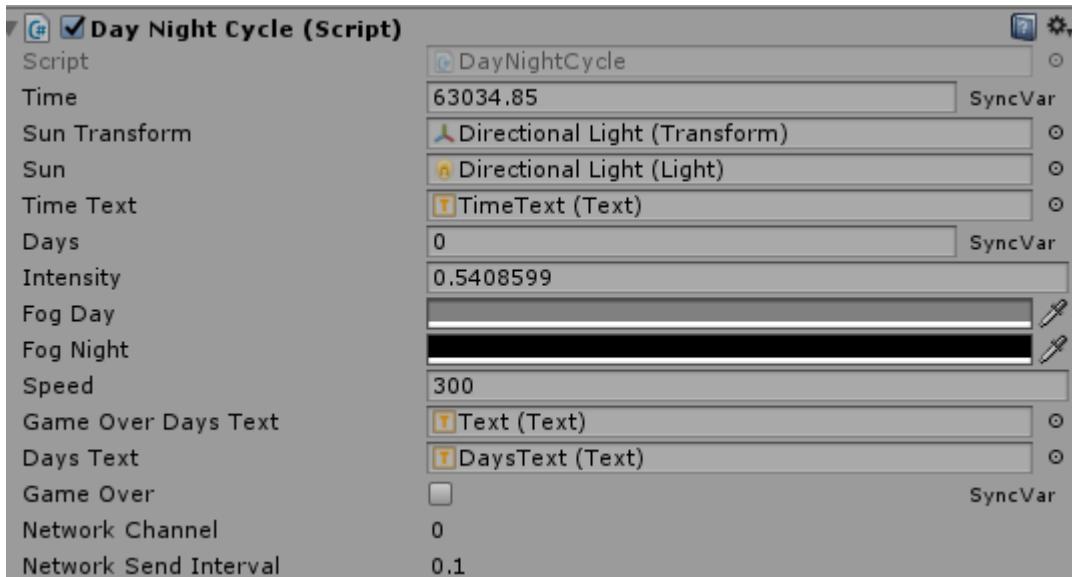


FIGURA 177. SCRIPT DAYNIGHTCYCLE

En la programación de este *script* es importante destacar la sincronización de la variable *time*, de tal manera que todos los jugadores obtengan la misma hora del día y, de esta manera, el resto de funciones del *script* quedarían automáticamente sincronizadas, ya que todas depende totalmente de la variable *time*.

En este *script* el Sol está representado por una luz direccional que rota alrededor del mundo.

```
sunTransform.rotation = Quaternion.Euler(new Vector3((time - 21600) / 86400 * 360, 0, 0));
```

FIGURA 178. CÓDIGO SOBRE LA ROTACIÓN DEL SOL

De esta forma, la iluminación del Sol no será la misma a las 12:00 del mediodía que durante el atardecer, escondiéndose en el horizonte. Esta iluminación no era suficiente para el efecto que yo quería lograr, ya que aun siendo de noche había una luz tenue, y mi idea es que durante la noche no se pudiese ver nada. Para controlar mejor la iluminación, la idea fue modificar la iluminación ambiental en función de la hora del día, cambiando la luz ambiental de forma suave e interpolada entre las horas del día.

Finalmente, cada vez que termine el día (a partir de las 00:00) se sumará al contador de *Days* un día más, siendo importante también sincronizar este contador en todos los clientes. El día y la hora actual se muestran en la parte superior derecha de la pantalla.



FIGURA 179. DÍAS TRANSCURRIDOS Y HORA DENTRO DEL JUEGO

Cuando un jugador muere, la variable *time* se detiene, y se mostrará en la pantalla de *GameOver* los días que han sobrevivido.



FIGURA 180. VENTANA DE GAMEOVER

5.3.6.2 Lluvia

Para la implementación de la lluvia, hacer un sistema de partículas que cubriese la isla en su totalidad rompía el juego debido a la enorme cantidad de partículas simultáneas, de tal manera que se optó por crear un *GameObject* que se ha enlazado a la jerarquía de objetos del *prefab* del jugador, el cual contiene un sistema de partículas y sigue al jugador a todas partes.

