

2012-08-29

# Manual de creación de videojuego con Unity 3D

Ouazzani, Iman

---

<http://hdl.handle.net/10016/16345>

---

*Descargado de e-Archivo, repositorio institucional de la Universidad Carlos III de Madrid*

UNIVERSIDAD CARLOS III



# MANUAL DE CREACIÓN DE VIDEOJUEGO CON UNITY 3D

---



29/08/2012

**AUTOR:** IMAN OUAZZANI

**TUTOR:** JUAN PERALTA DONATE



*A todos los míos,*





## **AGRADECIMIENTOS:**

La realización de este proyecto representa la última etapa en mis estudios de ingeniería informática por lo cual quisiera aprovechar la oportunidad que se me brinda para agradecer todo el apoyo que se me ha dado durante todos estos años.

En primer lugar agradecer a mis padres Patrick y Aicha toda su paciencia, su apoyo moral y material y la oportunidad que me han brindado de realizar mis estudios en el extranjero con todo lo que ello conlleva, una carrera profesional, una nueva cultura y un nuevo idioma y un largo etc.

A mi novio, Julián por estar a mi lado todos estos años, por ser mi compañero de prácticas, aguantarme y apoyarme en los momentos difíciles.

A mi tutor Juan Peralta por haberme dado la oportunidad de realizar este proyecto, por sus consejos y por el tiempo que me ha dedicado.

En general, a todos aquellos profesores y compañeros que de una u otra forma me han ayudado a lo largo de estos años, que aunque no les mencione de forma explícita, no les puedo negar un sincero agradecimiento.



## **CONTENIDO**

1. INTRODUCCIÓN:	10
1.1. Estructura del documento:	11
1.2. Motivación:	12
1.3. Objetivos del proyecto:	12
2. ESTADO DEL ARTE:	14
2.1. Motor de videojuegos:	14
2.2. Historia de los videojuegos:	16
2.3. Tipos de motores:	17
2.3.1. Roll-your-own:	17
2.3.2. Mostly ready :	17
2.3.3. Point-and-click	18
2.4. Listado de motores:	18
2.4.1. Motores open source:	18
2.4.2. Motores freeware:	19
2.4.3. Motores propietario:	20
3. UNITY 3D:	21
4. HISTORIA DEL PACMAN:	26
5. PRIMER ACCESO AL MOTOR:	28
6. LA INTERFAZ DE USUARIO DE UNITY:	29
6.1. Menú de la aplicación:	30
6.2. Botones de control:	30
6.3. Vista de escena:	31
6.4. Vista del juego:	32
6.5. Vista de proyecto:	33
6.6. Vista de jerarquía:	34
6.7. Vista de inspector:	35
7. CREAR UN NUEVO PROYECTO:	36
7.1. Tutorial: Crear una nueva escena:	37
7.2. Tutorial: Creación de objetos:	38
7.3. Tutorial: Manipulación de objetos:	39
7.4. Tutorial: Añadir luz a la escena:	41
7.5. Tutorial: Añadir textura:	43
7.6. Tutorial: Configurar la textura:	45
8. EDITOR DE TERRENOS:	47



---

8.1.	Importación de los heightmaps:.....	47
8.2.	Tutorial: Creación de un terreno.....	47
8.3.	Tutorial: La geometría del terreno.....	50
9.	CONCEPTO DE LIGHTMAP:.....	56
10.	EL ENTORNO DEL TERRENO.....	59
10.1.	Tutorial: Añadir agua a la escena: .....	59
10.2.	Tutorial: Añadir un cielo: .....	61
10.3.	Tutorial: Un FPC .....	63
10.4.	Tutorial: Crear un gusano .....	66
10.5.	Tutorial: La cámara sigue el personaje .....	68
10.6.	Tutorial: Movilidad del gusano.....	69
10.7.	Tutorial: Crear capsulas .....	70
10.8.	Tutorial: Sonido en la escena.....	71
10.9.	Tutorial: Un contador.....	73
10.10.	Tutorial: Encender/apagar la luz .....	74
11.	EL PRIMER SCRIPT: .....	76
11.1.	Tutorial: Script de rotación .....	77
12.	PREFAB:.....	79
12.1.	Tutorial: Crear un prefabricado .....	79
12.2.	Tutorial: Movimientos del personaje:.....	80
13.	LOS COLLIDERS: .....	83
13.1.	Tutorial: Objeto de colisión .....	84
13.2.	Tutorial: Recogida de las capsulas.....	85
14.	SISTEMA DE PARTÍCULAS:.....	87
14.1.	Tutorial: El fuego.....	87
14.2.	Tutorial: El Humo.....	91
15.	SHADERS:.....	94
15.1.	Tutorial: Agua transparente .....	94
16.	MATERIALES FÍSICOS:.....	98
16.1.	Tutorial: Rigidbody.....	98
16.2.	Tutorial: Propiedades de materiales.....	99
16.3.	Tutorial: Fixed Joint.....	101
16.4.	Tutorial: Hinge Joint.....	102
16.5.	Tutorial: Simulación de tela.....	103
17.	PERSONALIZAR LA INTERFAZ: .....	107
17.1.	Tutorial: Barra de desplazamiento:.....	107



---

17.2. Tutorial: Cronometro: .....	109
18. IMPORTACIÓN DE MODELOS: .....	111
18.1. Tutorial: Importar un personaje .....	112
19. INTELIGENCIA ARTIFICIAL:.....	114
19.1. Tutorial: Una bala .....	114
19.2. Tutorial: Un arma.....	115
19.3. Tutorial: Movimiento de los enemigos .....	117
20. PACMAN:.....	120
20.1. Tutorial: Poner varias cámaras en la escena: .....	120
20.2. Tutorial: Máquina de estados:.....	122
20.3. Tutorial: Puertas invisibles: .....	127
20.4. Tutorial: Introducir un botón de interfaz: .....	128
20.5. Tutorial: Cambio de propiedades en ejecución:.....	129
20.6. Tutorial: Pasar de una escena a otra: .....	130
20.7. Tutorial: Cerrar la aplicación:.....	131
21. COMPILACIÓN PARA WEBPLAYER:.....	132
22. GESTIÓN DEL PROYECTO.....	134
22.1. Diagramas de Gantt. ....	134
22.2. Planificación estimada. ....	134
22.3. Planificación real: .....	136
22.4. Análisis de la planificación:.....	138
23. PRESUPUESTO .....	139
24. COMERCIALIZACIÓN DEL PROYECTO:.....	144
25. CONCLUSIONES: .....	145
26. LÍNEAS FUTURAS: .....	146
27. REFERENCIAS:.....	147
28. ANEXOS:.....	150
28.1. Glosario de términos:.....	150
28.2. Guía del juego “Pacman”:.....	152



**INDICE DE TABLAS:**

Tabla 1: Motores open source .....	19
Tabla 2: Motores freeware.....	19
Tabla 3: Motores propietario .....	20
Tabla 4: Fantasmas del pacman.....	27
Tabla 5: Recursos humanos.....	139
Tabla 6: Recursos materiales.....	140
Tabla 7: Costes añadidos .....	142
Tabla 8: Presupuesto total del proyecto.....	143
Tabla 9: Glosario de términos .....	151





## INDICE DE IMÁGENES:

Ilustración 1: Tiger Woods .....	23
Ilustración 2: Naval War Arctic Circle.....	24
Ilustración 3: Bubble Bang.....	24
Ilustración 4: Max and the magic marker.....	24
Ilustración 5: Cabals, The card game .....	25
Ilustración 6: Escape Plan.....	25
Ilustración 7: versiones del pacman.....	26
Ilustración 8: Tropical paradise demo de unity .....	28
Ilustración 9: Interfaz de usuario .....	29
Ilustración 10: Menú de la aplicación.....	30
Ilustración 11: Botones de manipulación .....	30
Ilustración 12: Botones de control.....	31
Ilustración 13: vista de escena .....	31
Ilustración 14: Menú de la vista de escena .....	31
Ilustración 15: Render Mode .....	31
Ilustración 16: Modos de visualización .....	32
Ilustración 17: Vista del juego.....	33
Ilustración 18: Vista de proyecto.....	33
Ilustración 19: Menú de la vista de proyecto.....	34
Ilustración 20: Vista de jerarquía .....	34
Ilustración 21: Vista de inspector .....	35
Ilustración 22: Creación de un proyecto.....	36
Ilustración 23: Creación de una escena .....	37
Ilustración 24: Creación de un GameObject.....	38
Ilustración 25: GameObject cubo .....	39
Ilustración 26: manipulación de objetos 1 .....	39
Ilustración 27: Movimiento del objeto 1 .....	39
Ilustración 28: Opción de rotación .....	40
Ilustración 29: Rotación del objeto.....	40
Ilustración 30: manipulación de objeto 3 .....	40
Ilustración 31: Escala del objeto.....	40
Ilustración 32: Manipulación desde la vista del inspector.....	41
Ilustración 33: Directional light.....	41
Ilustración 34: Características de la luz .....	42
Ilustración 35: Texturización de un objeto. ....	43
Ilustración 36: Selección de la textura.....	44
Ilustración 37: Objeto con textura .....	44
Ilustración 38: Parámetros de textura .....	45
Ilustración 39: Reflexión de la luz.....	46
Ilustración 40: Efecto material .....	46



---

Ilustración 41: Creación de un terreno 1 .....	47
Ilustración 42: Características del terreno .....	48
Ilustración 43: Creación de terreno 2 .....	49
Ilustración 44: Creación de terreno 3 .....	49
Ilustración 45: Moldeado del terreno 1.....	50
Ilustración 46: Moldeado del terreno 2.....	51
Ilustración 47: Moldeado del terreno 3.....	51
Ilustración 48: Resultado del moldeado .....	52
Ilustración 49: Texturizar el terreno .....	52
Ilustración 50: Propiedades de la textura.....	53
Ilustración 51: Terreno texturizado .....	53
Ilustración 52: Añadir arboles al terreno 1. ....	54
Ilustración 53: Añadir arboles al terreno 2 .....	54
Ilustración 54: Añadir arboles al terreno 3 .....	55
Ilustración 55: Terreno completado.....	55
Ilustración 56: Lightmapping .....	56
Ilustración 57: Bake de lightmapping.....	57
Ilustración 58: Daylight simple water .....	59
Ilustración 59: Añadir agua a la escena .....	60
Ilustración 60: Características del agua .....	60
Ilustración 61: Añadir un skybox .....	61
Ilustración 62: Textura de cielo .....	62
Ilustración 63: Escena con skybox .....	62
Ilustración 64: First person controller 1 .....	63
Ilustración 65: First person controller 2 .....	64
Ilustración 66: Character Controller.....	64
Ilustración 67: Character motor.....	65
Ilustración 68: FPSInput controller .....	66
Ilustración 69: Creación de personaje sencillo .....	66
Ilustración 70: EL personaje gusano 1 .....	67
Ilustración 71: El personaje gusano 2.....	67
Ilustración 72: Smooth follow 1 .....	68
Ilustración 73: Smooth follow 2 .....	69
Ilustración 74: Movilidad del personaje .....	69
Ilustración 75: Características de la movilidad.....	70
Ilustración 76: Creación de capsulas .....	70
Ilustración 77: Importación de sonido.....	71
Ilustración 78: Características del sonido.....	71
Ilustración 79: Componente audio source .....	72
Ilustración 80: Configuración del audio source.....	73
Ilustración 81: Contador del score.....	73
Ilustración 82: Script de luz.....	74



Ilustración 83: Encendido y apagado de luz .....	75
Ilustración 84: Editor de script uniscite .....	76
Ilustración 85: Creación de un javascript 1 .....	77
Ilustración 86: Creación de un javascript 2 .....	77
Ilustración 87: Rotación de una capsula .....	78
Ilustración 88: velocidad de rotación .....	78
Ilustración 89: El prefab capsula .....	79
Ilustración 90: El prefab capsula .....	79
Ilustración 91: Utilización del prefab de capsulas .....	80
Ilustración 92: Remove Component .....	80
Ilustración 93: Script de movimiento .....	81
Ilustración 94: Mensaje de error de compilación .....	82
Ilustración 95: Mesh collider 1 .....	84
Ilustración 96: Mesh collider 2 .....	84
Ilustración 97: Inserción de sonido .....	85
Ilustración 98: Añadir el script de colisión .....	85
Ilustración 99: Efecto explosión de colores .....	86
Ilustración 100: Recogida de las capsulas .....	86
Ilustración 101: Particule system .....	87
Ilustración 102: Elipsoide particule emitter .....	88
Ilustración 103: Resultado del sistema de partículas .....	89
Ilustración 104: Particule Renderer .....	89
Ilustración 105: El sistema fuego .....	90
Ilustración 106: Particule Animator .....	90
Ilustración 107: Point light para el fuego .....	91
Ilustración 108: El fuego .....	91
Ilustración 109: Características del humo 1 .....	92
Ilustración 110: Características del humo 2 .....	92
Ilustración 111: El humo .....	93
Ilustración 112: Combinación fuego y humo .....	93
Ilustración 113: Los shaders .....	94
Ilustración 114: Terreno para lago .....	95
Ilustración 115: Aplicación del agua .....	95
Ilustración 116: Shaders .....	95
Ilustración 117: Agua transporte .....	96
Ilustración 118: WaterPlane .....	96
Ilustración 119: Texturas de agua .....	96
Ilustración 120: Texturas WaterPlane .....	97
Ilustración 121: Aplicar agua transparente .....	97
Ilustración 122: Aspecto final del lago .....	97
Ilustración 123: Rigidbody .....	98
Ilustración 124: Componente Rigidbody .....	98



Ilustración 125: Características del Rigidbody.....	99
Ilustración 126: Añadir material 1 .....	99
Ilustración 127: Materiales existentes .....	100
Ilustración 128: Propiedades de los materiales.....	100
Ilustración 129: Creación de un material.....	100
Ilustración 130: Fixed Joint 1 .....	101
Ilustración 131: Fixed Joint 2 .....	101
Ilustración 132: Elemento conectado con Fixed Joint.....	102
Ilustración 133: Hinge Joint .....	102
Ilustración 134: Movimiento del péndulo .....	103
Ilustración 135: Interactive Cloth.....	103
Ilustración 136: Características del Interactive Cloth.....	104
Ilustración 137: Gameobject Cloth.....	104
Ilustración 138: Caída de la alfombra.....	105
Ilustración 139: Inserción de colliders .....	105
Ilustración 140: Propiedades bandera.....	106
Ilustración 141: Movimiento de la bandera .....	106
Ilustración 142: Cubo rotando .....	108
Ilustración 143: Barra de desplazamiento .....	109
Ilustración 144: TimerScript.....	110
Ilustración 145: Asset Store 1.....	111
Ilustración 146: Asset Store 2.....	111
Ilustración 147: Modelos 3D .....	112
Ilustración 148: Modelos humanoides.....	112
Ilustración 149: Importación del modelo 3D.....	113
Ilustración 150: Inserción del modelo 3D .....	113
Ilustración 151: Creación del arma.....	114
Ilustración 152: Creación de la bala .....	114
Ilustración 153: Creación de la cabeza rotatoria .....	115
Ilustración 154: Script de inteligencia artificial .....	116
Ilustración 155: Los disparos del arma.....	117
Ilustración 156: El enemigo.....	117
Ilustración 157: zonas de movimiento.....	119
Ilustración 158: Movimiento de los enemigos .....	119
Ilustración 159: Vista desde arriba .....	121
Ilustración 160: Vista en primera persona .....	121
Ilustración 161: Vista en tercera persona .....	122
Ilustración 162: Comportamiento de las cámaras.....	122
Ilustración 163: Puertas invisibles.....	127
Ilustración 164: El botón pausar/continuar.....	129
Ilustración 165: Menú del pacman .....	131
Ilustración 166: Build Settings .....	132



## 1. INTRODUCCIÓN:

A continuación se va hacer una breve introducción al proyecto para exponer cuál ha sido su motivación, qué objetivos se han perseguido en su realización y cuáles son los contenidos ofrecidos en este manual de usuario.

### 1.1. Estructura del documento:

A continuación vienen especificados los diferentes capítulos que componen este manual:

- **Introducción:** Presentación de las motivaciones y los objetivos del proyecto.
- **Estado del arte:** Presentación de los motores de videojuegos, sus características más importantes, su evolución desde los inicios, los tipos de motores más representativos y algunos ejemplos existentes.
- **Unity 3D:** Presentación del motor Unity 3D, su evolución en el tiempo y las características que hacen que sea uno de los motores más utilizados del momento. Además se citan varios ejemplos de videojuegos realizados con Unity.
- **Historia del Pacman:** Presentación del videojuego Pacman, los elementos que lo componen y su historia.
- **Primer acceso al motor:** Breves indicaciones para instalar unity y probar el juego de la demo.
- **La interfaz de usuario de unity:** Explicación del editor gráfico de unity junto con las diferentes vistas que lo componen y las opciones que permiten cada una de las vistas.
- **Crear un nuevo proyecto:** Diferentes tutoriales para aprender a usar unity. Tutoriales que abarcan desde la manipulación de objetos hasta implementación de inteligencia artificial pasando por texturas, iluminación, sonidos, editor de terrenos, cielo, agua, creación de scripts, prefab, colliders, shaders, propiedades de materiales físicos y sistemas de partículas.
- **Pacman:** Explicación de algunas funcionalidades incluidas en el juego ejemplo “Pacman” mediante tutoriales. Algunos de estos tutoriales son la inclusión de múltiples cámaras en una escena, inclusión de una máquina de estados para los fantasmas o añadir puertas invisibles.
- **Compilación para Webplayer:** Pasos a seguir para compilar el pacman creado para PC a un pacman para Webplayer.
- **Gestión del proyecto y presupuesto:** El detalle de las fases, tareas y actividades en las que se ha dividido el desarrollo del proyecto.



Además se especifican los recursos, tiempos y costes que han sido necesarios.

- **Comercialización y conclusiones:** las diferentes maneras posibles para comercializar el proyecto y las conclusiones que se han deducido al realizar el manual o con el aprendizaje de unity.
- **Líneas futuras:** las líneas de exploración que aún faltan por investigar y estudiar afín de descubrir más utilidades de unity.
- **Glosario y referencias:** glosario de términos y acrónimos utilizado en el proyecto. Referencias a las páginas web donde se ha conseguido información.

## 1.2. Motivación:

Cada año se publican varios estudios sobre la evolución del mercado de los videojuegos, tanto de la parte material como de la parte software. En estos estudios se pueden ver las tendencias que evolucionan cada vez más de prisa y cifras impresionantes que muestran el peso de los videojuegos en la economía mundial.

En el año 2011, el mercado de los videojuegos ha generado un volumen de negocio de 41,9 billones de euros en el mundo y las previsiones para los años por venir auguran que podría llegar a los 60,6 billones de euros. Este auge se explica por varias razones, la llegada al mercado de nuevas generaciones de consolas portátiles, la evolución favorable de los videojuegos de los móviles o en línea.

Partiendo de este contexto de evolución constante, la elección de la plataforma para la creación de videojuegos por parte de los creadores es clave. Actualmente los videojuegos se pueden crear para diferentes dispositivos, consolas de salón, de móvil, de PC, etc..., por lo cual lo más inteligente es elegir una herramienta que permite crear aplicaciones multiplataforma para abarcar el mayor número de usuarios posible.

Este razonamiento hace que Unity 3D sea la herramienta elegida por muchos desarrolladores para la creación de sus videojuegos. En efecto Unity 3D ofrece diversas ventajas con respecto a los demás motores de videojuegos que hacen que sea cada vez más cotizado. En los apartados que siguen se van a ver en detalle las características de unity.



### 1.3. Objetivos del proyecto:

El objetivo de este proyecto fin de carrera es la elaboración de un manual de usuario para la utilización del motor de videojuegos Unity 3D. Este manual no pretende explicar todas las funcionalidades de Unity 3D puesto que no se dispone del tiempo necesario para ello pero ayudará al usuario que desea crear un videojuego en sus comienzos.

Para un aprendizaje más práctico y a modo de ejemplo, se va a proceder a la creación de un “Pacman” utilizando la versión gratuita de Unity. A lo largo de la formación ofrecida en este manual, el usuario aprenderá a:

- Modelar y crear entornos
- Incluir un personaje controlable por el jugador.
- Escribir scripts sencillos para crear objetos interactivos, eventos dinámicos, gestionar colisiones, etc...
- Crear objetivos, desafíos, objetos, un sistema de score.
- Personalizar la interfaz
- Utilizar diferentes modos de distribución: PC y Webplayer.

Además de la explicación de una serie de funcionalidades básicas en Unity 3D y del desarrollo del Pacman, se va a ofrecer al usuario de este manual información adicional sobre los motores de videojuegos y su evolución.

Las etapas seguidas para la realización del proyecto de fin de carrera son:

- Estudio de la historia de los motores de videojuegos
- Historia de Unity 3D.
- Estudio de las herramientas y utilidades ofrecidas por Unity 3D.
- Uso de estas herramientas para la creación de un videojuego.
- Elaboración de un videojuego 3D para que sirva como ejemplo.
- Plasmar toda la información adquirida en este manual.



## 2. ESTADO DEL ARTE:

En este apartado se va a hacer un recorrido a través de la historia de los motores de videojuegos, su evolución en el tiempo y los diferentes tipos de motores que han surgido.

### 2.1. Motor de videojuegos:

Un motor de videojuegos es un conjunto de herramientas que realizan cálculos geométricos y físicos utilizados en los videojuegos. Este conjunto de utilidades representa un simulador ágil en tiempo real que reproduce las características de los mundos imaginarios en los que transcurren los videojuegos. El objetivo es permitir al equipo de desarrollo de un videojuego concentrarse sobre el contenido del juego y no sobre la resolución de problemas informáticos.

El fin que persiguen los motores de juego es que una vez incorporadas todas las funciones del motor, usuarios sin conocimientos de programación pueden manejar con facilidad el motor.

Puesto que las funcionalidades proporcionadas por un motor son limitadas, estos con frecuencia son dedicados a un tipo específico de juego. La manipulación del motor se hace generalmente a través de un lenguaje de script o por la interfaz.

Una línea borrosa separa el motor del videojuego y muy a menudo no se sabe distinguir entre ambos. Los motores ofrecen componentes reutilizables que pueden ser manipulados para reproducir un juego. Algunas de los componentes que forman el motor pueden ser la carga, la animación de modelos, la detección de colisiones entre objetos, la física, interfaces gráficas de usuario o incluso herramientas de inteligencia artificial. Mientras que el contenido del juego está compuesto por los modelos, texturas específicas, el comportamiento ante las colisiones de objetos o la forma que tienen los objetos de interactuar con el entorno.

Un motor de video juegos debe ofrecer como mínimo las siguientes utilidades:

- Un motor 3D que permite la creación y visualización de un juego en un universo 3D.
- Un motor audio que permite la integración de elementos sonoros y música en el juego.





- Un motor físico que permita gestionar los comportamientos físicos de los objetos en un universo 3D, como la gravedad por ejemplo.
- Herramientas de gestión de red para añadir por ejemplo un componente multi-jugadores en red.

Otros conceptos muy ligados a los motores de videojuegos son las API y SDK. Las API (interfaz de programación de aplicaciones) son las interfaces de software ofrecidas por los sistemas operativos, bibliotecas y servicios para que el usuario pueda aprovecharlas en la creación del juego. Un SDK (Kit de desarrollo de software) es un conjunto de bibliotecas, API y herramientas disponibles para la programación. La mayoría de los motores de videojuegos proporcionan API en sus SDK.

Existen diferentes maneras de clasificar los motores de videojuegos dependiendo de las características que nos interesan. Una de estas características es el coste, efectivamente existen motores de videojuegos gratuitos o de pago que pueden llegar hasta el medio millón de dólares. Estos últimos son destinados a estudios de desarrollo profesional con gran poder financiero. Estos motores ofrecen un marco de juego y en algunos casos un entorno de desarrollo integrado. Este entorno permite crear o importar todos los elementos que se quieren integrar en el juego y determinar mecánicas de juego con la ayuda de los scripts.

Los motores gratuitos están destinados a los desarrolladores principiantes o independientes y no suelen tener interfaz global. Para utilizar estos motores hay que descargar un SDK (Software Development Kit), un kit de desarrollo (herramientas y documentación) que permite utilizarlos, pero se debe generar el programa del juego entero a mano, escribiendo código: CrystalSpace, Ogre 3D, Irrlicht.

Los motores intermediarios, en esta categoría se incluyen los motores con herramientas de desarrollo más potentes. Estos motores disponen de un entorno de desarrollo integrado (IDE) que permite la creación de juegos manipulando los elementos en una interfaz gráfica y generando automáticamente una gran parte del código del juego. Este tipo está dirigido esencialmente a los profesionales independientes: Unity 3D, Torque 3D.

Los motores profesionales están dirigidos a los profesionales con gran presupuesto. Son los motores más completos y los más potentes del mercado, se mantienen al día por sus creadores y disponen de los últimos avances tecnológicos: Id Tech 4, Unreal Engine 3, Cry Engine 3.



## 2.2. Historia de los videojuegos:

Durante mucho tiempo, las empresas han desarrollado sus propios motores de juego pero como un producto propio y sin comercializarlos. Con el aumento del coste de creación de un motor, las empresas han empezado a especializarse en la fabricación de los motores, ya sean completos o componentes para su posterior venta a otras empresas. Este tipo de empresas se las conoce como proveedoras de middleware.

El término “Game engine” o motor de juego ha aparecido en 1990, haciendo referencia a los programas utilizados en los juegos de tiro (también llamados FPS, first person shooter) como **Doom**. La arquitectura de **Doom** distinguía de manera clara entre los componentes del motor de juego y los recursos digitales del juego como los gráficos, sonidos y reglas de avance que reproducen el ambiente del juego. De esta manera los componentes del motor han podido ser reutilizados mediante sencillas modificaciones en otros juegos lo que ha permitido sacar al mercado nuevos videojuegos de manera más rápida.

Algunas empresas se han especializado en el desarrollo de los programas dichos Framework, ofreciendo funciones que se pueden personalizar. La utilización de estos productos permite evitar reinventar la rueda y reutilizar una serie de programas cuya eficiencia ya ha sido probada, incluyendo elementos necesarios para la creación de un juego. Estos programas facilitan el desarrollo de gráficos, sonidos, de la física y la implementación de la inteligencia artificial. Algunos motores de este tipo: **Gamebryo** y **RenderWare**.

Otros programas se centran en ofrecer un tipo concreto de utilidad como por ejemplo **SpeedTree**. Esta especialización permite generar imágenes más convincentes que los motores más generalistas. Algunos frameworks se usan con todo el código fuente, otros con la documentación de su interfaz de programación para permitir la reutilización de otros programas.

Últimamente la tendencia de los motores de videojuegos es que sean cada vez más intuitivos y amigables para que los pueda usar gente con muy pocas nociones de programación. Además los motores multiplataforma son muy cotizados puesto que se quieren comercializar los videojuegos de diferentes maneras, para consolas de salón, para móviles, PC, etc.



## 2.3. Tipos de motores:

Existen muchos tipos diferentes de motores de videojuegos y distintas maneras de clasificarlos ya sea por las herramientas que incluyen, las funcionalidades que ofrecen, el grado de programación requerido o por lo que engloban, si es un motor más bien general o especializado en un tipo de juego concreto y así una larga lista de enfoques que nos permitirían hacer una clasificación u otra. Se van a citar tres de estos tipos de motores roll-your-own version, mostly-ready version y el motor point-and-click.

### 2.3.1. Roll-your-own:

Este tipo de motores son del nivel más bajo e implica que básicamente para crear un videojuego haya que crearse el motor. La creación de los motores consume mucho tiempo y recursos pero muchas empresas siguen haciéndolo, para ello recurren a aplicaciones disponibles al público tales como las API como XNA, DirectX, OpenGL, la API de Windows y Linux y SDL y bibliotecas comerciales o de código abierto. Estas herramientas facilitan la creación del motor de físicas, funcionalidades graficas u otros.

La ventaja de este tipo de motores es la flexibilidad que permite al creador la selección de los componentes necesarios para el videojuego sin sobrecargar el procesador con herramientas que le restan potencia y por consecuente calidad al videojuego. El gran inconveniente de este tipo de motores y lo que hace que no sea de los más apreciados por los usuarios es que requiere mucho tiempo y recursos.

### 2.3.2. Mostly ready :

Motores listos para su uso, proporcionan muchas herramientas para la creación de videojuegos. Herramientas como una interfaz de usuario, un motor de físicas u opciones de renderización. Estos motores suelen requerir algo de programación para la creación del videojuego y generalmente proporcionan un mejor resultado de renderización que los motores “Roll your own” en menos tiempo y con menos esfuerzos.

Existen varios motores de este tipo y van desde los gratuitos como ORGE, Genesis hasta los que tiene un precio elevado como Unreal, id Tech o GameBryo pasando por los de bajo precio como Torque.



### 2.3.3. Point-and-click

El nivel más alto y el más extendido hoy en día. Son motores completos para la creación de un videojuego de manera intuitiva, amigable y teniendo muy pocas nociones de programación. Dicho esto hay que saber elegir el motor adecuado según el tipo de videojuego que se quiera crear porque estos motores pueden ser limitados al no poder englobar las herramientas necesarias para cualquier tipo de videojuegos.

A pesar de estas limitaciones son motores muy solicitados y apreciados por los creadores porque permiten la creación de un juego de manera muy rápida sin tener que preocuparse por los aspectos matemáticos y centrándose en los aspectos creativos. De allí que existan ya varios motores de este tipo como GameMarker, Torque, Game Builer o el que vamos a ver en esta manual Unity 3D.

## 2.4. Listado de motores:

Se van a citar algunos ejemplos motores de videojuegos existentes en el mercado. Como ya se ha mencionado anteriormente van desde los gratuitos hasta los de pago.

### 2.4.1. Motores open source:

Los motores más importantes con licencia gratuita y código abierto con libre distribución del código y desarrollo libre.

Nombre	Descripción
Aleph_One	Motor desarrollado en el lenguaje C y utilizado para implementar el juego “ <i>Marathon</i> ”. Licencia tipo GPL.
Blender	Motor desarrollado en lenguaje C++ con licencia GPL. Un ejemplo de juego creado es “ <i>Color Cube</i> ”.
Box2D	Motor desarrollado en lenguaje C++ con licencia MIT. Un juego desarrollado con este motor es “ <i>Angry Birds</i> ”.
Build_engine	Motor desarrollado en lenguaje C y licencia Custom. Un ejemplo de juego es “ <i>Duke Nuken 3D</i> ”.
Crystal_Space	Motor desarrollado en lenguaje C++ con licencia LGPL.
Genesis3D	Motor desarrollado en lenguaje C.
id Tech	Motor desarrollado en lenguaje C con licencia GPL. Ejemplos de juegos creados con este motor son “ <i>Doom</i> ”, “ <i>Doom 2</i> ”, “ <i>HeXen</i> ”, “ <i>Quake II</i> ” y “ <i>Quake III Arena</i> ”.
OpenSceneGraph	Motor desarrollado en lenguaje C++.



Panda3D	Motor desarrollado en lenguaje C++ y licencia BSD. El juego “Pirates of the Caribbean” ha sido creado con este motor.
Axiom Engine	Motor desarrollado en lenguaje C# con licencia GPL.
Cube	Motor desarrollado en lenguaje C++ y su licencia es zlib.
Delta3d	Motor desarrollado en lenguaje C++ con licencia LGPL.
Flexible	Motor desarrollado en lenguaje C++ y su licencia es LGPL.
Isometric_Free Engine	Un ejemplo de juego desarrollado con este motor es “Unknown Horizons”.
PixelLight	Motor desarrollado en lenguaje C++ con licencia LGPL.
Quake engine	Motor desarrollado en lenguaje C con licencia GPL.
Second Life	Motor desarrollado en lenguaje C++.

*Tabla 1: Motores open source*

#### 2.4.2. Motores freeware:

Los motores con licencia freeware siendo motores de software no libre que se distribuyen sin costo para su uso y por un tiempo ilimitado una versión de prueba.

Nombre	Descripción
Adventure Game Studio	Este motor es uno de los más populares a la hora de desarrollar juegos de aventuras amateur.
DX Studio	Motor freeware con un conjunto muy completo de herramientas para el desarrollo de videojuegos en 3D. Posee una licencia de pago para tener acceso a ciertas funcionalidades.
Unity	Motor muy completo en el que se puede desarrollar para web, Windows y Mac. Obteniendo una licencia se puede desarrollar para iPhone, Android, Nintendo Wii, Playstation 3 y la Xbox 360.
Unreal Engine	Completo motor para desarrollar juegos para PC. Obteniendo una licencia se puede desarrollar juegos para Xbox 360 y PS3.

*Tabla 2: Motores freeware*



### 2.4.3. Motores propietario:

Los motores de videojuegos con licencia en la que el usuario tiene limitaciones para usarlo, modificarlo o redistribuirlo.

	Descripción
Bork 3D Game Engine	Motor para desarrollar juegos de plataformas para iPhone e iPad.
BigWorld	Servidor, cliente y herramientas de desarrollo de MMOG para Windows, Xbox 360 y PS.
Source engine	Desarrollado por “Valve Software” para el juego “Half-Life 2”.
Unity	Motor que permite el desarrollo de videojuegos para web, Windows, Mac OS X, iOS (iPod, iPhone, and iPad), Android, Nintendo Wii, Xbox 360 y PS3.
Enigma Engine	Motor de juegos de táctica en tiempo real, utilizado para “Blitzkrieg”.
Freescape (1986)	Uno de los primeros motores 3D propietarios, utilizado en “Driller” y “3D Construction Kit”.
id Tech 4 (también conocido “motor Doom 3”)	Utilizado para los juegos “Doom 3, Quake 4, Prey y Quake Wars”. En Septiembre de 2011 llegará a ser Open Source.
Infinity Engine	Permite la creación de videojuegos de rol.
RAGE	Creado por “Rockstar Games” para mejorar sus juegos para Xbox 360 y PlayStation 3. Utilizado para el juego Grand Theft Auto 4.
BRender	Motor gráfico 3D en tiempo real para simuladores y herramientas gráficas.
DX Studio	Motor que permite la creación de juegos en tiempo real y simuladores.
M.U.G.E.N	Creado por “Elecbyte” para juegos de lucha e 2D.
Q (game engine)	Completo framework desarrollado por el equipo de Direct3D que permite el desarrollo para PC, Wii, PS2, PS3, Xbox, Xbox 360, PSP, iPhone etc.
Scaleform	Motor gráfico utilizado para visualizar elementos Adobe Flash, HUDs y texturas animadas para juegos en PC, Mac, Linux, Xbox 360, PlayStation 2, PlayStation Portable, PlayStation 3 y Wii.
Zillions of Games	Utilizado para desarrollar juegos de tablero, como ajedrez.

Tabla 3: Motores propietario



### 3. UNITY 3D:

Unity 3D es un motor de creación de videojuegos 3D lanzado oficialmente como tal el 1 de Junio 2005. Este motor permite la creación de juegos y otros contenidos interactivos como diseños arquitectónicos o animaciones 3D en tiempo real.

Muchas personas interesadas por el desarrollo se topan con la dificultad de aprender los lenguajes de programación y los motores que los utilizan. Sin estudios de programación o de animación por ordenador, el aprendizaje de los conceptos, métodos y los principios necesarios para la creación de un videojuego se hace muy difícil.

Unity Technologies es una de las empresas que ha decidido rectificar esta situación. Desde el lanzamiento de la primera versión 1.0.1 en 2001, esta empresa danesa se ha esforzado para que sus herramientas sean accesibles y fáciles de usar. El equipo de desarrollo de unity ha decidido mantener el código fuente ofreciendo al usuario una interfaz gráfica completa de manera a que el usuario pueda controlar el código fuente sin tener que crear nuevos elementos en el código. Este factor ha hecho que unity sea muy popular entre los desarrolladores de videojuegos.

Unity pone la potencia de su motor al servicio de los utilizadores permitiéndoles obtener un resultado de máxima calidad con un mínimo de esfuerzo. Además las actualizaciones, mejoras e inclusión de nuevas funcionalidades no han cesado hasta llegar a la actual versión la 4.0 y su desarrollo sigue en curso. Unity existe en versión profesional que se puede adquirir previo pago y una versión libre completamente gratuita que se puede descargar en la página Web de Unity. Esta última versión incluye menos funcionalidades pero aun así permite la creación de videojuegos de muy buena calidad.

Unity es una aplicación 3D en tiempo real y multimedia además de ser motor 3D y físico utilizado para la creación de juegos en red, de animación en tiempo real, de contenido interactivo compuesto por audio, video y objetos 3D.

Este motor no permite la modelización pero permite crear escenas que soportan iluminación, terrenos, cámaras, texturas. Fue creado en un principio para la plataforma Mac y ha sido exportado a Windows, permite obtener aplicaciones compatibles con Windows, Mac OS X, iOS, Android, Wii, Playstation 3, Xbox 360, Nintendo, iPad, iPhone con Web gracias a un plugin y recientemente desde la versión 3.5 con el formato Flash de Adobe.

La evolución de la industria del ocio y del marketing hace que los videojuegos tiene que ser producidos rápidamente por lo cual muchas empresas se ayudan de soluciones integradas como unity para que sus desarrolladores saquen productos de manera sencilla y rápida.



Unity presenta varias ventajas que hacen que sea uno de los motores de videojuego más cotizado del momento. En los siguientes párrafos se van a ir citando todas estas ventajas:

- Permite la importación de numerosos formatos 3D como 3ds Max, Maya, Cinema 4D, Cheetah3D y Softimage, Blender, Modo, ZBrush, FBX o recursos variados tales como texturas Photoshop, PNG, TIFF, audios y videos. Estos recursos se optimizan posteriormente mediante filtros.
- es compatible con las API graficas de Direct3D, OpenGL y Wii. Además de ser compatible con QuickTime y utilizar internamente el formato Ogg Vorbis
- En Unity, el juego se construye mediante el editor y un lenguaje de scripts por lo cual el usuario no tiene que ser un experto en programación para usarlo. En efecto, este software tiene la particularidad de incluir la herramienta de desarrollo MonoDevelop con la que se pueden crear scripts en JavaScript, C# y un dialecto de Python llamado Boo con los que extender la funcionalidad del editor, utilizando las API que provee y la cual encontramos documentada junto a tutoriales y recursos en su web oficial.
- La estructura de los juegos creados por Unity viene definida mediante escenas que representan alguna parte del juego.
- Incluye un editor de terrenos que permite la creación de estos partiendo de cero. Este editor permite esculpir la geometría del terreno, su texturización y la inclusión de elementos 3D importados desde aplicaciones 3D o ya predefinidos en Unity.
- Si no se quiere modelar en 3D y se necesitan recursos para un videojuego, en la propia aplicación se puede acceder al Asset Store donde existe multitud de recursos gratuitos y de pago. Incluso se puede extender la herramienta mediante plugins que se obtienen en esta misma tienda.
- En cuanto a los usuarios que no tienen ninguna noción de programación existen plugins como Playmaker que permiten "programar con cajitas" mediante máquinas de estados, de una forma visual. La utilización de estos plugins supone un coste añadido.





- Dispone de una interfaz de desarrollo muy bien definida e intuitiva que permite crear rápidamente min-juegos.
- Existe en varias versiones en función de los módulos elegidos, la versión más simple destinada a los amateurs es gratuita.
- Tal como se ha especificado antes es multiplataforma por lo cual permite la creación de juegos compatibles con distintas consolas (Para la mayoría de las plataformas citadas, se requiere una licencia adicional):
  - Microsoft Windows o Mac OS X ejecutable
  - Linux
  - En la web (a través del plugin Unity Web Player para Internet Explorer, Firefox, Safari, Mozilla, Netscape, Opera, Google Chrome y Camino) en Windows y OS X
  - En Mac OS X Dashboard widget.
  - Para Nintendo Wii
  - iPhone / iPad
  - Google Android
  - Google Native Client
  - Microsoft Xbox 360
  - Adobe Flash
  - Sony PlayStation 3
  - BlackBerry PlayBook

En definitiva la creación de videojuegos es mucho más sencilla y rápida con esta aplicación por lo cual el número de usuarios está en aumento constante.