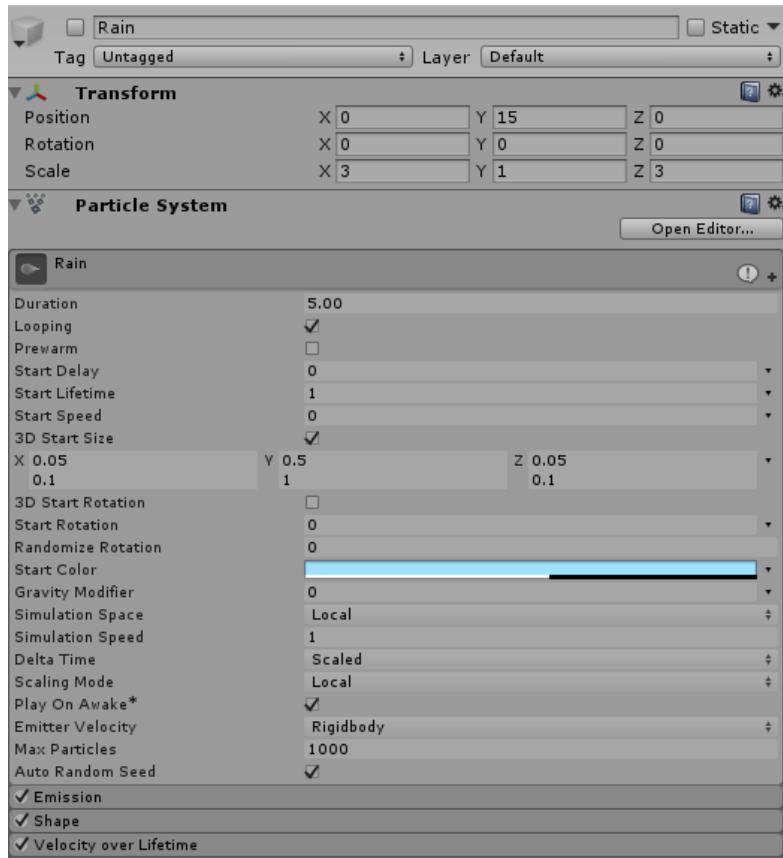


FIGURA 181. GAMEOBJECT RAIN, CON SISTEMA DE PARTÍCULAS

En este sistema de partículas se han definido múltiples parámetros, como puede ser la velocidad, el color y las formas de las gotas de lluvia, o incluso conseguir el efecto de que las gotas de lluvia salpiquen al colisionar con el terreno.



FIGURA 182. EJEMPLO DE LLUVIA DENTRO DEL JUEGO

Solamente podrá llover una vez al día. Esto está establecido en el *script* del ciclo de día-noche, en el cual, al inicio de cada día (y al inicio de la partida, las cuales comienzan a las 09:00) se calcula de manera aleatoria si va a llover ese día (un 50% de probabilidades siempre), y en caso afirmativo, se calcula la hora de inicio, de forma aleatoria, y el número de horas que durará la lluvia, que podrá ser entre 1 y 3 horas.

5.3.7 Cámara

La cámara implementada es en tercera persona. Se trata de un *GameObject* enlazado a los objetos de la jerarquía del *prefab* del jugador. Hay que tener en cuenta que, cuando se juega con varios jugadores, es importante tener de manera local únicamente activada la cámara correspondiente a tu jugador. El *script* encargado de la configuración de la cámara es el siguiente.

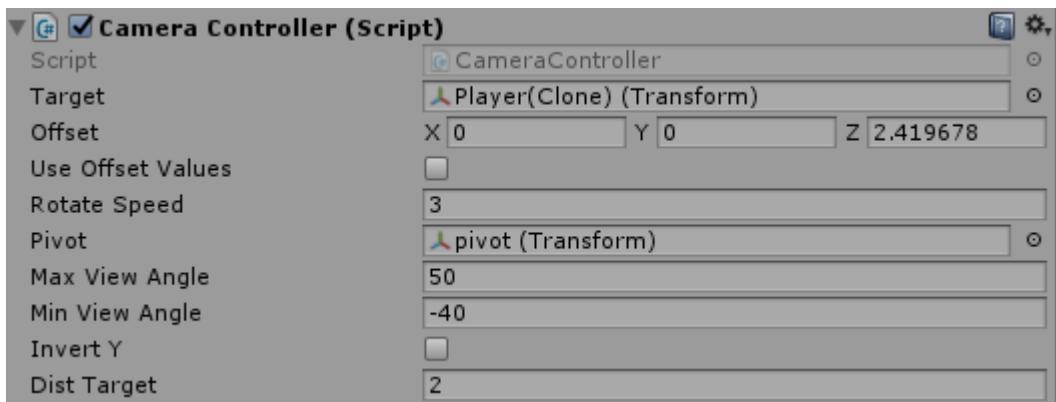


FIGURA 183. SCRIPT CAMERACONTROLLER

En él se establecen parámetros como el objetivo (siempre será el jugador), la distancia entre el jugador y la cámara, invertir el eje Y, o hasta límites de ángulos para no sobrepasar la cámara ni por arriba ni por debajo del jugador.

Por otro lado, la propia cámara cuenta también con un *script* encargado de añadir efectos de postprocesado.

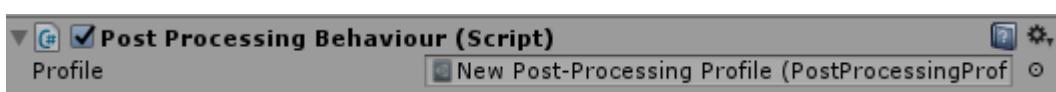


FIGURA 184. SCRIPT POSTPROCESSINGBEHAVIOUR

Este *script* incluye un *Asset* con una serie de parámetros totalmente configurables, tales como el *antialiasing*, la profundidad de campo, *Bloom*, etc.