Por último decir que Unity 3D ganó varios premios. Por ejemplo en 2006 fue finalista para el mejor uso de los gráficos en Mac OS X en los premios de diseño Apple. En 2010, fue ganador del premio de la innovación tecnológica del Wall Street Journal en la categoría de software.

#### EJEMPLOS DE JUEGOS DESARROLLADOS CON UNITY 3D:

##### **Para PC:**

- EA's Tiger Wood's PGA Tour Online



*Ilustración 1: Tiger Woods*



- StarWars: The quest of R2-D2
- Volkswagen Rally Touareg
- Among the Sleep
- Wasteland 2
- Naval War: Arctic Circle



*Ilustración 2: Naval War Arctic Circle*

#### **Para iPhone:**

- Samurai: Way of the warrior
- Bubble Bang



*Ilustración 3: Bubble Bang*

#### **Para Wii:**

- My Animal Centre
- Max and the magic marker



*Ilustración 4: Max and the magic marker*



**Para Web (Online):**

- Battlestar Galactica Online
- Space Paranoids Online
- Family Guy Online

**Para iPhone, iPad, dispositivos Android y navegadores de Internet:**

- Cabals: The Card Game.



*Ilustración 5: Cabals, The card game*

**Para Playstation:**

- Escape Plan



*Ilustración 6: Escape Plan*

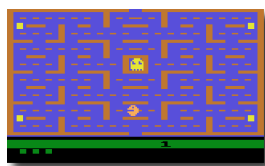
## 4. HISTORIA DEL PACMAN:

El Pac-Man es uno de los videojuegos más populares en el mundo. ¿Quién no conoce el personaje del Pac-Man? Su creador Tohru Iwatani se inspiró en una pizza a la cual faltaba un trozo y creó el personaje para la empresa Namco.

Este juego apareció en el año 1979 en Japón bajo el nombre de “Puck Man” y tuvo un gran éxito por su originalidad y porque contrastaba con los juegos existentes en esa época. Los padres encontraron por fin un videojuego no violento para sus hijos. Su nombre viene definido por una palabra japonesa que corresponde a una onomatopeya del ruido de apertura y cierre de la boca “Paku Paku”. En las primeras versiones, la inteligencia artificial del juego dejaba mucho que desear porque los movimientos de los fantasmas estaban predefinidos y se podían memorizar fácilmente recorridos. Este fallo fue rápidamente solucionado.

Este juego innovador y con un potencial comercial excepcional fue rápidamente recuperado por la empresa Midway. El nombre fue adaptado para el mercado norteamericano en Pac-Man para evitar caer en transformaciones vulgares del nombre. El juego tuvo un gran éxito y permitió a los Estados Unidos reactivar la creatividad de los creadores de videojuegos.

Posteriormente el juego fue adaptado para las diferentes consolas y ordenadores de la época. Existen versiones del Pac-Man para: Atari 2600 (1982), Apple II (1983), Atari 5200 (1983), Commodore 64 (1983), Intellivision (1983), MSX (1984), NES (1984), Oric (Oric Munch 1983), ColecoVision (Mouse Trap 1983), Famicom (1987), Game Gear (1990), etc....



Versión Atari 2600



Oric Much Oric



Mouse Trap Colecovision

*Ilustración 7: versiones del pacman*

El principio del juego consiste en que el personaje que se encuentra en un laberinto tiene que comer todas las capsulas antes de que los fantasmas lo atrapen. Algunas de las capsulas le permiten al Pac-Man ser invulnerable y así destruir los fantasmas. A medida que se avance en el juego aparecen elementos como fruta o llaves. En la versión original todos los niveles eran idénticos y con prácticamente ningún límite. En cuanto a los fantasmas, cada uno tiene su nombre tal como viene indicado en la tabla que sigue:



Color del fantasma	Personalidad	apodo
Rojo	Shadow	Blinky
Rosa	Speedy	Pinky
Azul	Bashful	Inky
Naranja	Pokey	Clyde

*Tabla 4: Fantasma del pacman*

Cada uno de los fantasmas tiene su propio comportamiento: blinky se mueve rápidamente, Pinky tiene tendencia a poner emboscadas, Inky es imprevisible y Clyde finge la indiferencia. Esta diversidad se obtiene con reglas simple de programación.

Desde la salida del juego han aparecido numerosos clones del Pacman en el mercado de los cuales vamos a citar los siguientes:

- Guppy en Exelvision, una variante con un pescado y pulpos.
- Clean Sweep en la consola Vectrex, en este juego el usuario dirige un aspirador que recupera unas monedas, esto hace que engorde y se ralentice por lo cual tiene que ir a la banca para depositar lo que ha recogido.
- Devil World es un clone creado para Nintendo. El personaje principal Tamagon, tiene que limpiar la pantalla de cruces. La diferencia reside en la diversidad de los enemigos pero el principio es el mismo recorrer la pantalla evitando los enemigos.



## 5. PRIMER ACCESO AL MOTOR:

Unity existe en versión gratuita y versión profesional. Este proyecto se ceñirá a las funcionalidades accesibles a los principiantes, dicho de otra manera a la versión gratuita.

Se puede conseguir la versión gratuita en la página Web de Unity, la versión actual es la 3.5.5. Cuando ya está instalada la aplicación, el primer acceso es a un proyecto de demostración llamado “Island demo”. Tal como su nombre lo indica se trata de un proyecto destinado a presentar las capacidades de Unity y a permitir a los nuevos usuarios descubrir las funcionalidades que ofrece estudiando las creaciones de sus desarrolladores.



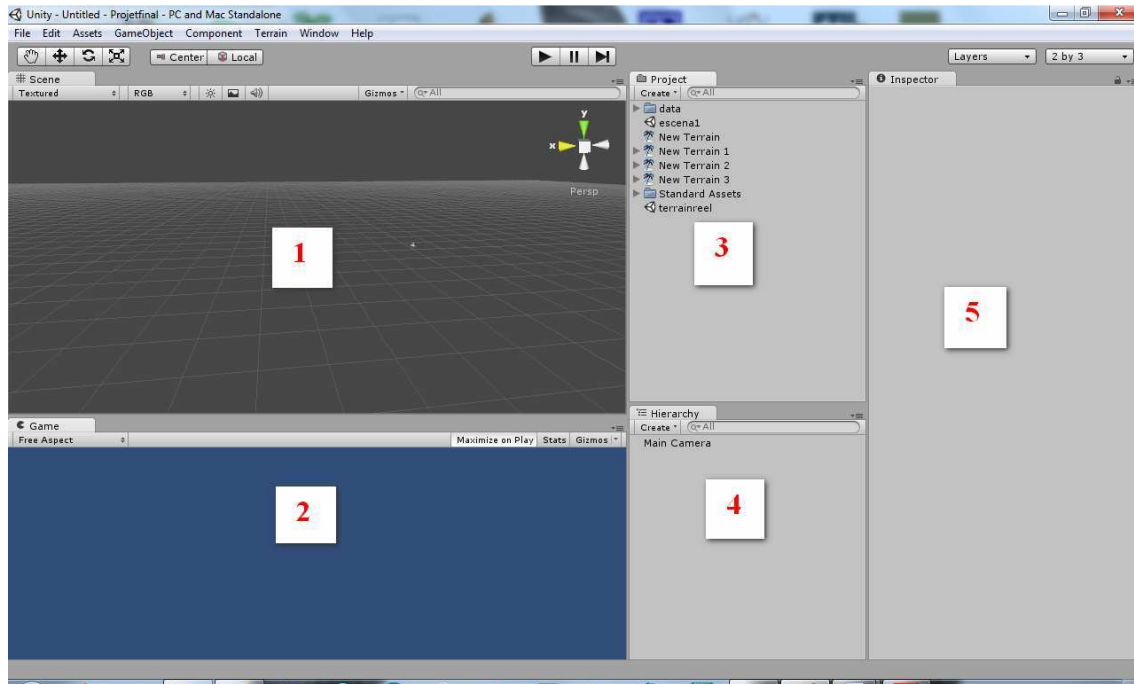
*Ilustración 8: Tropical paradise demo de unity*

En la siguiente etapa se van a detallar las diferentes partes de la interfaz gráfica de la aplicación.



## 6. LA INTERFAZ DE USUARIO DE UNITY:

Al arrancar Unity 3D aparece la siguiente interfaz de usuario. Existen principalmente 5 áreas de la interfaz de Unity 3D, numeradas en la imagen de abajo.



*Ilustración 9: Interfaz de usuario*

La ventana número 1 es la ventana de escena ó el área de construcción de Unity donde se construye visualmente cada escena del juego.

La ventana número 2 es la vista de juego donde se obtiene una pre-visualización del juego. En cualquier momento se puede reproducir el juego y jugarlo en esta vista.

La vista número 3, la vista de proyecto es la librería de assets para el juego. Se pueden importar objetos 3D de distintas aplicaciones a la librería, se pueden importar texturas y crear otros objetos como Scripts o Prefabs que se almacenaran aquí. Todos los assets que se importen en el juego se almacenaran aquí para que se puedan usar en el juego.

Un juego normal contendrá varias escenas y una gran cantidad de assets por lo cual es necesario estructurar la librería en diferentes carpetas que harán que los assets se encuentren organizados y sea más fácil su uso.

La vista número 4 es la vista de jerarquía y contiene todos los objetos de la escena actual.

La vista número 5 ó la vista de inspector permite seleccionar objetos y que se muestren sus propiedades y personalizar sus características.

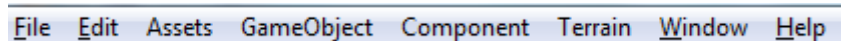




También contiene la configuración para ciertas herramientas como la herramienta de terrenos si se tiene el terreno seleccionado.

## 6.1. Menú de la aplicación:

A continuación se muestra el menú de opciones de Unity que se encuentra en la parte superior izquierda. A lo largo del manual se irán mostrando las utilidades de cada una de las secciones de este menú.



*Ilustración 10: Menú de la aplicación*

## 6.2. Botones de control:

Debajo del menú de Unity vienen definidos los botones de control tal y como se ve en la imagen que sigue:



*Ilustración 11: Botones de manipulación*



Este botón permite moverse alrededor de la vista de escena:

- ALT permite rotar.
- COMMAND/CTRL permite hacer zoom.
- SHIFT incrementa la velocidad de movimiento mientras se usa la herramienta.



Permite mover los objetos seleccionados en la escena, el movimiento se puede realizar en los ejes X, Y e Z.



Permite rotar los objetos seleccionados en la escena, la rotación se puede realizar en los ejes X, Y e Z.



Permite escalar los objetos seleccionados en la escena. En este caso también se puede escalar en cualquiera de los ejes X, Y y Z.

En unity 3d se puede reproducir el juego sin salir del editor, la imagen que sigue muestra los controles de reproducción. El primer botón permite lanzar el juego y visualizarlo, el segundo permite pausarlo y el último a la derecha permite saltar adelante en el juego. Se puede visualizar el videojuego en la vista de juego tal como está o maximizando la pantalla.

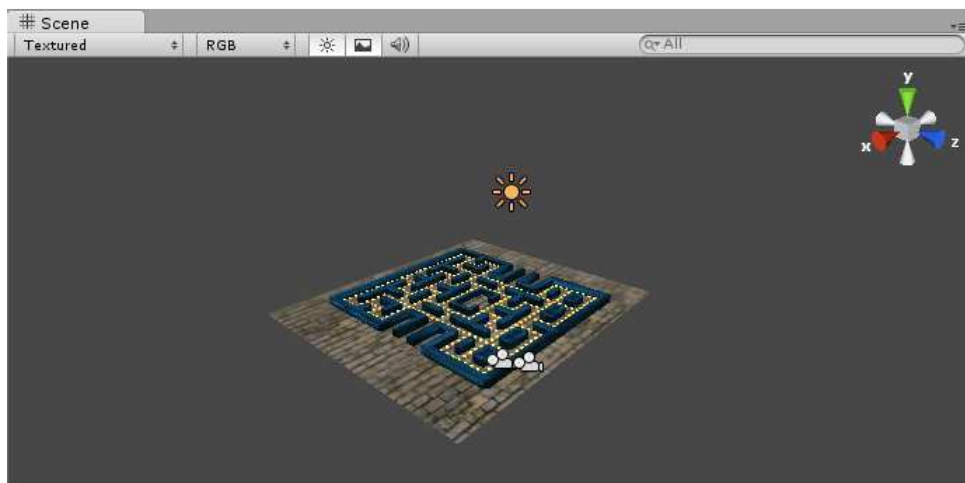




*Ilustración 12: Botones de control*

### 6.3. Vista de escena:

Los juegos en Unity se dividen en escenas y para la creación de estas últimas se dispone de un entorno 3d que es la vista de escena. Esta vista permite la visualización del aspecto grafico de nuestra aplicación en todo momento.



*Ilustración 13: vista de escena*

La vista de escena permite crear escenas de forma sencilla, se pueden ir arrastrando objetos desde la vista de proyecto y manejarlos (posicionar, escalar, rotar...). En esta vista también se pueden editar los terrenos y moldearlos.

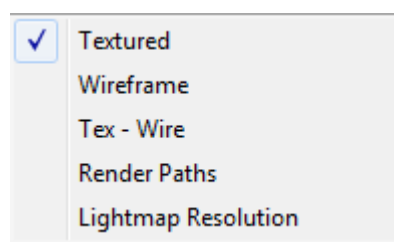
A continuación se van a explicar las opciones de la vista de escena:



*Ilustración 14: Menú de la vista de escena*

- Render Mode:

Por defecto aparecerá en “Textured” y al hacer clic se despliega una lista de opciones de renderizado.



*Ilustración 15: Render Mode*



- Textured: Las texturas se renderizan en la vista.
- Wireframe: Las superficies no se renderizan, solo vemos la malla.
- Textured Wireframe: Las texturas se renderizan, pero también vemos la malla.

Prueba a cambiar entre las tres opciones para entender cómo se ven realmente.

### Modos de visualización:

En la esquina superior derecha de la vista de escena se encuentra el Gizmo que se puede observar a continuación. Este Gizmo permite ver la escena desde diferentes puntos de vista. El modo de visualización por defecto es en perspectiva 3D y se obtiene haciendo clic sobre la caja central gris.



*Ilustración 16: Modos de visualización*

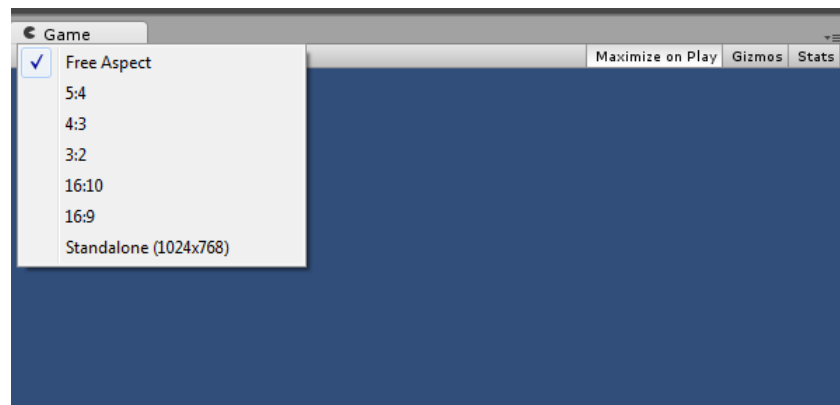
- El cono verde “y” permite pasar al modo Top-Down (vista aérea de la escena).
- El cono rojo “x” permite pasar al modo Side (derecha).
- El cono azul “z” permite pasar al modo Front (frontal).

También están los 3 conos grises que llevan a los siguientes modos: Back (Atrás), Left (Izquierda) y Bottom (Abajo).

## 6.4. Vista del juego:

La vista del juego permite pre-visualizar el juego en la vista tal como está o maximizándola a pantalla completa. Se puede ver en esta vista el juego en movimiento para comprobar el buen funcionamiento del juego.

La vista de Game es muy útil pero tiene un inconveniente, cuando se prueba el juego en el ordenador, se mueve el personaje con las teclas o el ratón lo cual no ayuda mucho si el juego va a ser compilado para un móvil. En estos casos las pruebas se tienen que hacer mediante emulador o compilando una versión para el móvil.



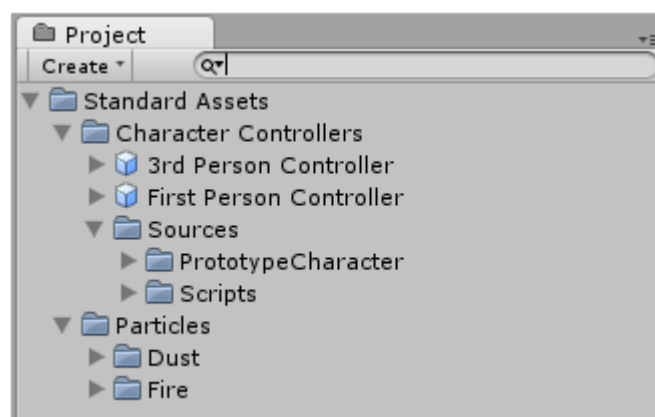
*Ilustración 17: Vista del juego*

- El botón “Maximize on Play”: permite maximizar a pantalla completa la vista del juego.
- El botón “Gizmos”:
- El botón “Stats”:
- Desplegable “Free Aspect”:

## 6.5. Vista de proyecto:

La vista de proyecto es una librería de assets para la creación del juego. En esta vista se guardan todos los elementos que se crean y se importan para ser usados en el juego. Estos elementos pueden ser objetos 3D, texturas, sonidos o scripts.

En esta ventana se puede tener multitud de recursos pero esto no implica la utilización de todos esos recursos para la creación del videojuego.

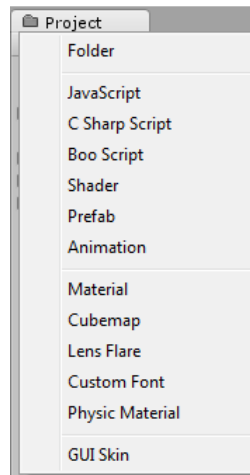


*Ilustración 18: Vista de proyecto.*

Puesto que esta vista contiene tanto los elementos usados en la escena como los que no, puede llegar a tener una gran cantidad de datos. Esto implica que hay que mantener una buena estructura para facilitar el trabajo al usuario. La organización de esta vista, se puede hacer mediante la creación de una jerarquía



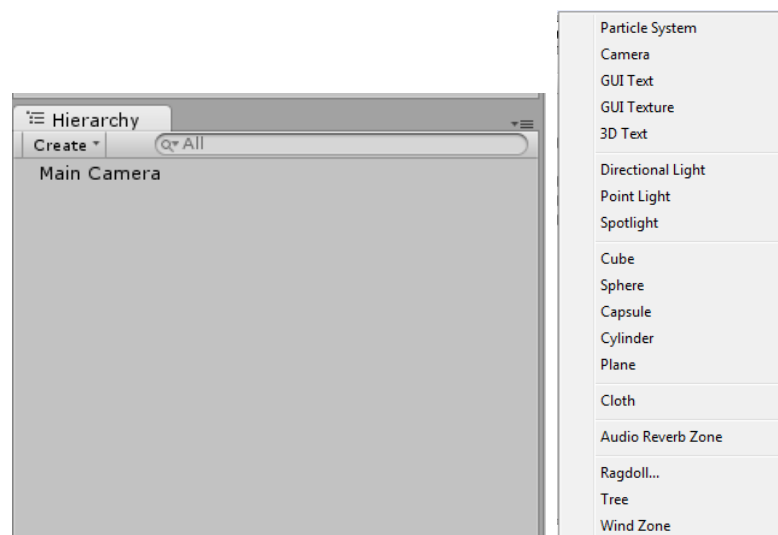
de carpetas. El desplegable de “create” contiene las opciones que permiten una buena organización de la vista.



*Ilustración 19: Menú de la vista de proyecto.*

## 6.6. Vista de jerarquía:

La vista de jerarquía contiene todos los elementos que conforman la escena del juego. Se pueden manipular todos los objetos de la escena a partir de esta vista, cada vez que se introduce un elemento en la escena, se añade una entrada para este elemento en esta vista. Al seleccionar un objeto en esta vista se selecciona también en la escena y en el inspector. Lo cual permite y facilita el movimiento, el escalado, la rotación y el borrado del objeto o la edición de sus parámetros. Todo ello sin necesidad de tocar ni una línea de código.



*Ilustración 20: Vista de jerarquía*

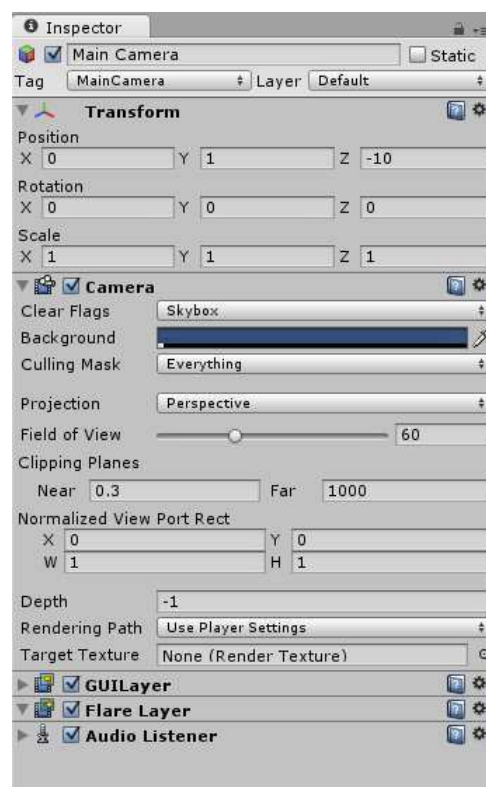


## 6.7. Vista de inspector:

En la vista de inspector se muestran los parámetros de los objetos seleccionados en la vista de escena o en la vista de jerarquía. De esta manera al seleccionar un objeto en la escena, se pueden modificar sus parámetros en el inspector (posición, rotación, traslación, etc). El inspector sirve también como panel de herramientas para algunos elementos tales como los terrenos permitiendo su escultura, añadir texturas y más cosas.

En la vista de inspector se pueden ver los elementos que definen el comportamiento de los objetos, estos elementos se llaman componentes. Además aquí mismo se puede proceder a la activación o desactivación de los objetos y los componentes asociados. Los componentes pueden ser scripts, animaciones, colliders...etc.

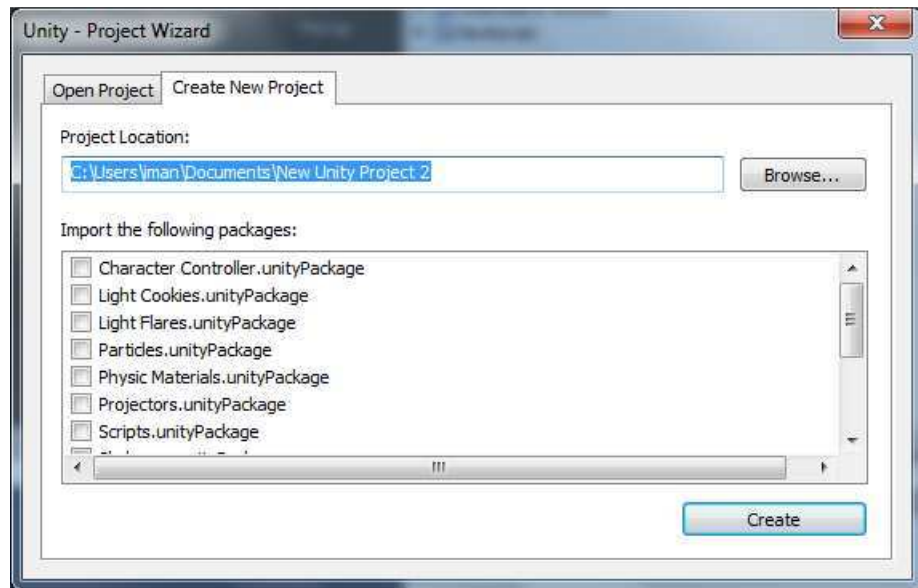
Por último aclarar que el inspector cambia según el elemento seleccionado.



*Ilustración 21: Vista de inspector*

## 7. CREAR UN NUEVO PROYECTO:

La creación de un nuevo proyecto implica ir a la barra superior de unity y hacer clic en File -> New Project para que se abra la ventana de dialogo “Create new Project” tal como muestra la imagen:



*Ilustración 22: Creación de un proyecto*

Primero hay que seleccionar la carpeta en la que se quiere guardar el proyecto haciendo clic en “Browse”. Para la creación de un nuevo proyecto hay que importar los paquetes necesarios para el juego. Los paquetes son conjuntos de assets y se pueden importar al crear el proyecto o posteriormente en cualquier momento. Los paquetes importados pueden ser los predefinidos en Unity, otros creados por el usuario o conseguidos en la Web. Del listado que aparece en la imagen se seleccionan los assets y se hace clic en “Create”. El paquete básico a importar es el conjunto de assets estándar.

Al crear el nuevo proyecto, se reinicia Unity 3D creando la estructura del proyecto en la carpeta indicada. Se abre la pantalla de inicio de Unity 3D en la que aparecen los assets importados en la vista de proyecto y una cámara en la vista de escena y de jerarquía.

Algunos de los paquetes más importantes:

- Character Controller permite crear scripts para desplazar el personaje.
- Particles permite crear partículas por ejemplo fuego, humo, cascada y explosiones.
- Skyboxes permite la creación de cielos.
- Terrain permite crear terrenos y esculpirlos.
- Water para crear agua (mar, ríos, etc).
- Tree Creator para crear árboles.



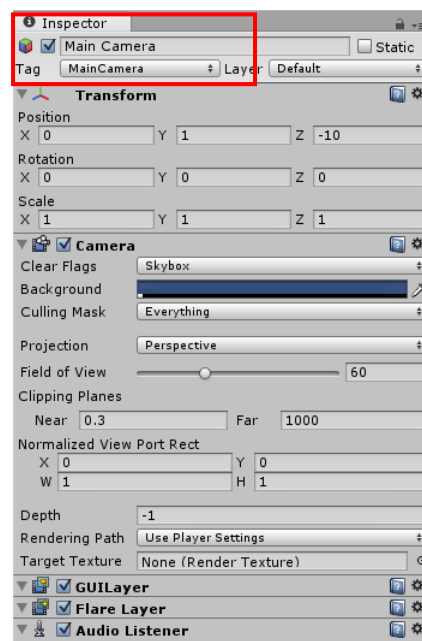
## 7.1. Tutorial: Crear una nueva escena:

Unity 3D funciona por escenas y al crear un nuevo proyecto nos crea una nueva escena automáticamente, para crear otra escena, ir a File → New Scene. Hay que guardar la escena creada automáticamente en File → Save Scene. La escena se guarda en la carpeta de asset del proyecto y aparecerá en la vista de proyecto.

Una vez guardada la escena ya se puede ir creando todos los elementos que la componen. Diferentes maneras de manipular la escena:

- Para ver la escena desde los diferentes ángulos se puede rotar alrededor utilizando ALT y moviendo el ratón.
- Para acercar o alejar la escena utilizando el zoom se puede usar la rueda del ratón.
- Las flechas del teclado permiten mover la escena a la izquierda, derecha, arriba y abajo. Se puede hacer lo mismo manteniendo clicado el ratón y moviéndolo en un sentido u otro.
- Se puede aumentar la velocidad de movimiento de la escena manteniendo SHIFT.

Al crear una escena aparece por defecto una cámara en nuestro mundo. Haciendo clic sobre esta “Main Camera” se pueden visualizar las características de la cámara en el inspector y modificarlos (posiciones, rotación, escala). La cámara define el punto de vista del jugador y se pueden combinar varias cámaras en una misma escena (se verá más adelante en el manual).



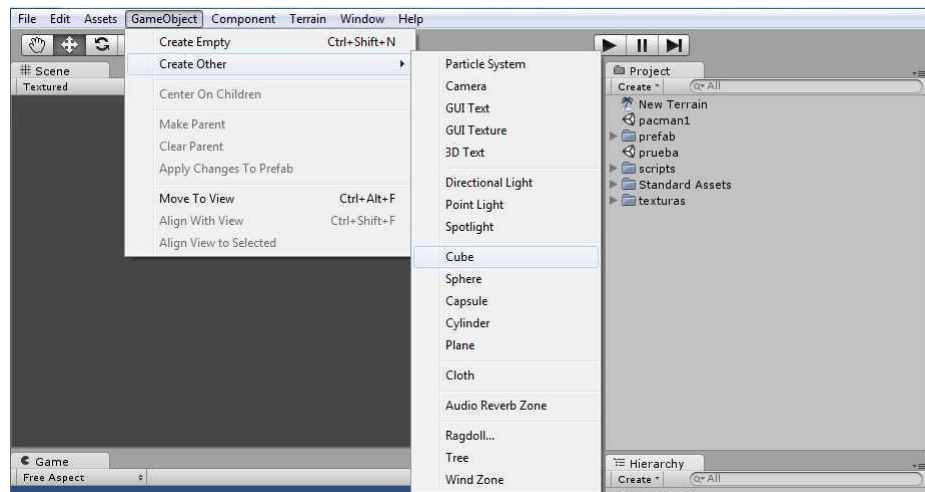
*Ilustración 23: Creación de una escena*



## 7.2. Tutorial: Creación de objetos:

En este apartado se va a explicar cómo se van creando los elementos que van a componer la escena y la manipulación de estos.

Para crear los objetos predefinidos en Unity 3D, se accede a GameObject > Create Other se despliega una lista de elementos.



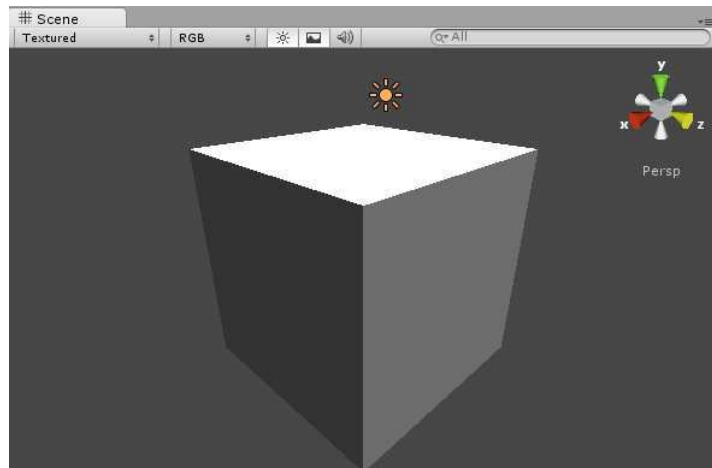
*Ilustración 24: Creación de un GameObject*

La lista se compone de varios GameObject, algunos de los más importantes son:

- Camera: permite introducir una cámara en la escena, aunque al crear un proyecto Unity coloca una cámara por defecto.
- Directional light: introduce una luz direccional uniforme que permite ver la escena.
- Cube: poner un cubo en la escena.
- Sphere: meter una esfera
- Capsule: introduce una capsula
- Cylinder: poner un cilindro
- Plane: introduce un plano en la escena.

Al seleccionar uno de los “GameObject”, este aparece en las vistas de escena, la de juego y jerarquía. Además en la vista de inspector se pueden ver los atributos del objeto para la parametrización de este.



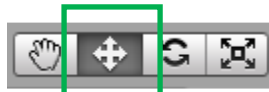


*Ilustración 25: GameObject cubo*

### 7.3. Tutorial: Manipulación de objetos:

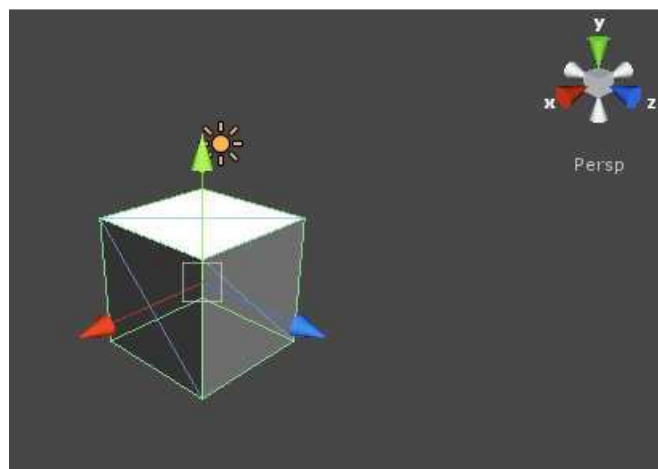
Una vez creado el cubo, este se puede modificar para tener las dimensiones, posición y forma necesarias para lograr la escena deseada.

Se puede modificar la posición del cubo utilizando el segundo botón de control, el resaltado en el recuadro:



*Ilustración 26: manipulación de objetos 1*

Tal como se muestra en la imagen que sigue, se puede mover el cubo en los ejes X (rojo), Y (verde) e Z (azul). Para mover el cubo solo hay que hacer clic sobre el eje en el que se quiere mover y arrastrar hasta la nueva posición.



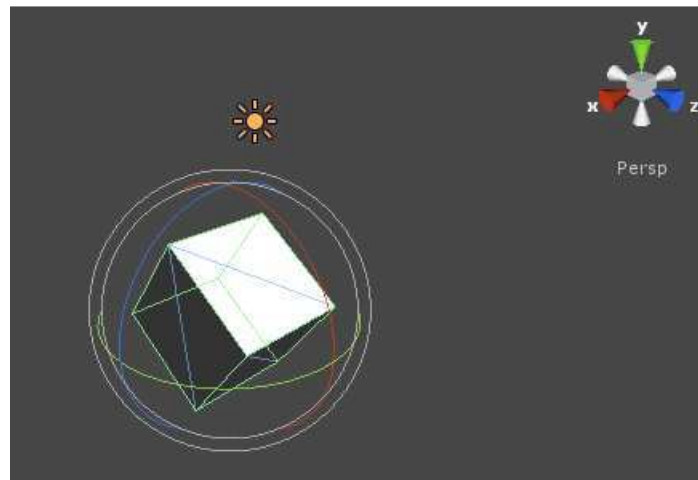
*Ilustración 27: Movimiento del objeto 1*

El tercer botón de control (el resaltado por el cuadro verde) permite rotar el cubo.



*Ilustración 28: Opción de rotación*

Comparando la imagen que sigue con la 27, se puede ver cómo ha rotado el cubo. Los diferentes círculos que aparecen alrededor del cubo cuando se selecciona la opción de rotación permiten al usuario elegir el eje en el cual quiere realizar esta operación. Para rotar un elemento, se hace clic sobre el eje representado por los círculos de colores y se mueve el ratón en el sentido seleccionado.



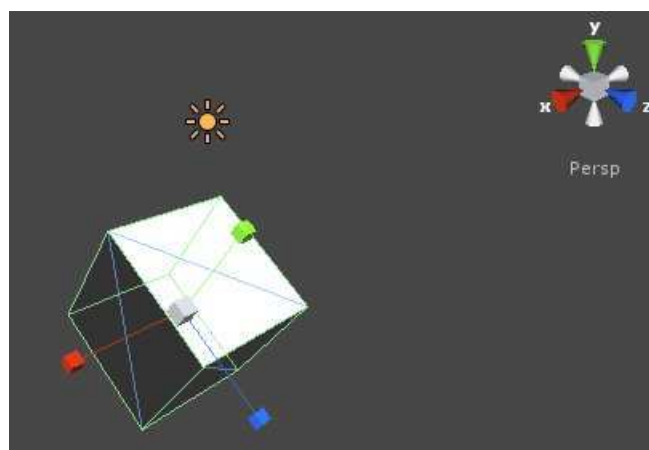
*Ilustración 29: Rotación del objeto*

El último botón de control permite escalar el cubo para hacerlo más grande o más pequeño.



*Ilustración 30: manipulación de objeto 3*

Se puede observar en la imagen que sigue como se escala el cubo, esto se puede realizar en los tres ejes por separado ó sobre los tres a la vez haciendo clic en el centro del objeto y moviendo el ratón.



*Ilustración 31: Escala del objeto*



- La manipulación del cubo se puede hacer de manera más precisa utilizando la vista del inspector:
- El movimiento del cubo se hace determinando las coordenadas X, Y e Z en el recuadro de posición.
- La rotación del cubo se hace indicando los ángulos de rotación en los ejes X,Y e Z.
- El escalado del cubo se hace de la misma manera indicando en qué medida se quiere escalar en cada uno de los ejes.

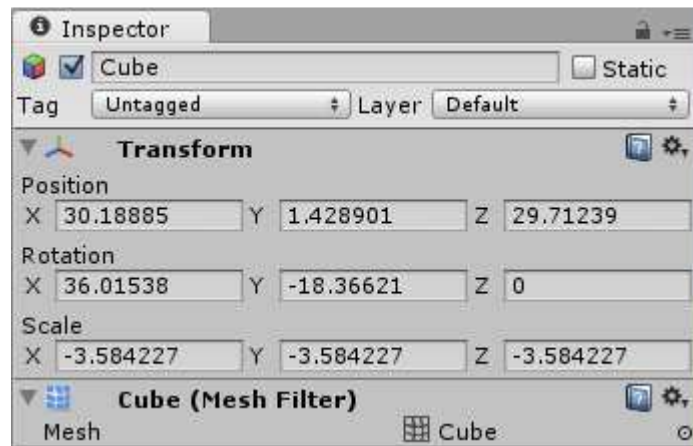


Ilustración 32: Manipulación desde la vista del inspector

#### 7.4. Tutorial: Añadir luz a la escena:

La visualización de la escena requiere la introducción de iluminación. Se pueden introducir una o varias luces a la escena. Acceder a “GameObject > Create Other” y seleccionar una de las luces “Directional Light”, “Point Light” ó SpotLight según el efecto deseado. En este caso se va a usar la luz direccional “Directional Light” y se van a combinar varias luces para obtener un resultado más realista. Para que la luz tenga efecto sobre la escena es necesario activar el botón en la vista de escena.