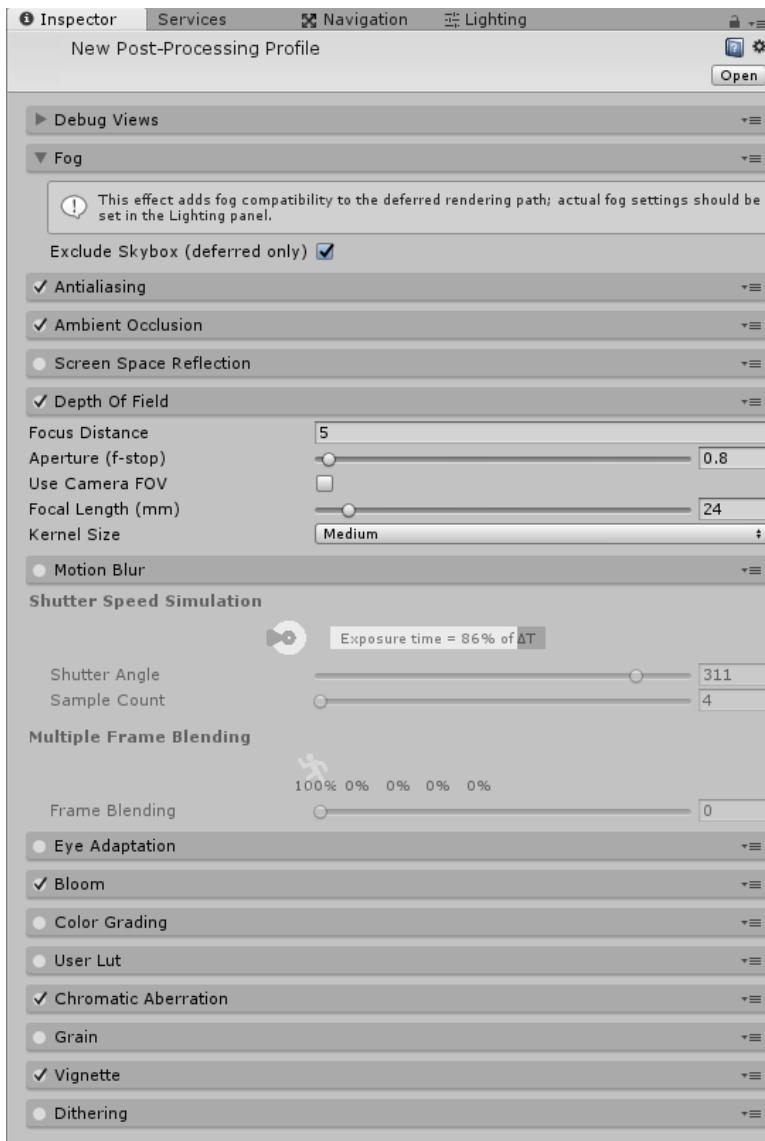


FIGURA 185. CONFIGURACIÓN DE LOS PARÁMETROS DEL ASSET DE POSTPROCESADO

5.3.8 Sonido

Para la implementación del sonido se ha configurado un *GameObject* con el nombre de **AudioManager**.

Este *GameObject* posee un *script* encargado de gestionar todos y cada uno de los sonidos (efectos de sonido y música) que se ejecutan durante la partida.

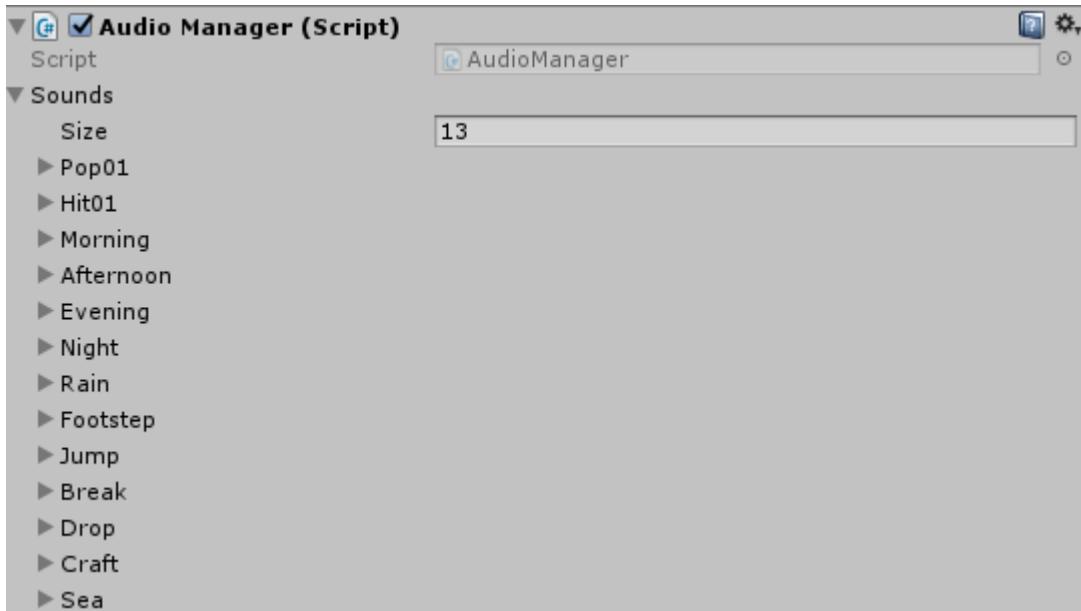


FIGURA 186. SCRIPT AUDIOMANAGER

He implementado un total de 13 sonidos, aunque en algunos de ellos he reutilizado el archivo de audio, cambiando algunos de sus parámetros de tal manera que suenen de forma diferente.

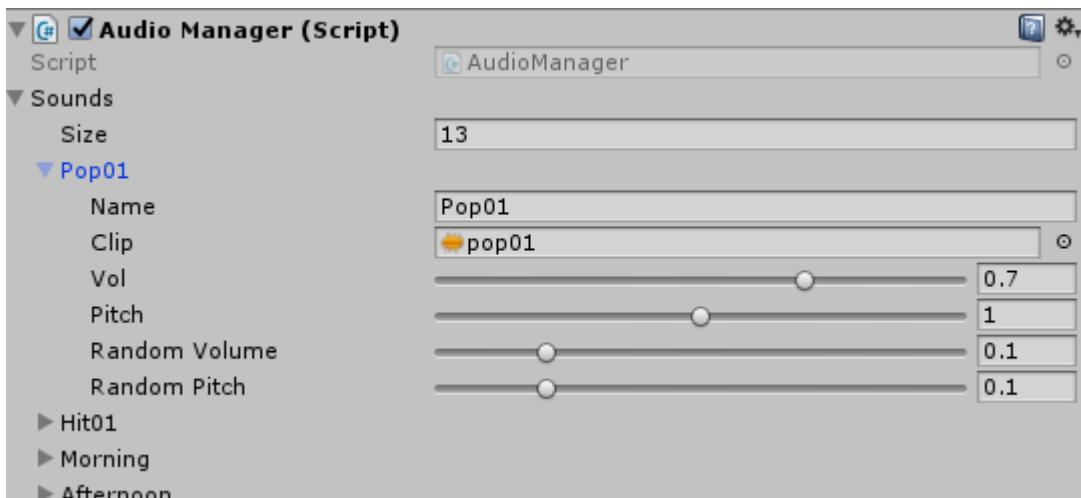


FIGURA 187. PARÁMETROS DE UN SONIDO

Entre los distintos parámetros que he establecido para configurar cada uno de los sonidos, cabe destacar ***RandomVolume*** y ***RandomPitch***. Mediante el *script*, la idea es *randomizar* estos dos parámetros, los cuales influyen directamente sobre ***Vol*** y ***Pitch***, aportando valores aleatorios a estos dos parámetros cada vez que se ejecute el sonido, cambiando el volumen y el tono en cada ejecución, haciendo que, de esta manera, el

sonido suene de forma distinta. Esto es sobre todo para algunos efectos de sonido, como el *Pop* que suena a la hora de obtener un objeto y añadirlo al inventario. En el caso de los sonidos que se corresponden con la música, los parámetros ***RandomVolume*** y ***RandomPitch*** deben estar a 0, ya que éstas sonarán siempre con el mismo tono y volumen.

Por otro lado, el *script* está configurado para que, durante la ejecución del videojuego, se genere un *GameObject* por cada sonido, y cada uno de estos tendrá una componente *AudioSource*, donde aparecen todos los parámetros de un sonido.

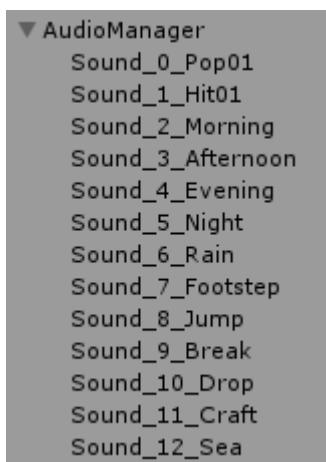


FIGURA 188. GAMEOBJECT DE LOS SONIDOS DURANTE LA EJECUCIÓN

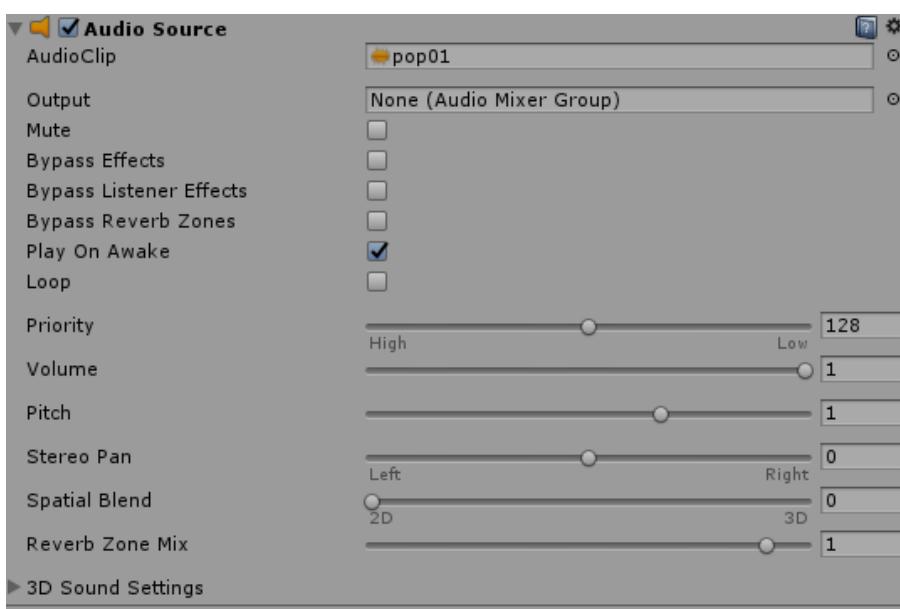


FIGURA 189. COMPONENTE AUDIOSOURCE

También, mencionar la importancia del parámetro *Loop* en la componente de la imagen anterior. Este parámetro permite reproducir un sonido en bucle, así pues, me ha sido de utilidad para reproducir en bucle el sonido de las olas del mar, o las pisadas de los jugadores mientras caminan, estableciendo este parámetro a *true* mediante el *script* mencionado anteriormente.