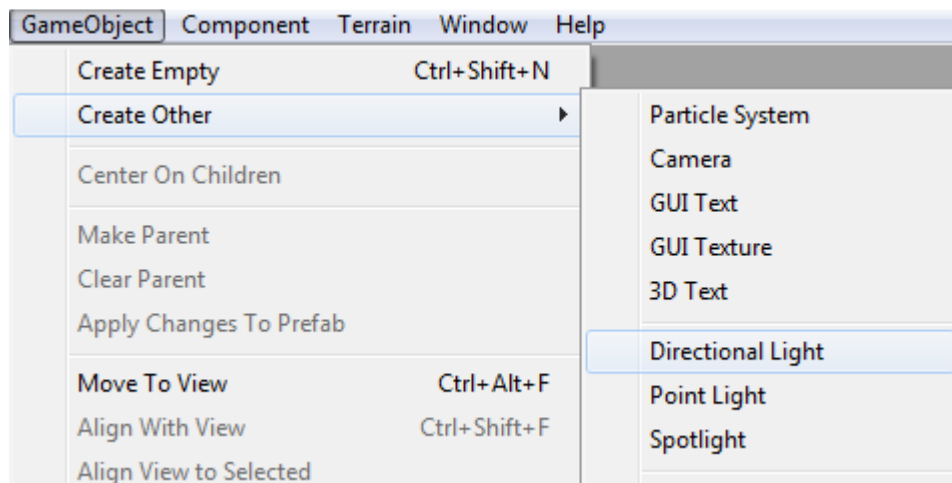
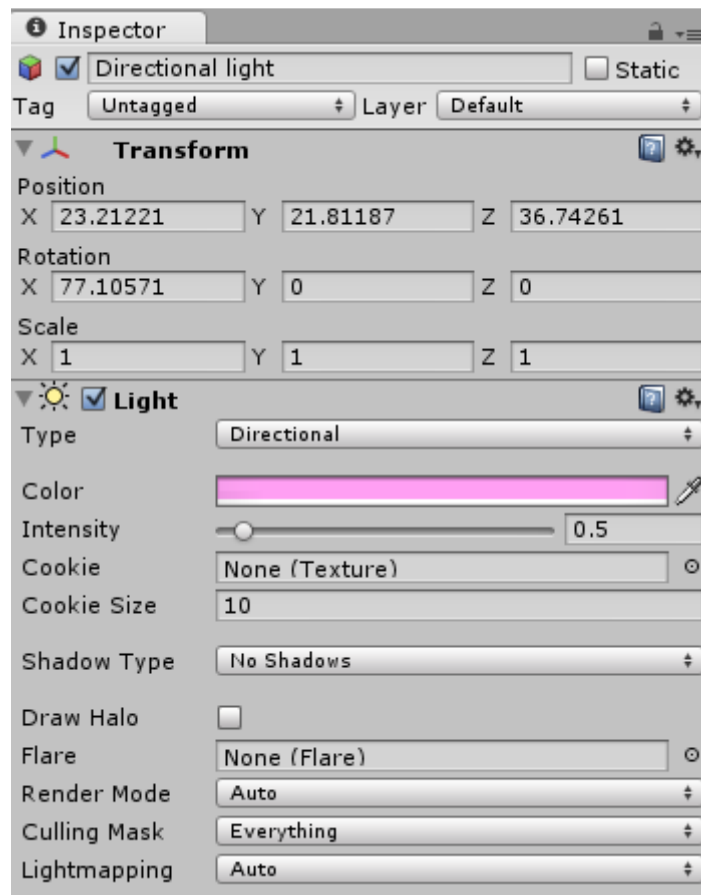


Ilustración 33: Directional light



Las luces se pueden manipular de la misma manera que cualquier objeto tanto en la vista de escena como en el inspector de manera a obtener la iluminación adecuada. La luz direccional tiene efecto global independientemente de su posición.

Lo importante para personalizar la iluminación de la escena y obtener el efecto deseado es definir la dirección (rotación) y los parámetros que siguen:



*Ilustración 34: Características de la luz*

#### Explicación de los parámetros:

- Color: el color de la luz
- Intensity: la intensidad del brillo de la luz., su valor por defecto para una luz direccional es de 0,5.
- Cookie: se utiliza como una máscara que determina el brillo de la luz en diferentes lugares.
- Shadow Type: tamaño de la proyección de una cookie.
- Draw Halo: un halo esférico de luz que se dibuja con un radio igual en rango.



## 7.5. Tutorial: Añadir textura:

Una vez creado el objeto, se le puede añadir textura. Existen varias maneras de añadir textura al GameObject.

Se selecciona el elemento ya sea en la vista de escena o en la vista de jerarquía para que se puedan visualizar los atributos del elemento en la vista de inspector. En el componente de “Mesh Render” en “Materials” aparece un círculo (resaltado en la imagen en el cuadro verde).

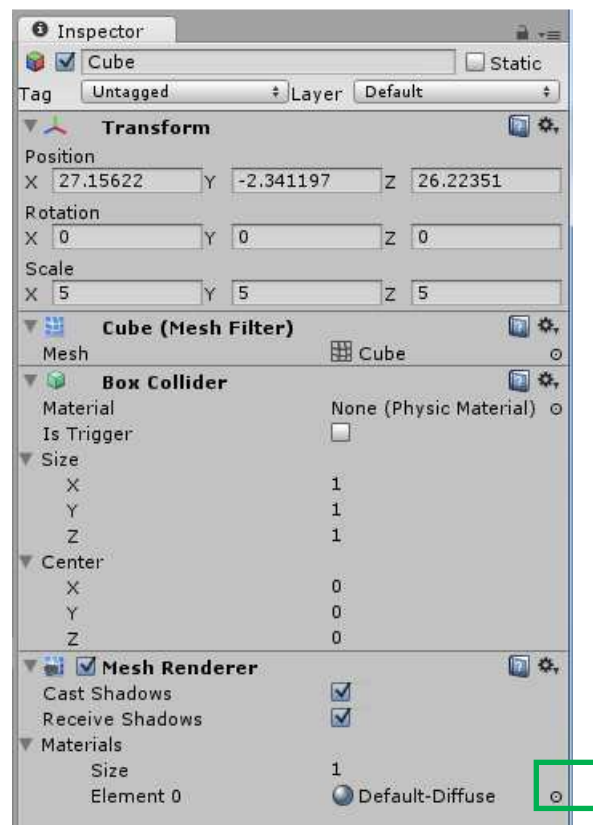


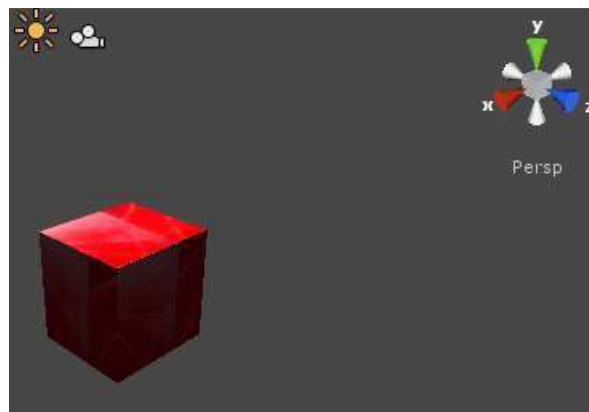
Ilustración 35: Texturización de un objeto.

Al hacer clic en el círculo aparece una ventana con las texturas existentes. Se selecciona una textura y esta se aplica automáticamente al cubo.



*Ilustración 36: Selección de la textura*

Otra manera de añadir textura es a partir de la vista de proyecto, en la carpeta de Standard Assets > texturas, escoger una de las texturas y arrastrarla encima del cubo.



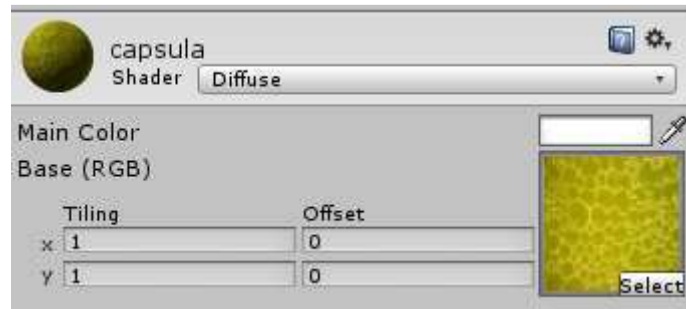
*Ilustración 37: Objeto con textura*

El usuario también puede importar una textura y añadírsela al GameObject. El usuario puede descargarse una textura o crearla, se arrastra la textura al Unity en la vista de proyecto en la carpeta de texturas. Una vez la textura importada se aplica al GameObject.

## 7.6. Tutorial: Configurar la textura

En este apartado se va a trabajar un poco más el aspecto visual de la textura partiendo del cubo texturizado en el punto anterior.

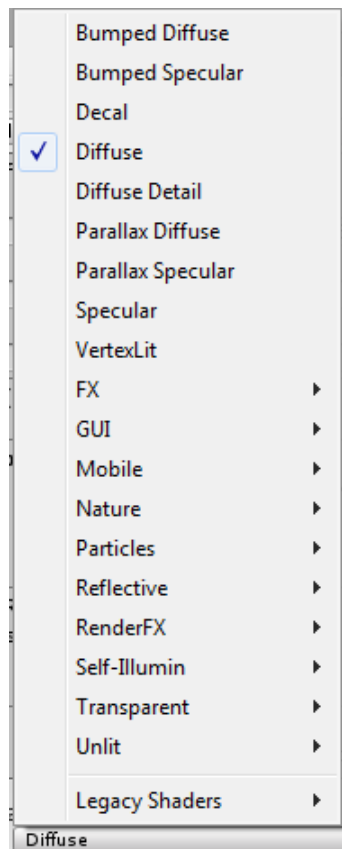
Cuando se selecciona el cubo y se observa la vista de inspector se puede ver en la parte inferior las propiedades de la textura:



*Ilustración 38: Parámetros de textura*

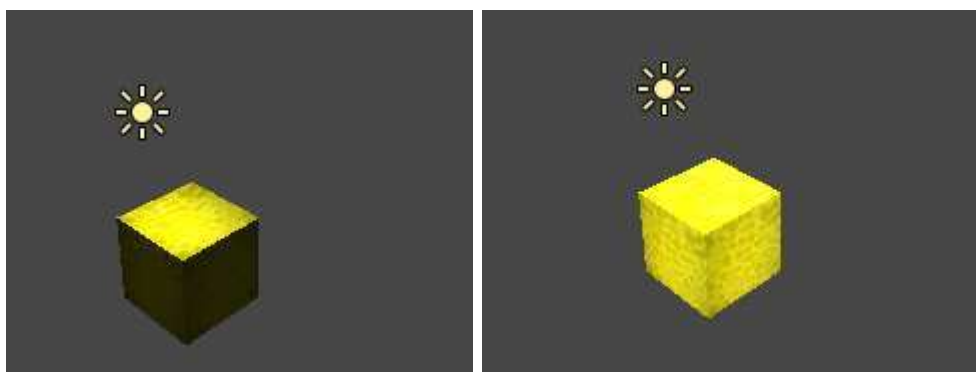
La caja permite la modificación de la textura elegida para el elemento y el “Tiling” indica la cantidad de veces que la textura se proyecta sobre las caras del cubo. Ésta opción resulta especialmente útil en algunos casos por ejemplo componer suelos y paredes con una única textura de baldosa y un plano sobre el que se repite varias veces.

Las texturas se ven afectadas por la luz. Las texturas representan un material y este puede reflejar o absorber la luz en función de si es un material difusor o especular. Ahora se hace clic en la lista desplegable del “Shader”(concepto que se va a ver en capítulos posteriores) y se selecciona “Specular” para hacer que el material del cubo pase a reflejar la luz que le llega. Unity incluye por defecto distintos shaders para materiales pero la gran mayoría son o bien difusores o bien especulares.



*Ilustración 39: Reflexión de la luz*

Se puede observar que al ejecutar estos pasos el aspecto visual del material cambia. Se puede ir probando los diferentes shaders que integra unity 3D. Se puede concluir que con una única textura según el shaders que se le aplique se obtiene un material u otro.



*Ilustración 40: Efecto material*





## 8. EDITOR DE TERRENOS:

Un editor de terrenos es imprescindible para cualquier desarrollador que quiere crear un juego en un entorno exterior. Unity integra uno desde la versión 2.0, lo cual facilita y acelera la construcción de los entornos.

### 8.1. Importación de los heightmaps:

Los heightmaps son gráficos en 2D, cuyas zonas claras y oscuras representan la topografía del terreno. Se pueden importar y representan una alternativa a las herramientas que permiten la definición de los niveles del terreno con el pincel en Unity.

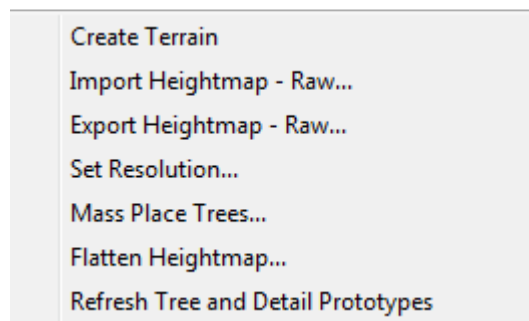
Se crean en una aplicación de ilustración como photoshop y se guardan con formato .RAW, se utilizan a menudo en el desarrollo de los videojuegos porque son fáciles de exportar entre aplicaciones de diseño y entornos como Unity.

Puesto que se va a crear un terreno con las herramientas del editor de terrenos de Unity, en este manual no se van a utilizar heightmaps.

### 8.2. Tutorial: Creación de un terreno

Unity 3D dispone de una herramienta que permite definir terrenos. Unity maneja los terrenos como una malla plana, esta se puede texturizar y esculpir sin salir del editor. A partir del menú se puede crear un terreno haciendo clic en “Terrain -> Create Terrain”.

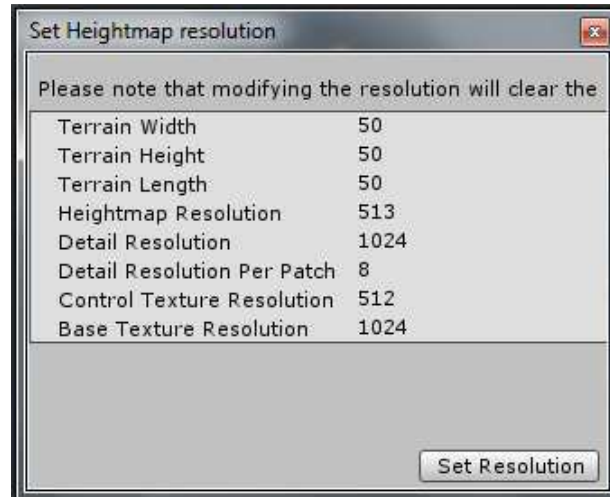
Se obtiene un terreno plano, si el terreno no es visible hay que desactivar las luces en la vista de escena. Se pueden modificar los parámetros del terreno utilizando la opción “Set Resolution” en el desplegable de Terrain”:



*Ilustración 41: Creación de un terreno 1*



Una vez creado el terreno, se pueden definir sus propiedades a través de la ventana “Set Heightmap Resolution”, a la que podemos acceder desde “Terrain -> Set Resolution”.

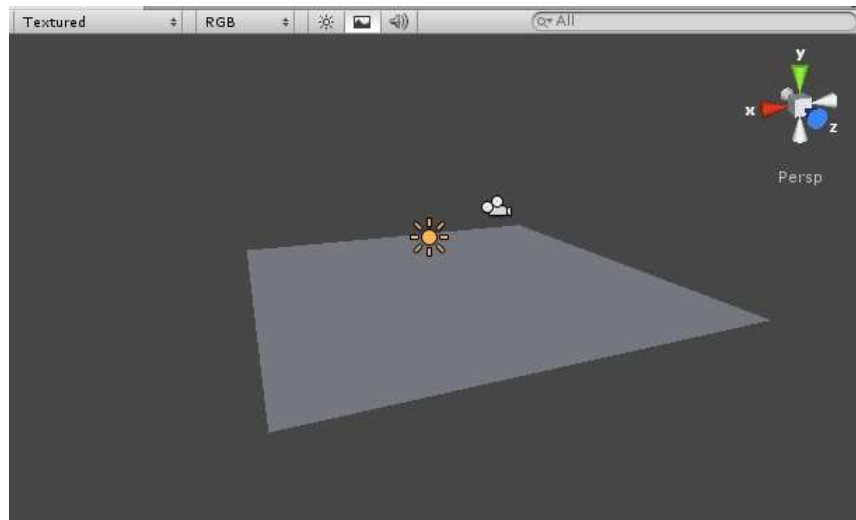


*Ilustración 42: Características del terreno*

Las propiedades de la ventana “Set Heightmap Resolution”:

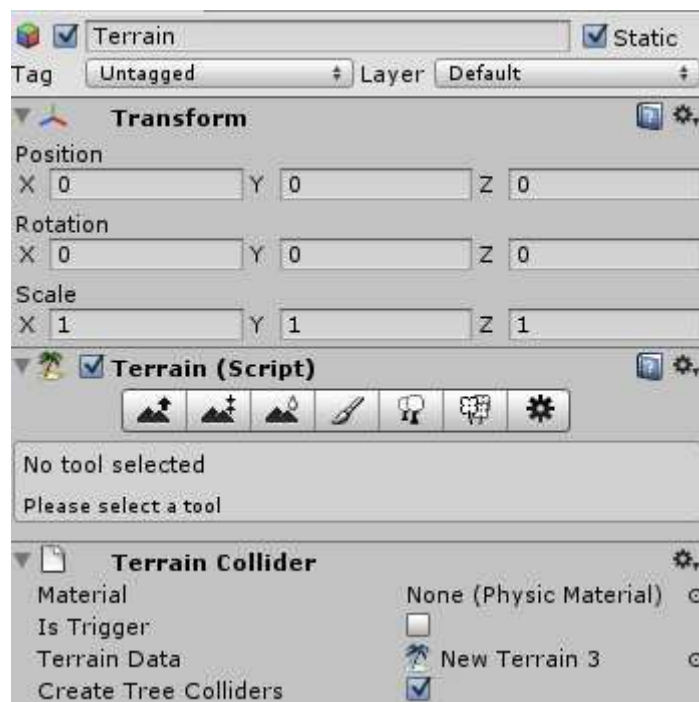
- Terrain Width: El ancho en metros del terreno.
- Terrain Length: La longitud en metros del terreno.
- Terrain Height: La altura máxima en metros del terreno.
- Heightmap Resolution: La resolución del heightmap expresada en píxeles. La textura que unity almacena para representar la topografía. Debe ser potencia de 2 + 1. (Ejemplo: 129,513)
- Detail Resolution: La resolución del mapa de detalles, cuanto más resolución, más precisión para dibujar los detalles del terreno y colocar objetos.
- Control Texture Resolution: La resolución de las texturas del terreno, más resolución significa más detalle, menor resolución equivale a más rendimiento.
- Base Texture Resolution: Esta es la resolución base de la textura que se renderiza desde distancia (LOD).

Una vez creado el terreno y modificados sus parámetros para hacerlo más manejable, se obtiene el siguiente resultado:



*Ilustración 43: Creación de terreno 2*

Para personalizar el terreno y hacerlo más realista hay que hacer uso de la herramienta de edición de terrenos. Se selecciona el terreno en la vista de escena o en la vista de jerarquía para que aparezcan en la vista de inspector la herramienta de edición de terrenos.



*Ilustración 44: Creación de terreno 3*

Esta herramienta está dividida en 3 partes:

- **Transform:** Permite mover, rotar y escalar el terreno sobre los ejes x,y,z. Igual que se ha explicado en el caso del cubo también se pueden hacer estas modificaciones desde los botones de control.



- Terrain Script: Contiene varias herramientas y propiedades para el terreno que se van a explicar más adelante.
- Terrain Collider: Contiene las propiedades de colisión para el terreno.

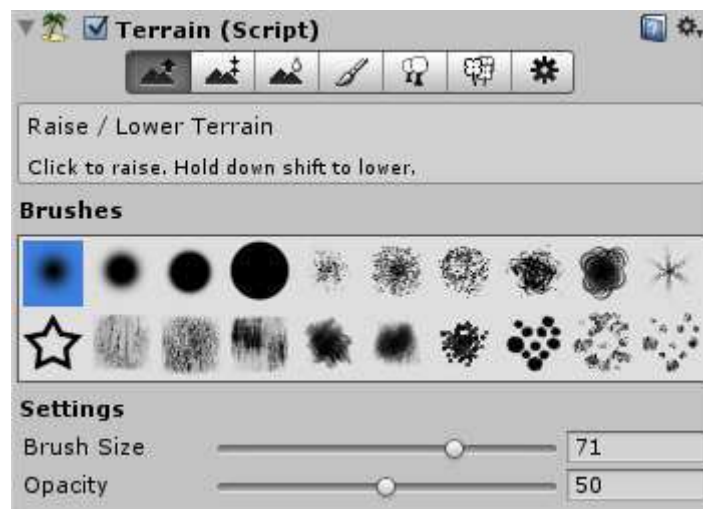
Se va a proceder a detallar el panel “Terrain Script” veremos una fila de botones. Estos botones editores de terreno permiten realizar diferentes tareas. La descripción de lo que permiten hacer los botones de izquierda a derecha:

- Raise / Lower: permite levantar y hundir la geometría del terreno usando un pincel.
- Set Height: pintamos el terreno con una altura límite.
- Smooth: permite suavizar un terreno para eliminar esquinas, por ejemplo.
- Paint Texture: permite pintar texturas sobre la superficie del terreno.
- Place Trees: permite colocar árboles.
- Paint Detail: permite dibujar los detalles del terreno como la hierba.
- Terrain Settings: Accede a las propiedades del terreno donde podemos cambiar varias propiedades.