Finalmente, en cuanto a la música del juego he establecido 3 melodías diferentes, sonando una melodía alegre durante las 06:00 y las 12:00, una melodía algo más lenta sobre las 12:00 y las 18:00, y otra melodía bastante más lenta y relajada sobre las 18:00 y las 23:00. Entre las 00:00 y las 06:00 se escucharán simplemente sonidos ambientales, como grillos. Todas estas melodías vendrán acompañadas del sonido de las olas del mar, reproduciéndose en bucle. Además, cabe destacar que, en el momento de detener la reproducción de una melodía, se ha implementado un algoritmo en el *script* que se encarga de simular un efecto de *fade out*, es decir, un efecto de desvanecimiento progresivo del volumen de la melodía hasta que deja de sonar. Para conseguir este efecto se ha hecho uso de una *co-rutina*, la cual se encarga de disminuir el volumen del sonido de forma progresiva.

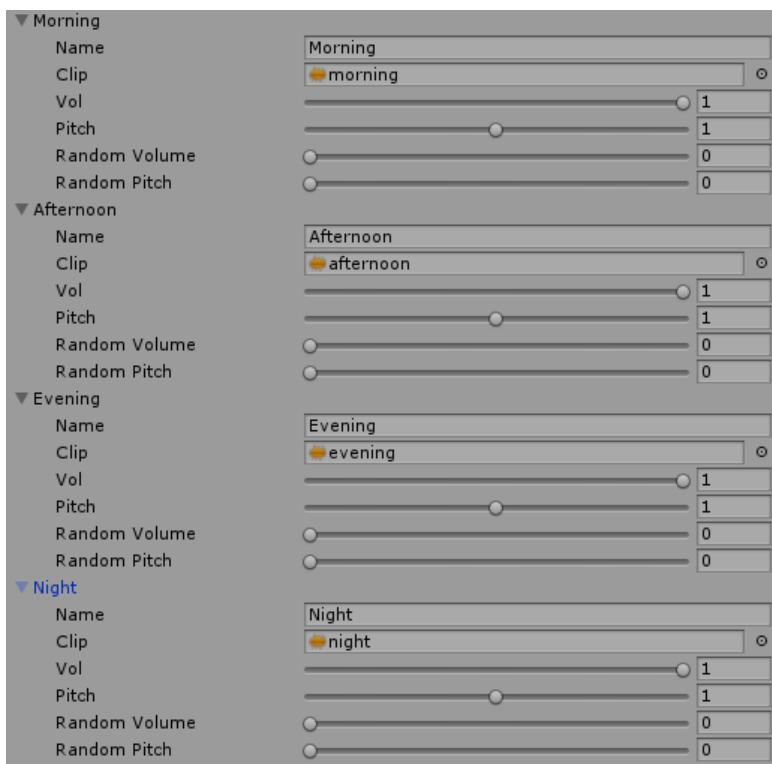


FIGURA 190. PARÁMETROS DE LAS MELODÍAS DEL JUEGO, Y LOS SONIDOS AMBIENTALES NOCTURNOS

5.3.9 Logo del videojuego

El logo ha sido creado mediante Photoshop, simulando el arte realizado con cartón.



FIGURA 191. EJEMPLO DE RETRATO HECHO CON CARTÓN



FIGURA 192. IMAGEN DEL PROYECTO DE PHOTOSHOP DEL LOGO

6. Conclusiones

Antes de comenzar con este proyecto, no tenía conocimientos sobre cómo crear este tipo de videojuegos online. La elaboración del TFG me ha resultado de gran utilidad para aprender. Destacar que, durante el proyecto, se han puesto en práctica gran cantidad de conocimientos adquiridos durante la carrera; conocimientos tanto de acceso a servicio web para la plataforma de juego online como en el campo del entretenimiento digital.

Por otra parte, debo comentar que debido al reducido ancho de banda que me brinda Unity de forma gratuita, he tenido problemas de desconexión con los clientes, ya que estos se desconectaban siempre pasado un tiempo de juego al superar el límite de ancho de banda. Conforme he ido implementando más y más cosas, el tiempo de juego antes de la desconexión se ha ido reduciendo, de tal forma que al inicio del proyecto y estando dos jugadores activos, el segundo jugador se desconectaba a los 8 minutos de partida, pero con toda la implementación actual, apenas llega a durar 10 segundos antes de la desconexión, siendo imposible actualmente poder jugar con más de 1 jugador. La cantidad de animales, vegetación, el mar, y la enorme extensión del mapa influyen en esto; más concretamente, todas aquellas variables que se sincronicen (las llamadas *SyncVar*) consumen ancho de banda.

A pesar de esto, en general me encuentro bastante satisfecho con el resultado de este proyecto. Es evidente que, al haberse realizado de manera individual, el producto ha terminado sin llegar a alcanzar la calidad con la que muchos videojuegos cuentan en el mercado profesional.

También, he de decir que, conforme avanzaba en el proyecto, el simple hecho de ir aprendiendo y practicando más la implementación de los diferentes apartados del videojuego, me han hecho darme cuenta de muchas cosas que, si tuviese que repetirlas, las haría de otra forma y de manera mucho más eficiente, siendo esto señal de que ha habido un claro aprendizaje durante todo el trabajo.

Muchas de las ideas que tenía inicialmente a la hora de comenzar a realizar el proyecto se han quedado en el tintero, pero eso me ha ayudado a darme cuenta de mis carencias y todo lo que aún me queda por aprender.

El siguiente listado muestra algunos de los aspectos a mejorar de este videojuego:

- Implementación de biomas.
- Algunos animales flotan, literalmente. Esto es debido a que la **navmesh** no se adapta por completo a la maya del terreno. Esto es debido a que se crea en tiempo de ejecución.
- Mientras que un jugador corre, y golpea, se ejecuta la animación de golpear, en la cual no mueve los pies; sin embargo, el personaje está desplazándose porque está corriendo. Esto se podría mejorar implementando **Blend Trees** o árboles de mezcla, permitiendo poder mezclar animaciones (por ejemplo, que puedan correr con las piernas mientras mueven los brazos para golpear).
- Añadir más objetos con diferentes utilidades, como por ejemplo un parapente.
- Un sistema de construcción de estructuras mediante cubos o planos.
- Añadir más sonidos.
- Añadir efectos visuales, por ejemplo, al golpear a un animal que se genere algún sistema de partículas.

Finalmente, he de decir que he cumplido con todos los **objetivos principales** que se marcaron al inicio del proyecto:

- **Aprender a utilizar Unity:** Mis conocimientos sobre este motor gráfico se han ampliado en gran parte.
- **Estudiar el algoritmo de generación procedural de terrenos:** El proyecto me ha ayudado a entender uno de los algoritmos de generación procedural más utilizados tanto en el cine como en el campo de los videojuegos.
- **Estudiar y aplicar técnicas de modelado, texturizado y materiales:** El aspecto visual de los modelos 3D dentro del juego es bastante satisfactorio,

mediante el uso de mapas de normales, texturizar los modelos con colores simples (pero con degradados), pequeñas modificaciones en los parámetros de los materiales en Unity, etc.

- **Estudiar sistemas de red multijugador:** Al inicio, esta parte me supuso un gran obstáculo en el avance del proyecto, pero con el paso del tiempo fui aprendiendo a dominarlo.
- **Estudiar plataformas digitales con servicio social de jugadores:** Al igual que el punto anterior, fue otro de los grandes obstáculos, en parte debido a que la documentación de la API de Steam es, en mi opinión, compleja de entender.

7. Bibliografía y referencias

Definición de videojuego según Wikipedia:

<https://es.wikipedia.org/wiki/Videojuego>

Imagen de Copia de Tennis for Two:

https://es.wikipedia.org/wiki/Videojuego#/media/File:Tennis_for_Two_Machine_at_CAX_2010.jpg

Imagen de Computer Space:

https://commons.wikimedia.org/wiki/File:Nutting_ComputerSpace-Blue.JPG

Imagen de Pong:

<https://commons.wikimedia.org/wiki/File:Pong.png>

Géneros de videojuegos según Wikipedia:

https://es.wikipedia.org/wiki/G%C3%A9nero_de_videojuegos

Reportaje sobre videojuegos de supervivencia en Vandal:

<https://vandal.elespanol.com/reportaje/juegos-de-supervivencia-la-revolucion-independiente>

Imagen de Dayz:

https://media.vandal.net/i/1200x630/16507/dayz-2013121785254_1.jpg

Imagen de Minecraft:

https://img00.deviantart.net/4686/i/2012/085/8/f/minecraft_survival_mode_my_use_by_nero_darkpulse-d4tzg1k.png

Imagen de Don't Starve:

<https://xombitgames.com/files/2014/08/Dont-Starve.jpg>

Imagen de Rust:

<https://i11a.3djuegos.com/juegos/10077/rust/fotos/set/rust-2467569.jpg>

Imagen de ARK: Survival Evolved:

https://vignette.wikia.nocookie.net/ark-survival-evolved/images/6/61/ARK-Ankylosaurus_Screenshot_003.jpg/revision/latest?cb=20150909012137

Reportaje sobre ARK: Survival Evolved en 3djuegos:

<https://www.3djuegos.com/juegos/avances/22285/4505/0/ark-survival-evolved/>

Imagen de Terraria:

<https://cdn3.dualshockers.com/wp-content/uploads/2017/12/Terraria-Nintendo-Switch.jpg>

Imagen de The Legend of Zelda: Breath of the Wild:

<https://venturebeat.com/wp-content/uploads/2017/02/Zelda-breath-of-the-wild-10.jpg?resize=1280%2C720&strip=all>

Definición de motor gráfico según Wikipedia:

https://es.wikipedia.org/wiki/Motor_de_videojuego

Reportaje sobre motores gráficos por parte de VidaExtra:

<https://www.vidaextra.com/listas/si-quieres-hacer-tus-propios-juegos-estos-son-los-mejores-motores-que-vas-a-encontrar>

Logo de Unity:

<http://www.freelogovectors.net/wp-content/uploads/2013/06/unity3d-logo.jpg>

Logo de Unreal Engine:

<https://steamuserimages.akamaihd.net/ugc/767148481029971829/D531C176558ACA905307D3A3F477EB3218E865B9/>

Información sobre Cocos2d:

<https://software.intel.com/es-es/articles/creating-multi-platform-games-with-cocos2d-x>