### 8.3. Tutorial: La geometría del terreno

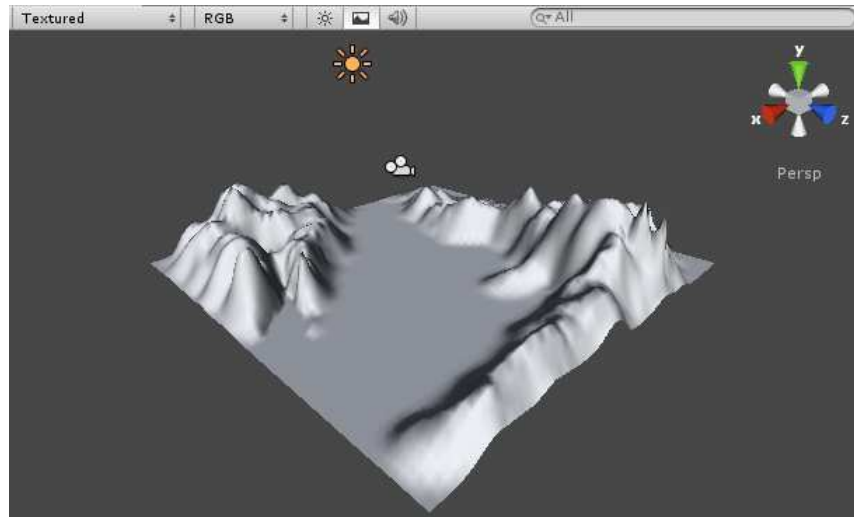
Las tres primeras opciones permiten modificar la geometría del terreno. La primera opción “Raise/Lower” nos permite levantar y hundir el suelo por lo cual se pueden definir montañas, dunas, ríos, etc...

Al seleccionar esta opción se puede elegir el tipo de pincel que se va a usar, su tamaño y la opacidad. La opacidad define la fuerza con la que se va a dibujar el terreno, una opacidad baja permitirá crear las formas del terreno despacio y una alta creara las formas de manera más pronunciada.



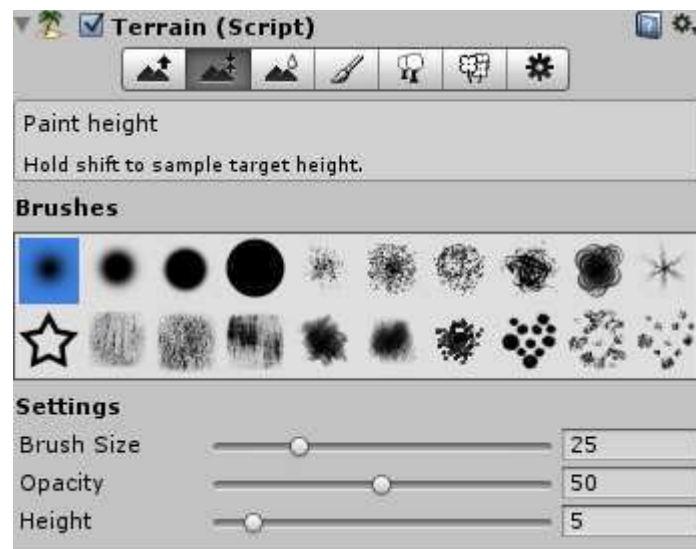
*Ilustración 45: Moldeado del terreno 1*

Primero se van a definir las alturas del terreno para ello solo hay que hacer clic en los sitios donde se quiere poner una montaña, cuanto más tiempo se mantiene pulsado el ratón más grande se hará la montaña. El terreno se transforma en la imagen que sigue:



*Ilustración 46: Moldeado del terreno 2*

La segunda opción “Set Height” permite fijar una altura máxima en el campo Height”



*Ilustración 47: Moldeado del terreno 3*

Al fijar una altura máxima, se puede seguir moldeando el terreno para conseguir darle la geometría adecuada. Una vez que se llega a la altura máxima el terreno se aplanará. La tercera opción “Smooth” permite suavizar los picos del terreno. Es recomendable esculpir por pasos, primero las partes grandes y después se refinan los detalles más pequeños. Al final se obtiene un terreno tal como se ve en la imagen:

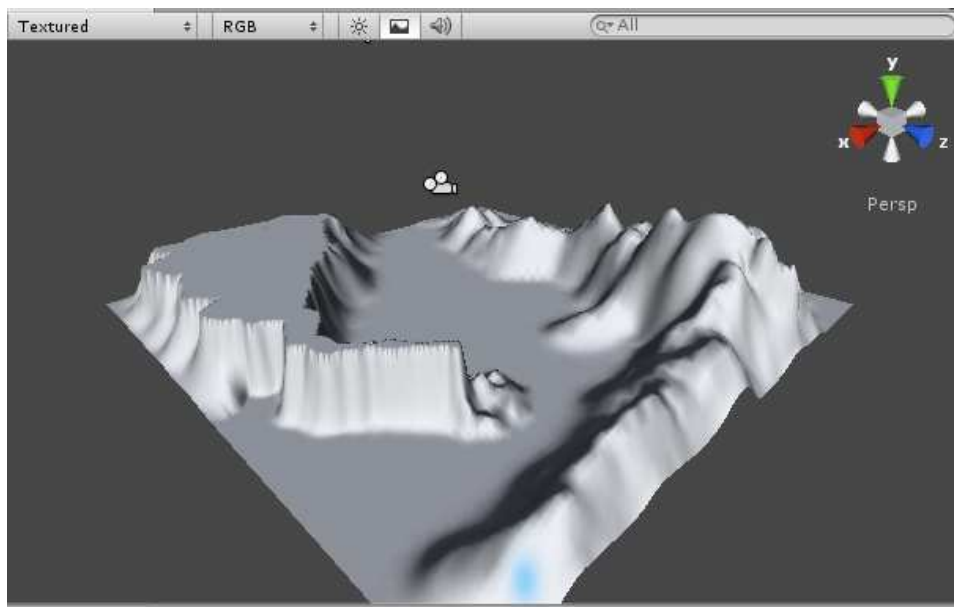


Ilustración 48: Resultado del moldeado

Una vez que se obtiene un terreno con el aspecto deseado, se pueden aplicar varias texturas. La primera textura es considerada como la textura base y se aplica al terreno entero. Las demás texturas se aplicaran en capas superiores.

La cuarto opción “Paint the terrain texture” permite dar textura al terreno. Primero se elige el tipo de brocha que se quiere usar en “Brushes”, seguido se añade la textura elegida en “Textures” haciendo clic en “Edit texture> Add Texture”

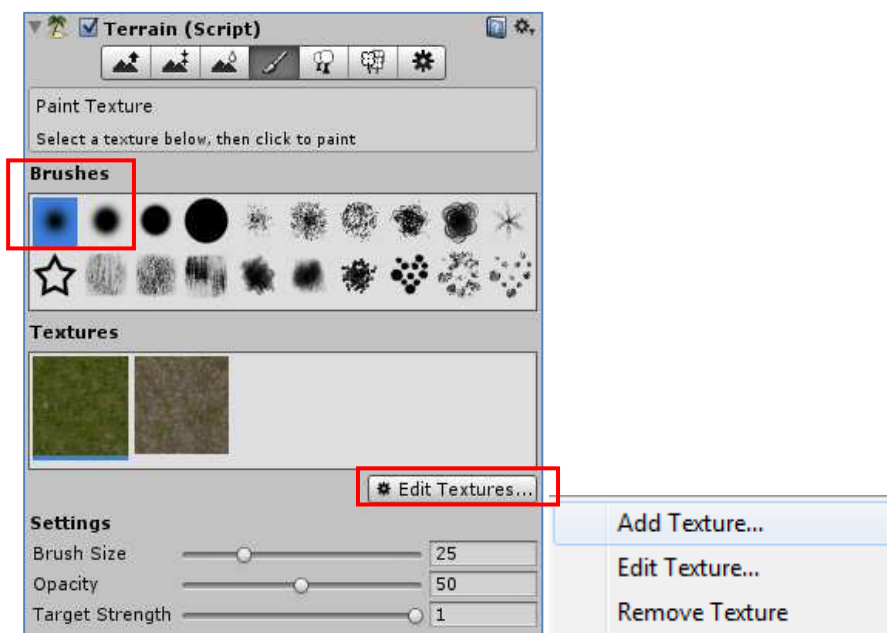
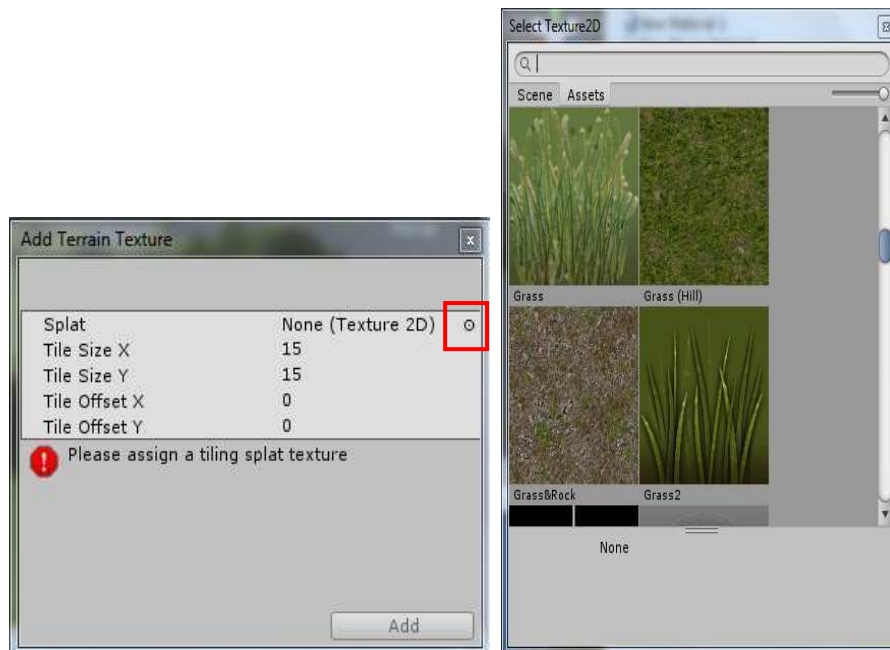


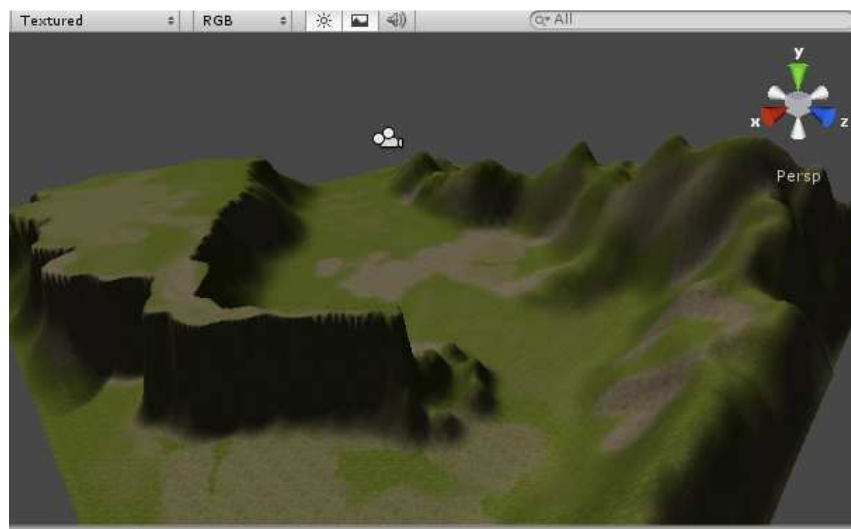
Ilustración 49: Texturizar el terreno

Aparece la ventana “Add Terrain Texture”, hacer clic sobre el círculo resaltado en rojo para obtener la ventana “Select Texture 2D”, se puede elegir una textura y aparecerá en el campo “Splat” y en el campo “Texture” en el inspector. Además también se puede elegir el ancho y largo de esa textura.



*Ilustración 50: Propiedades de la textura*

Una vez aplicados las texturas, estas aparecen en el terreno que tendrá el aspecto como sigue:



*Ilustración 51: Terreno texturizado*

Unity 3D ofrece un apoyo especial para introducir árboles. Se pueden añadir tantos arboles como se desea utilizando la quinta opción “Places tree”. Primero se hace uso de la opción “Edit Trees” para elegir el árbol que se quiere añadir en nuestro mundo, seguido se personaliza el árbol definiendo





varios parámetros, el tamaño del pincel con el que se va a colocar el árbol, la densidad de los arboles (determina la cantidad de árboles que se van a colocar), la variación del color (diferencia de color entre un árbol y otro), la altura y ancho del árbol.

Por último se hace clic en cualquier parte del terreno para añadir el árbol en el sitio elegido. Para borrar los árboles, se mantiene pulsada la tecla Shift y se hace clic en el terreno.

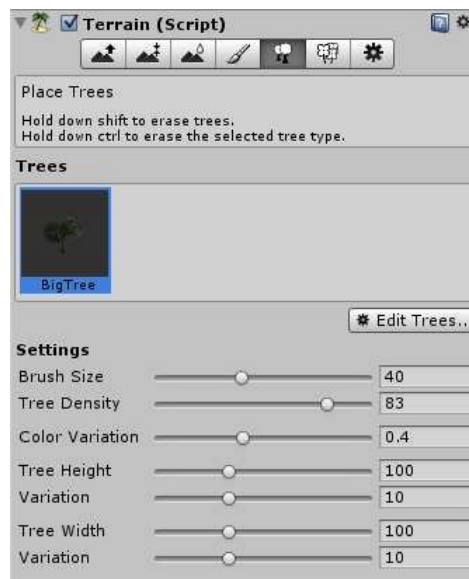


Ilustración 52: Añadir arboles al terreno 1.

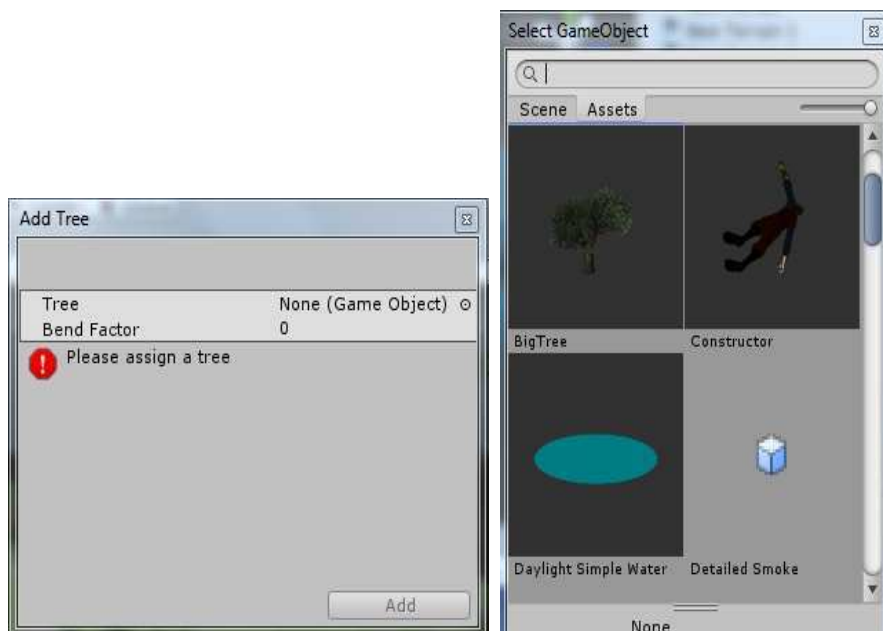


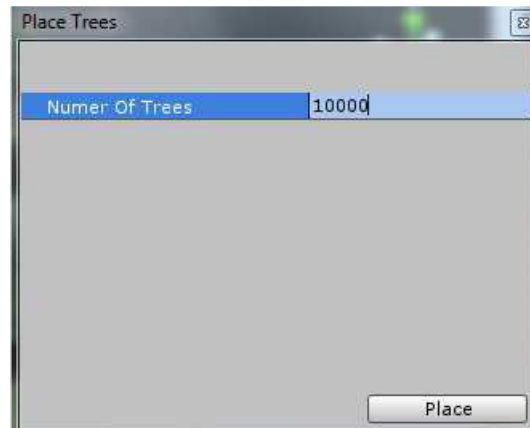
Ilustración 53: Añadir arboles al terreno 2

También se pueden añadir varios árboles a nuestro mundo de manera masiva y así crear un bosque de un solo golpe. Hacer clic en “Terrain->Mass

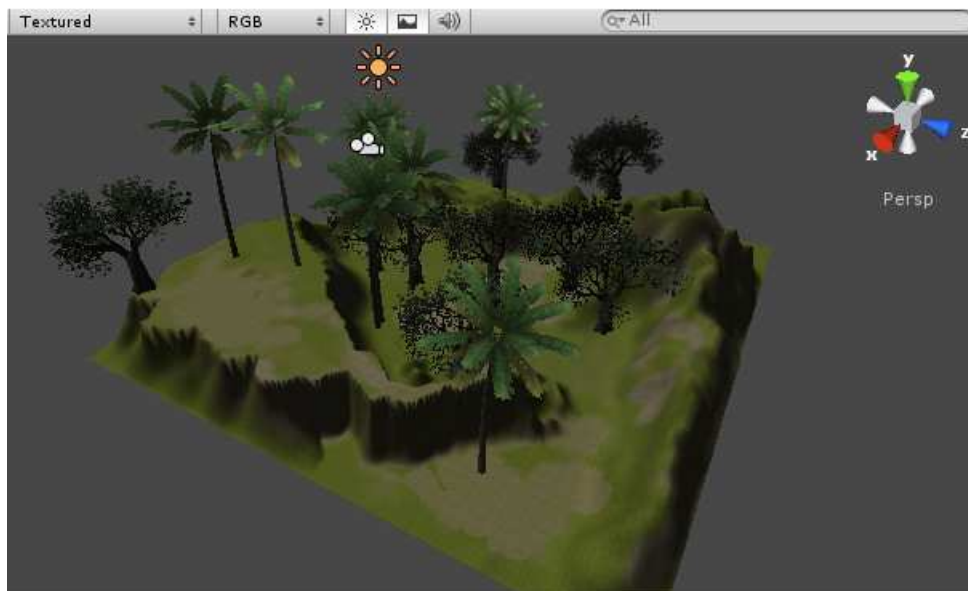




Place Trees” y aparecerá un cuadro de dialogo en el que se puede determinar el número de árboles que se quieren introducir en el juego.



*Ilustración 54: Añadir arboles al terreno 3*



*Ilustración 55: Terreno completado*

Se puede elegir un árbol accediendo a la carpeta “Standard assets > Tree Creator > Textures”, se selecciona uno de los árboles y se añade al terreno arrastrándolo.



## 9. CONCEPTO DE LIGHTMAP:

Unity integra la tecnología de Beast, un software de Autodesk que genera mapeados de iluminación de gran calidad.

El pop-up de “Create Lightmap” se utiliza para la creación de una lista de luces que permiten la iluminación de las texturas que definen la apariencia de la topografía del terreno.