Logo de Cocos2d:

https://www1-lw.xda-cdn.com/files/2014/10/cocos2dx_portrait.png

Información sobre UNET:

<https://docs.unity3d.com/es/current/Manual/UNetOverview.html>

Imagen metodología Kanban:

<https://static.kanbantool.com/seo-landing-page/kanban-method/a-simple-kanban-board.png>

Información sobre la metodología Kanban:

<https://kanbantool.com/es/metodologia-kanban>

Información sobre la metodología Kanban según Wikipedia:

[https://es.wikipedia.org/wiki/Kanban_\(desarrollo\)](https://es.wikipedia.org/wiki/Kanban_(desarrollo))

Información sobre Trello:

<https://es.wikipedia.org/wiki/Trello>

Logo de Trello:

<https://upload.wikimedia.org/wikipedia/commons/thumb/7/7a/Trello-logo-blue.svg/150px-Trello-logo-blue.svg.png>

Imagen de ejemplo de proyecto en Trello:

<https://d2k1ftgv7pobq7.cloudfront.net/meta/p/res/images/bb311e4d3417ac027a5e4545146389e7/usecases-board02.jpg>

Logo de Github:

<https://19386-presscdn-pagely.netdna-ssl.com/wp-content/uploads/2017/10/github-logo-1.png>

Información sobre Unity según Wikipedia:

[https://es.wikipedia.org/wiki/Unity_\(motor_de_juego\)](https://es.wikipedia.org/wiki/Unity_(motor_de_juego))

Información sobre Modelado 3D según Wikipedia:

https://es.wikipedia.org/wiki/Modelado_3D

Logo de Autodesk Maya:

<https://howtolearn.me/wp-content/uploads/2014/04/Autodesk-Mayan-logo.png>

Logo de Autodesk 3ds max:

<https://www.nke360.es/wp-content/uploads/3ds-max-icon-400px-social.png>

Información sobre Autodesk 3ds max según Wikipedia:

https://es.wikipedia.org/wiki/Autodesk_3ds_Max

Información sobre Autodesk Maya según Wikipedia:

https://es.wikipedia.org/wiki/Autodesk_Maya

Logo de Blender:

<https://download.blender.org/institute/BlenderDesktopLogo.png>

Información sobre Blender según Wikipedia:

<https://es.wikipedia.org/wiki/Blender>

Texturizar en gráficos por computadora según Wikipedia:

[https://es.wikipedia.org/wiki/Textura_\(gr%C3%A1ficos_por_computadora\)](https://es.wikipedia.org/wiki/Textura_(gr%C3%A1ficos_por_computadora))

Imagen de texturizado:

<https://upload.wikimedia.org/wikipedia/commons/thumb/3/30/Texturedm1a2.png/220px-Texturedm1a2.png>

Imagen de proyecto de ejemplo de Unity3D:

https://academiaandroid.com/imagenes_cursos/unity3D/VistaJuego.png

Logo de Autodesk Mudbox:

http://www.xfx.co.uk/s/cc_images/teaserbox_14286523.png?t=1485349912

Imagen ejemplo de Modelo 3D:

<https://www.softzone.es/app/uploads/2017/01/Dino-Modelado-3D.png?x=634&y=309>

Logo de GIMP:

https://upload.wikimedia.org/wikipedia/commons/thumb/4/45/The_GIMP_icon_-_gnome.svg/64px-The_GIMP_icon_-_gnome.svg.png

Información sobre GIMP según Wikipedia:

<https://es.wikipedia.org/wiki/GIMP>

Logo de Adobe Photoshop:

https://upload.wikimedia.org/wikipedia/commons/thumb/2/20/Photoshop_CC_icon.png/200px-Photoshop_CC_icon.png

Información sobre Adobe Photoshop según Wikipedia:

https://es.wikipedia.org/wiki/Adobe_Photoshop

Imagen de un ejemplo de proyecto de Photoshop:

<https://i.ytimg.com/vi/X9QEwlSesLQ/maxresdefault.jpg>

Logo de PaintTool SAI:

https://upload.wikimedia.org/wikipedia/en/f/f2/Paint_Tool_SAI_Logo.png

Información de PaintTool SAI según Wikipedia:

[https://es.wikipedia.org/wiki/SAI_\(software\)](https://es.wikipedia.org/wiki/SAI_(software))

Imagen de ejemplo de PaintTool SAI:

https://2.bp.blogspot.com/-ZCdN_MBI-Nk/WUhHw5JPb4I/AAAAAAA3I/55EE6Iav4t0_mTPxTkTVrsazbKexcdyIgCLcB_GAs/s1600/1.jpg

Información sobre Rigging según Wikipedia:

<https://es.wikipedia.org/wiki/Rigging>

Imagen de Rigging:

<https://img2.cgtrader.com/items/152355/f0dd5a6924/gorilla-maya-3d-model-rigged-ma-mb.jpg>

Información sobre Skinning:

<http://www.ite.educacion.es/formacion/materiales/181/cd/m10/skinning.html>

Imagen de ejemplo de Skinning:

https://blogs.unity3d.com/wp-content/uploads/2014/04/Character2Player_0001_Step_002.jpg

Imagen de ejemplo de Animación 3D:

<https://i.ytimg.com/vi/ax73HMCnGWI/maxresdefault.jpg>

Imagen Cliente-Servidor en UNET:

<https://docs.unity3d.com/es/current/uploads/Main/NetworkHost.png>

Información sobre UNET:

<https://docs.unity3d.com/es/current/Manual/UNetConcepts.html>

Logo de PUN:

<https://d2my8q2pm2pgs9.cloudfront.net/uploads/images/introduction-into-photon-unity-networking/header.png>

Información sobre PUN:

<http://200.35.84.131/portal/bases/marc/texto/2501-15-09037.pdf>

Información sobre uLink:

<http://developer.muchdifferent.com/unitypark/uLink/uLink>

Logo de Steam:

https://upload.wikimedia.org/wikipedia/commons/thumb/8/83/Steam_icon_logo.svg/100px-Steam_icon_logo.svg.png

Información sobre Steamworks:

<https://partner.steamgames.com/>

Logo de Steamworks:

https://partner.steamgames.com/public/images/home/bluelogo_steamworks.png

Información sobre Steam:

<https://es.wikipedia.org/wiki/Steam>

Logo de Origin:

<https://upload.wikimedia.org/wikipedia/commons/thumb/f/f2/Origin.svg/2000px-Origin.svg.png>

Información sobre Origin:

[https://es.wikipedia.org/wiki/Origin_\(software\)](https://es.wikipedia.org/wiki/Origin_(software))

Logo de Google Play:

https://upload.wikimedia.org/wikipedia/commons/e/ee/Google_Play_logo.png

Información sobre Google Play:

https://es.wikipedia.org/wiki/Google_Play#Play_Juegos

Información sobre la API de Google Play Games:

<https://developers.google.com/games/services/>

Logo de Google Play Games:

https://cdn.dribbble.com/users/621186/screenshots/2737810/tmp_26057-google_play_game_icon-1458341649_1x.png

Logo de uPlay:

<https://itscdn.com/resources/services/logowide/340/uplay-ubisoft.png>

Información sobre uPlay:

<https://es.wikipedia.org/wiki/Uplay>

Logo de clasificación PEGI:

https://upload.wikimedia.org/wikipedia/commons/thumb/6/6c/PEGI_-_Logo.svg/245px-PEGI_-_Logo.svg.png

Información sobre clasificación PEGI:

https://es.wikipedia.org/wiki/Pan_European_Game_Information

Logo de PEGI 3:

https://upload.wikimedia.org/wikipedia/commons/thumb/2/2c/PEGI_3.svg/60px-PEGI_3.svg.png

Logo de PEGI Online:

https://upload.wikimedia.org/wikipedia/commons/thumb/2/2c/PEGI_Online.png/60px-PEGI_Online.png

Imagen de Paper Mario: Color Splash: https://cdn02.nintendo-europe.com/media/images/06_screenshots/games_5/wiiu_7/wiiu_papermariocolorsplash/WiiU_PaperMarioColorSplash_07.jpg

Tutorial Perlin Noise (1):

https://www.youtube.com/watch?v=V0dwkrS6CoI&ab_channel=SystemCult

Tutorial Perlin Noise (2):

https://www.youtube.com/watch?v=vFvwyu_ZKfU

Texturizado automático del terreno:

https://www.youtube.com/playlist?list=PLoSHfhkVdu88Syb_CJFI78PEGMJ9R8WxA

Documentación sobre Perlin Noise:

<https://shanee.io/blog/2015/09/25/procedural-island-generation/>

Foro de Unity:

<https://forum.unity.com>

Steamworks en Unity:

https://www.youtube.com/watch?v=YzeGHTwS4gE&t=1100s&ab_channel=MikeBushell

Documentación de Steamworks:

<https://partner.steamgames.com/doc/sdk>

Documentación de Unity:

<https://docs.unity3d.com/Manual/index.html>

Tutorial ciclo día-noche:

https://www.youtube.com/watch?v=y6TCQfFB2xg&ab_channel=Gamad

Tutorial lluvia:

https://www.youtube.com/watch?v=cDHJuRV5Ays&ab_channel=Mirza

Tutorial animaciones Unity:

https://www.youtube.com/watch?v=BEIaakl9vJE&ab_channel=Holistic3d

Tutorial cámara en tercera persona:

https://www.youtube.com/watch?v=bJ6pbT3RzLA&ab_channel=gamesplusjames

Tutorial NetworkLobby:

https://www.youtube.com/watch?v=jklWlm5v21k&t=58s&ab_channel=Holistic3d

Tutorial AudioManager:

https://www.youtube.com/watch?v=HhFKtiRd0qI&ab_channel=Brackeys

Música y efectos de sonido:

https://www.youtube.com/watch?v=8Mk2WoUpmGM&ab_channel=Derek%26BrandonFiechter

https://www.youtube.com/watch?v=x8NpJ4myReU&ab_channel=Derek%26BrandonFiechter

https://www.youtube.com/watch?v=617AatEi-WI&ab_channel=TravisD

https://www.youtube.com/watch?v=paXZ1fjgpdc&ab_channel=AmbianceMagic

https://www.youtube.com/watch?v=NH-4HVCNUoI&t=182s&ab_channel=Clancer

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

https://www.youtube.com/watch?v=EVd1yBmyUXU&ab_channel=Adrien33

https://www.youtube.com/watch?v=8wxopUbQrSY&ab_channel=FastSolution