Se accede a Windows>Lightmapping y se obtiene el pop-up que sigue:

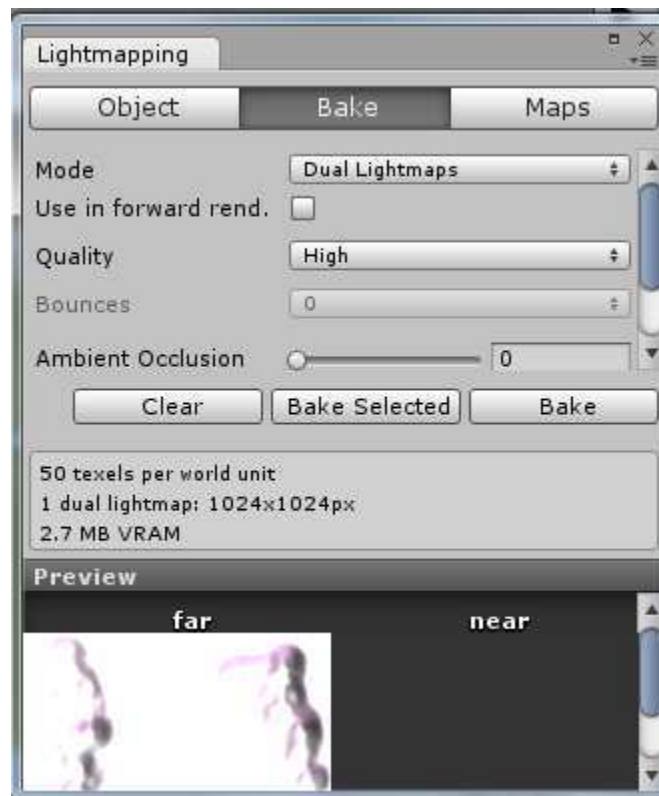


*Ilustración 56: Lightmapping*

Las mallas a tener en cuenta para el lightmapping deben tener UVs apropiados para ello. La manera más sencilla es usar la opción Generate Lightmap UVs en las opciones de importación de la malla.

En el panel Object del Lightmapping, marcar cualquier Mesh Render o Terrain como Static. Esto indicará a Unity que esos objetos no se moverán ni cambiarán por lo cual se les puede aplicar lightmapping.

Para controlar la resolución de los lightmaps, se ajusta el valor Resolution en el panel Bake. Puede ser útil activar en el lightmapping Display de la vista Scene la opción “Show Resolution”.



*Ilustración 57: Bake de lightmapping*

Al presionar Bake, el lightmap se aplica automáticamente. Hay que decir que el aspecto final de la escena depende mucho de la configuración de la iluminación y los ajustes del bake.

La opción del Lightmapping permite una mejor definición del efecto que debe tener la luz en el terreno y así en función de los elementos obtener un efecto u otro (por ejemplo una zona con sombras proyectados por la presencia de una montaña).

La sección Lights de la caja de dialogo permite aumentar el número de luces utilizadas para el rendimiento de la lightmap. Por ejemplo la escena puede ser iluminada por una luz direccional (apartado 7.4) que se comporta como la luz del sol. También se pueden insertar puntos de luz (Point Lights) para lámparas de exterior u hogueras (ejemplo más abajo, definición del fuego).

Cuando se crea una topografía con el editor de terrenos, los parámetros de la Lightmap a medida que el terreno evoluciona para obtener los efectos deseados.

Creando una Lightmap para las zonas claras y las zonas oscuras en el terreno, se reduce la potencia de cálculo necesaria a la hora de ejecutar el juego porque una parte de la iluminación ya está calculada. Mientras que los cálculos necesarios con la iluminación dinámica del terreno son más costosos.



### Explicación de los parámetros configurables:

#### Mesh Renderers y Terrains:

- Static: debe ser marcado como estático para poder usar lightmapping.
- Scale in Lightmap: solo para Mesh renderers, una valor alto supone mayor resolución dedicada al Mesh Renderer. Un valor nulo hará que el objeto no sea mapeado.
- Lightmap Size: tamaño del lightmap para este terreno.
- Atlas: información del Atlas que se actualiza automáticamente mientras está activado Lock Atlas.
- Lightmapping: modo de lightmapping, Realtime Only, Auto o Baked Only
- Color: color de la luz, se usa esta misma opción para el renderizado en tiempo real.
- Intensity: intensidad de la luz, se usa esta misma opción para el renderizado en tiempo real.
- Bounce Intensity: multiplicador de la intensidad de la luz indirecta emitida por una fuente en particular.
- Baked Shadow: comprueba si se emiten sombras desde los objetos iluminados por esa luz.

#### Bake:

- Mode: controla los modos de renderizado del lightmapping.
- Use in forward rendering: activa el modo dual para el forward rendering y requiere la creación de shaders propios.
- Quality: plantillas para la calidad ya sea alta o baja. Afecta al número de rayos, al contraste del umbral y algunos ajustes del anti-aliasing.
- Bounces: número de rebotes de luz en la simulación de iluminación global.
- Sky Light Color: La skylight imita la luz emitida desde el cielo en todas las direcciones, adecuada para iluminación exterior.
- Bounce Boost: potencia la iluminación indirecta, puede usarse para incrementar la cantidad de luz rebotada en la escena sin quemar el render demasiado rápido.

#### Maps:

- Compressed: activa la compresión en todos los lightmaps de la escena.
- Array size: tamaño de la matriz de lightmaps, de 0 a 254.
- Lightmaps array: matriz editable de los lightmaps de la escena actual.

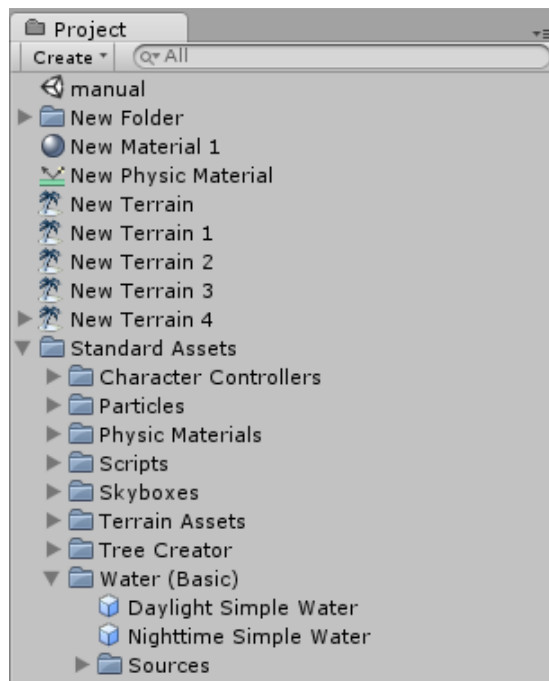


## 10. EL ENTORNO DEL TERRENO.

Los tutoriales de este apartado van a ayudar al usuario a personalizar el entorno en el que se encuentra el terreno por ejemplo añadir un cielo o agua alrededor del mundo.

### 10.1. Tutorial: Añadir agua a la escena:

Unity incluye varios prefabricados representativos del agua que el usuario puede utilizar directamente. Dos de estos prefabricados son la “Daylight Simple Water” y la “Nighttime Simple Water”.



*Ilustración 58: Daylight simple water*

En este caso se va incluir la “Daylight Simple Water” alrededor del terreno. Se accede a “Standard asset > Water”, se selecciona la “Daylight Simple Water” y se arrastra a la escena. Unity utiliza una malla ovalada para el agua y una vez incluida se puede manejar de la misma manera que cualquier otro objeto, se escala y se mueve para obtener este aspecto:

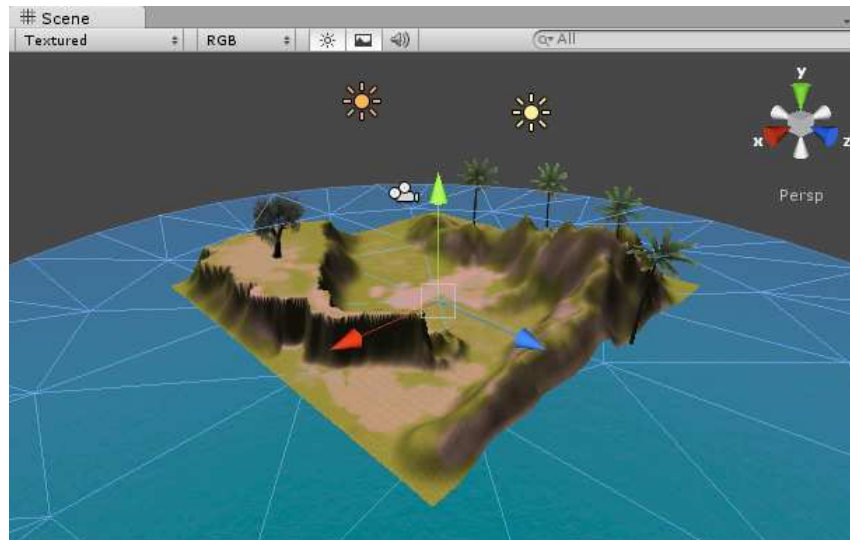


Ilustración 59: Añadir agua a la escena

Las características del agua se pueden ver en el inspector.

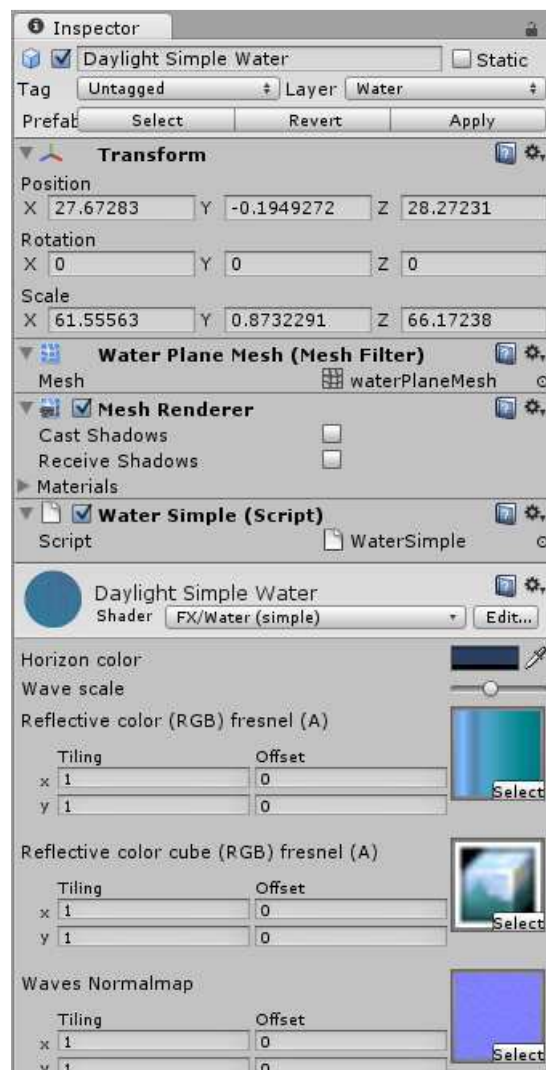
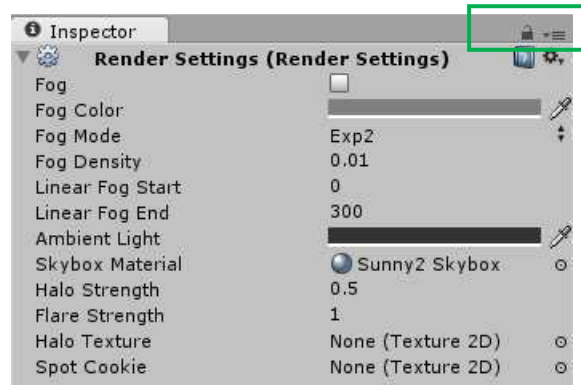


Ilustración 60: Características del agua

## 10.2. Tutorial: Añadir un cielo:

Se puede añadir un cielo de dos maneras, adjuntar un skybox a la cámara ó directamente a la escena. Se accede a “Edit > Render Settings” para que se activen los campos en el inspector como sigue:



*Ilustración 61: Añadir un skybox*

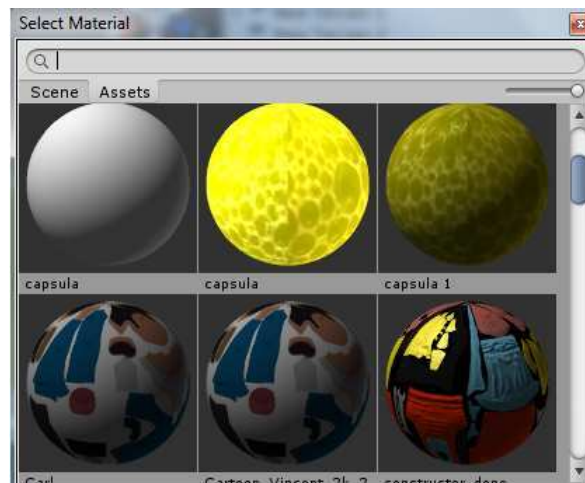
Este menú permite configurar el cielo de la escena, los efectos de niebla y algunos efectos globales de iluminación. Se pueden importar texturas propias o utilizar las que vienen por defecto en los assets de unity.

Para hacer que el menú Render Settings sea visible en la Vista de Inspector aunque se seleccione algún otro elemento de la escena, se accede a la esquina superior derecha donde hay un pequeño icono con forma de candado (icono resaltado en el cuadro verde). Haciendo clic en este candado se ancla el menú en la Vista de Inspector para que no desaparezca hasta que se vuelva a hacer clic sobre el candado.

Existen 2 opciones para aplicar el cielo a la escena:

- Opción 1: se accede al menú Render Settings, se hace clic en el pequeño icono en forma de disco que hay a la derecha del campo Skybox Material, se abrirá una nueva ventana con todos los recursos de materiales cargados en el proyecto.

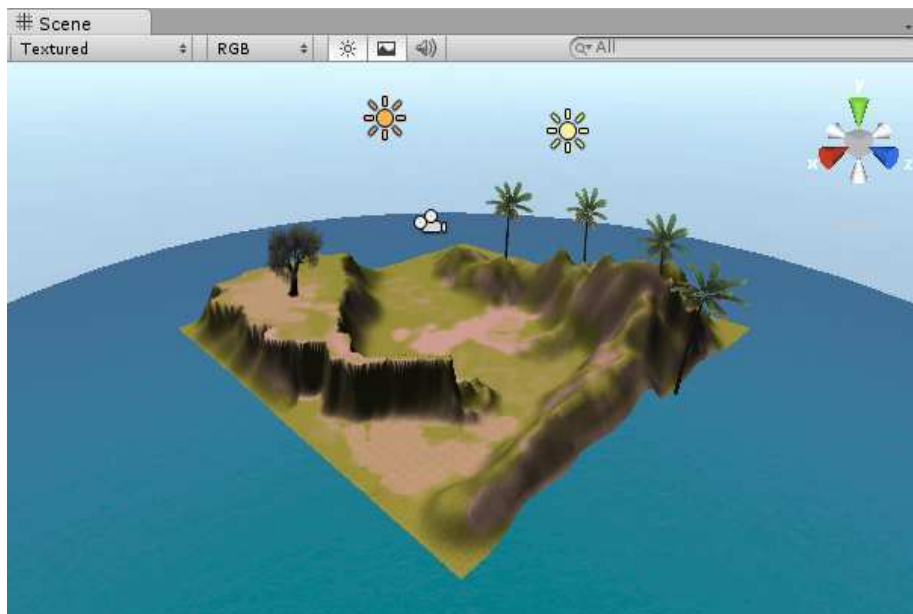




*Ilustración 62: Textura de cielo*

Tal como se puede observar en la ventana, se muestran todos los recursos de materiales cargados en el proyecto no sólo los de las cajas de cielo. Para que se nos muestren solamente las cajas de cielo, se va a la barra de búsqueda en la parte superior de la ventana de materiales y se escribe la palabra “sky”. Ya solo tenemos los materiales que nos interesan.

- Opción 2: en la vista de proyecto dentro de los “Standard Asset > Skyboxes”, se elige el cielo adecuado para nuestra escena y se arrastra a la “Sybox Material”. Para este resultado se ha elegido el cielo “Sunny 2 Skybox”.



*Ilustración 63: Escena con skybox*

Unity 3D ofrece varias opciones para personalizar el ambiente de la escena, estas





- Fog: permite añadir niebla a la escena, esta se vuelve brumosa.
- Fog Color: el color de la niebla, color por defecto el gris.
- Fog Mode: lineal, exponencial o exponencial al cuadrado. Esto controla la forma en que la niebla se desvanece.
- Fog Density: densidad de la niebla, solo se usa para el exponencial y exponencial al cuadrado. a. Una densidad mayor disminuirá la visibilidad y una densidad menor la aumentara.
- Linear Fog Start/End: inicio y fin de las distancias de la niebla, solo se utiliza si se activa la opción de niebla lineal.
- Ambient Light: por defecto es un gris. Si queremos dar a la escena un esquema diferente de iluminación, cambiar la luz ambiental es un buen principio.
- Skybox Material: la textura del cielo de la escena.
- Halo Strength: Tamaño de todos los halos de luz en relación con su rango.
- Flare Strength: Intensidad de todas las llamaradas en la escena.
- Halo Texture: referencia a una textura que aparecerá como el resplandor de todos los halos en las luces.
- Spot Cookie: La referencia a un Texture2D que aparecerá como la máscara de cookies para todos los focos.

### 10.3. Tutorial: Un FPC

Unity incluye un controlador en primera persona que facilita crear juegos con vista en primera persona. Accediendo a los “Standard Assets > Character Controllers”, se selecciona el “First Person Controller” y se arrastra a la posición donde se quiere colocar en la escena. En caso de tener un personaje importado se le aplica un Character Controllers accediendo a “Component > Physics > Character Controllers”.

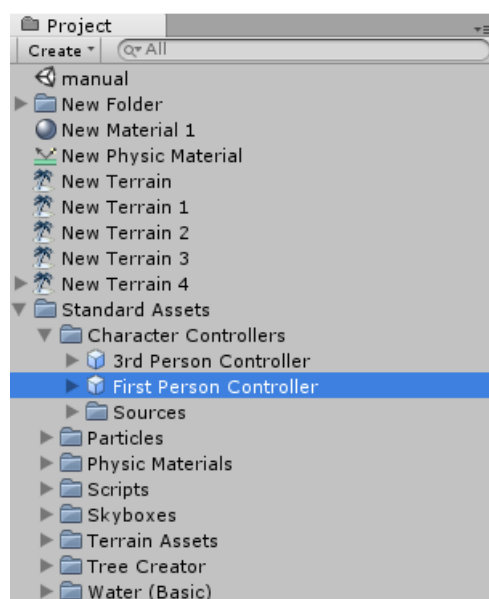


Ilustración 64: First person controller 1



Se puede observar que una capsula que tienen asociada una cámara se ha añadido a nuestra escena. Se puede lanzar el juego haciendo clic en el play para ver que somos un jugador en primera persona.

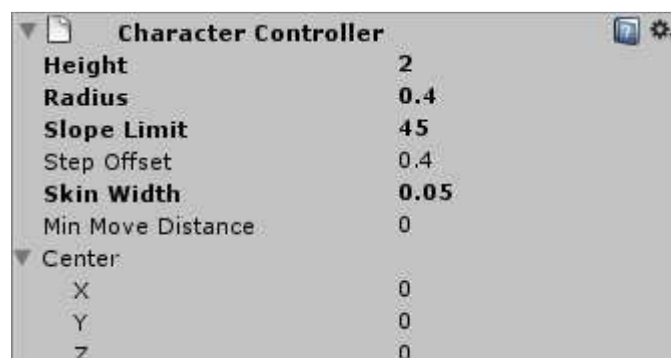
El manejo del personaje en la escena se hace mediante varias opciones:

- El movimiento del personaje: con las flechas o las teclas (W: adelante, A: izquierda, D: derecha y S: atrás).
- El control de la cámara se hace con el ratón.
- El salto con la tecla espacio.



*Ilustración 65: First person controller 2*

El “First Person Controller” se puede personalizar igual que los demás elemento en el inspector. Para ello se selecciona en la vista de jerarquía y aparecen en el inspector sus características.



*Ilustración 66: Character Controller*

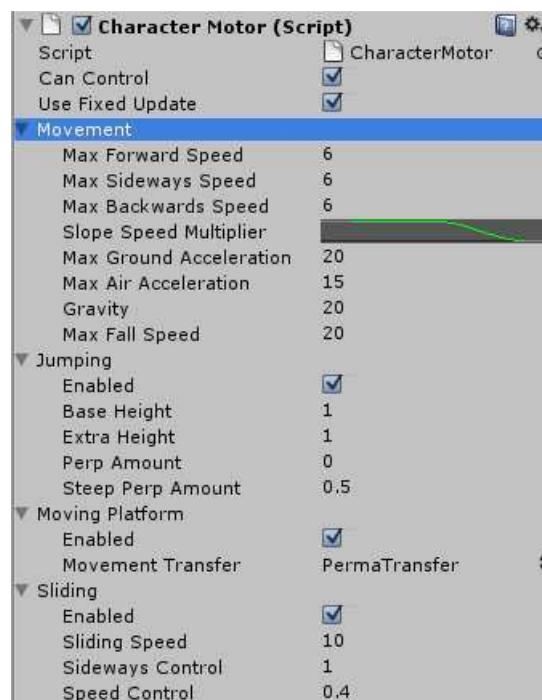


Explicación de los parámetros configurables:

- Height: altura del controlador
- Radius: radio del controlador
- Slope limit: valor máximo de la pendiente que el controlador puede subir.
- Step offset: este valor está comprendido entre 0,1 y 0,4 para un ser humano de 2 metros de tamaño.
- Skin width: permite controlar los colliders del personaje para que no quede bloqueado.
- Min move distance: Si el personaje intenta moverse por debajo del valor indicado, no se mueve en absoluto. Esto se puede utilizar para reducir las vibraciones. En la mayoría de situaciones de este valor se debe dejar en 0.
- Center: la posición del controlador, sus coordenadas: x, y, z.

En esta versión de Unity 3D el Character Controllers se añade junto con 2 scripts bastante complejos implementados en javascripts que facilitan el trabajo al usuario evitándolo implementar algunas funciones:

- El “Character Motor” que permite controlar los datos de movimiento en “Movement”. Datos tales como la velocidad de los movimientos hacia adelante y hacia atrás, la gravedad del personaje o la velocidad de caída. En este apartado también se puede controlar las opciones de “Jumping” o sea ser de los saltos del personaje y de “Sliding” que se refiere a los movimientos hacia los lados.



*Ilustración 67: Character motor*



- El “FPS Input Controller”



Ilustración 68: FPSInput controller

## 10.4. Tutorial: Crear un gusano

Se va a proceder a crear un personaje sencillo, un gusano a base de esferas. Se crea la primera esfera en “GameObject > Create Other > Sphere”:

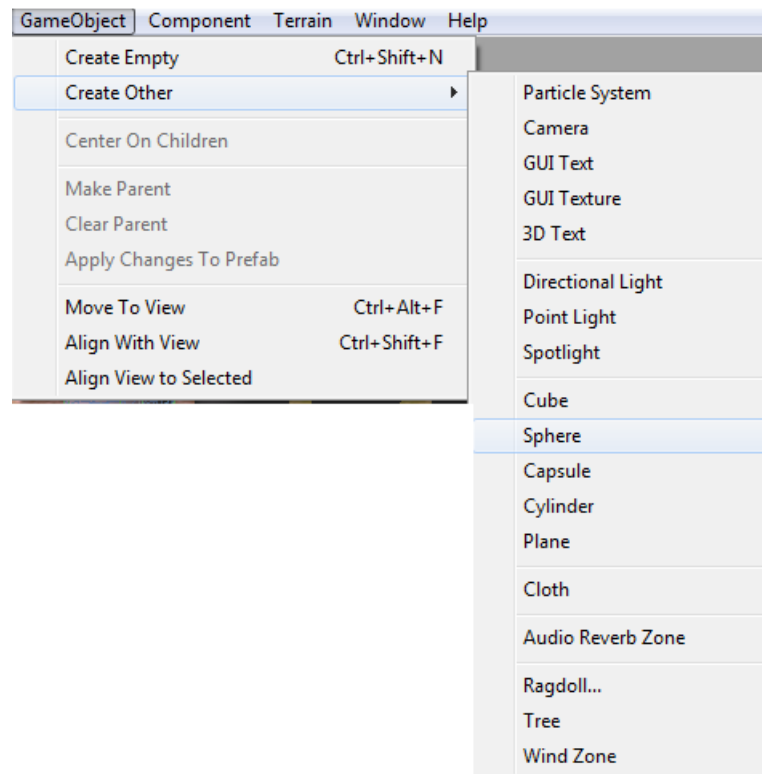


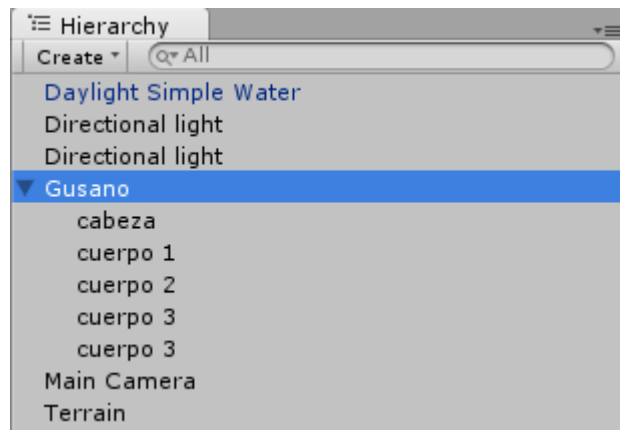
Ilustración 69: Creación de personaje sencillo

Se coloca la esfera en el terreno en la posición que queramos. Para crear otras esferas idénticas se hace “CTRL + D”, esta nueva esfera aparece sobre la antigua y ya se puede desplazar. Se repite la misma acción hasta obtener 4 esferas. Se puede observar que las esferas aparecen en la vista de jerarquía donde se puede renombrar la primera esfera como “cabeza” y las demás en “cuerpo n°”.

Para renombrar un GameObject se selecciona este en la vista de jerarquía, se hace clic en botón derecho y “Rename”. Se modifican las esferas que representan el cuerpo haciéndolas más pequeñas utilizando el inspector para más precisión.



Además para optimizar la organización de nuestro proyecto se crea un objeto vacío en “GameObject > Create Empty”, se renombra como gusano y se arrastran la cabeza y el cuerpo al gusano.



*Ilustración 70: EL personaje gusano 1*

Seguido se importa una textura, se aplica al personaje y se obtiene el siguiente resultado:



*Ilustración 71: El personaje gusano 2*

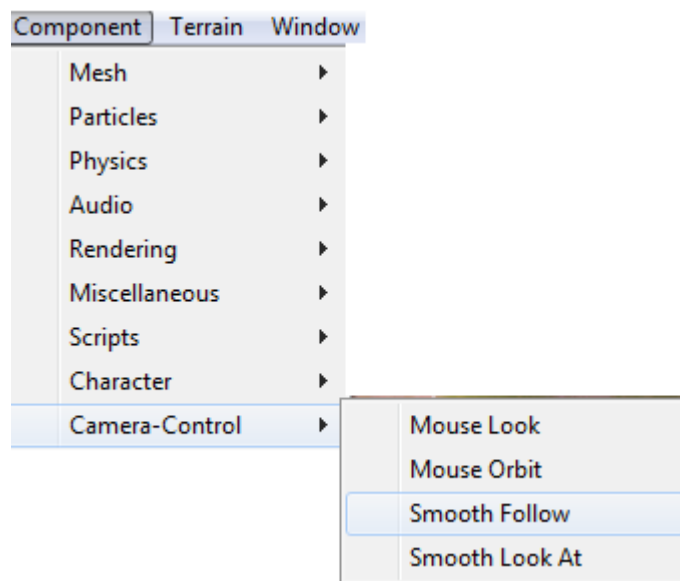
Por último para hacer que todos los elementos que conforman el cuerpo del gusano sigan a la cabeza se va a utilizar el componente “Smooth follow” que se va a explicar en el siguiente apartado.



## 10.5. Tutorial: La cámara sigue el personaje

Se va a utilizar el componente Smooth Follow para hacer que la cámara siga constantemente a la cabeza de nuestro personaje. Del mismo modo se va usar este componente para hacer que las esferas que componen el gusano se sigan las unas a las otras.

Se selecciona la cámara en la jerarquía “Main Camera” y se accede a Component > Camera-Control > Smooth Follow:



*Ilustración 72: Smooth follow 1*

Aparecerá un nuevo componente en el inspector con varias variables a configurar.

- Target: es el objetivo a seguir en este caso la cabeza. Se selecciona la cabeza en la jerarquía y se arrastra hasta “Target”.
- Distance: es la distancia de la cámara hasta el objetivo. En este caso la distancia hasta la cabeza que se va a fijar a 10 metros.
- Height: la altura de la cámara.
- Los últimos parámetros “Height Damping y Rotation Damping” permiten amortiguar los movimientos de la cámara, se dejan por defecto.

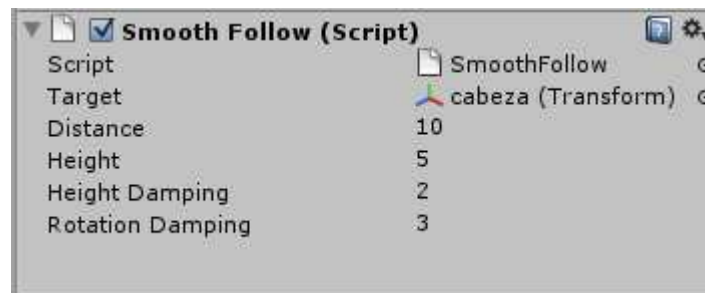


Ilustración 73: Smooth follow 2

Se guardan los cambios y se lanza el juego para probarlo. Se puede observar que la cámara sigue la cabeza.

De la misma manera se usara este componente para hacer que las esferas que componen el gusano se sigan. La primera esfera seguirá la cabeza, la segunda esfera seguirá la primera y así.

## 10.6. Tutorial: Movilidad del gusano

Se selecciona la cabeza en la vista de jerarquía, se accede a Component> Scripts> Third Person Controller. De esta manera se puede dar movilidad a la cabeza y manejarla en tercera persona. En caso de querer mover toda la serpiente habría que aplicar el “Third Person Controller” a toda la serpiente.

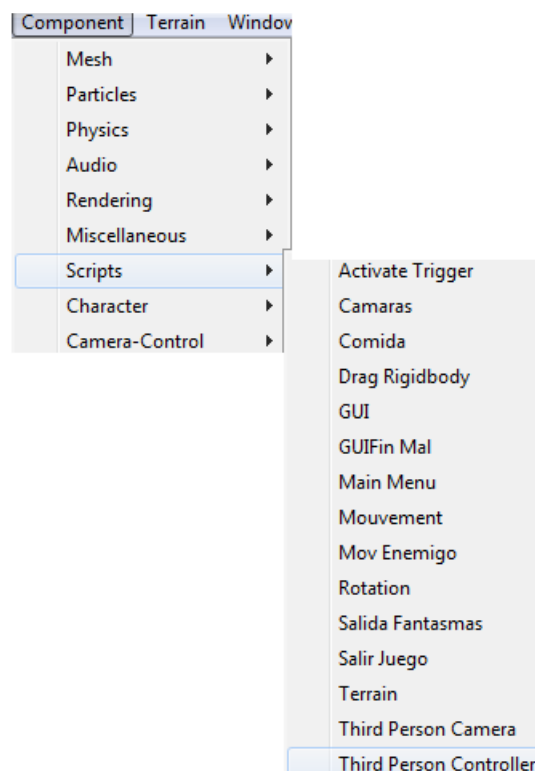
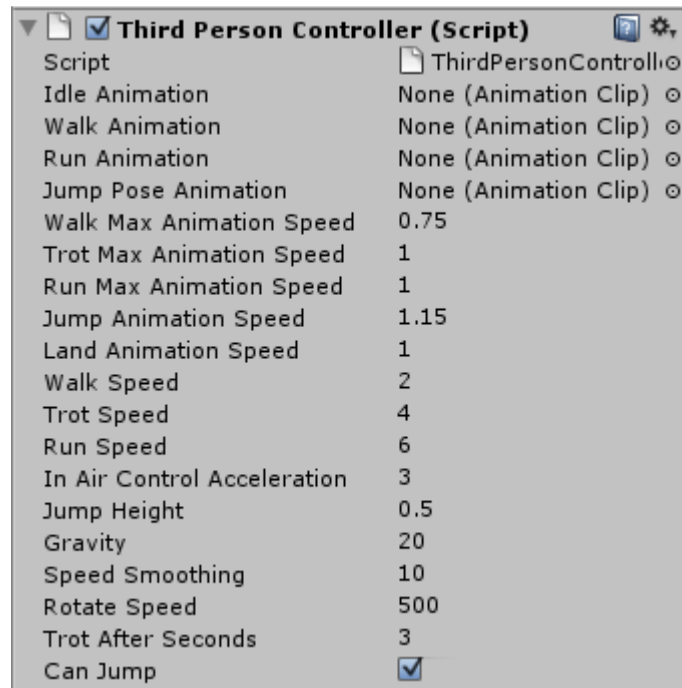


Ilustración 74: Movilidad del personaje





En la vista del inspector aparece el nuevo componente con todos los parámetros configurable:



*Ilustración 75: Características de la movilidad*

## 10.7. Tutorial: Crear capsulas

Acceder a GameObject>Create Other>Capsule de esta manera aparece la capsula en nuestro mundo y se modifica la escala de las capsulas en la vista de inspector para hacerlas más pequeñas. A continuación se aplica a las capsulas una textura.



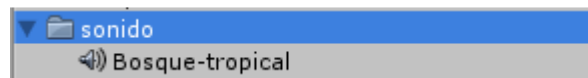
*Ilustración 76: Creación de capsulas*



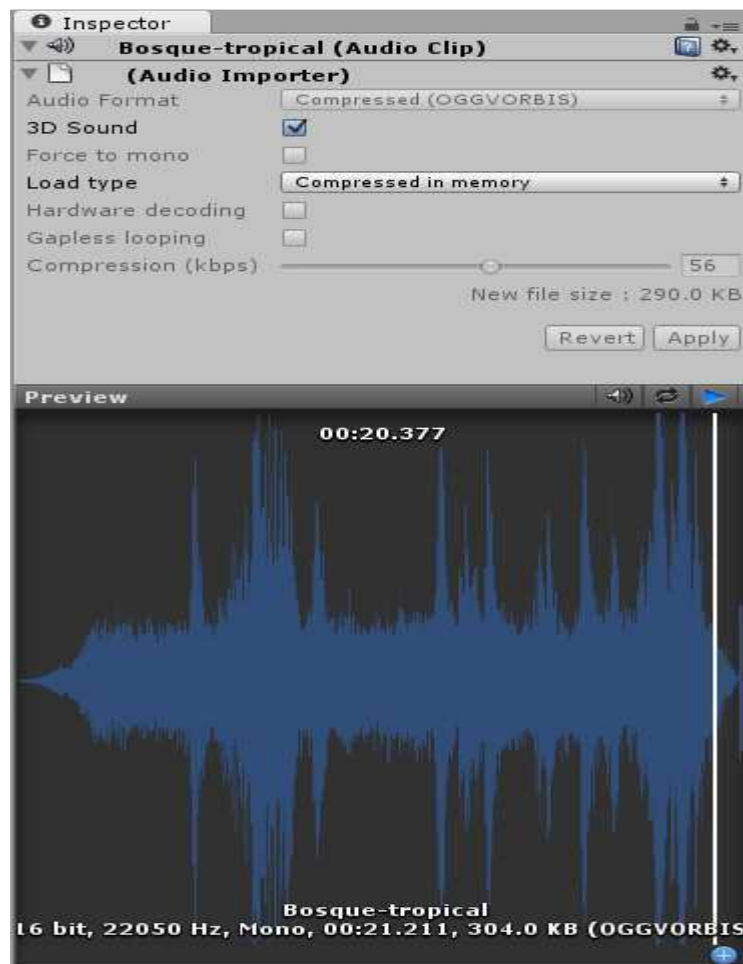
## 10.8. Tutorial: Sonido en la escena

En unity las fuentes de audio permiten insertar sonido en el juego. Existen 2 tipos de audio diferentes audio 3D o audio 2D. El primero se escucha en una posición y se va atenuando según nos alejamos de esa posición. El segundo se oye de manera uniforme en toda la escena.

Se crea una carpeta que se nombra “sonido” y en la cual se guarda un sonido importado que representa los sonidos ambientales de una selva.

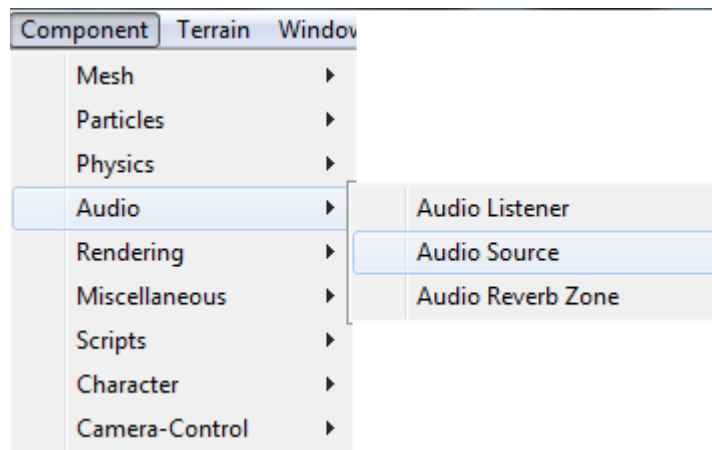


*Ilustración 77: Importación de sonido*



*Ilustración 78: Características del sonido*

Se selecciona la “Main Camara” y se accede a Component> Audio> Audio Source.



*Ilustración 79: Componente audio source*

Seguido se arrastra el sonido importado a la casilla “Audio Clip” en la vista del inspector y se procede a configurar el audio:

- Se activa la casilla “Play On Awake” para que el sonido empiece con el comienzo del juego.
- Se activa la casilla “Loop” para que la música se ejecute en bucle y se repita las veces que haga falta.
- En el grafico del “Listner” que aparece en el componente, se selecciona el punto verde que aparece en el punto más alto y se arrastra hasta el centro de coordenadas (0,0).
- Se define una distancia mínima de 0,1 en “Min Distance”.

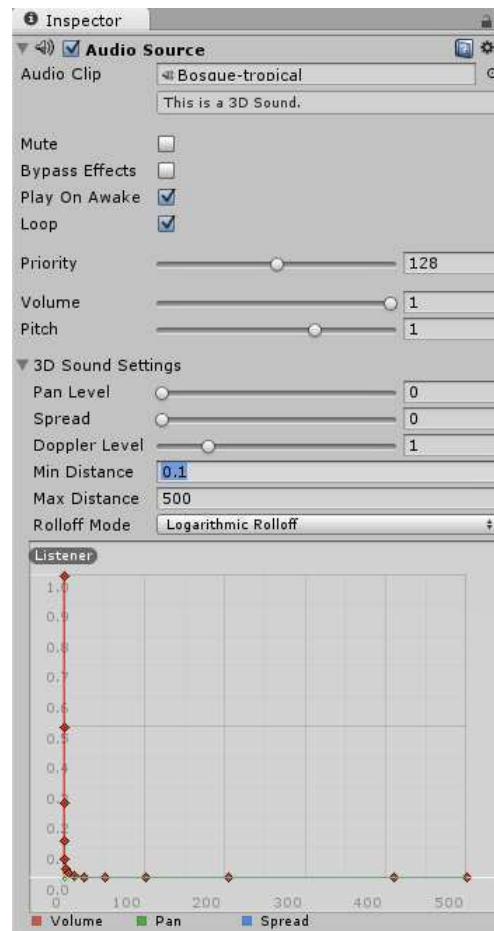


Ilustración 80: Configuración del audio source

Nuestra cámara ya tiene 2 componentes, el “Audio Source” para reproducir sonido y el “Audio Listener” para poder escuchar ese sonido. Se puede lanzar el juego y comprobar que dispone de una música ambiental.

## 10.9. Tutorial: Un contador

En el script anterior, se va a declarar una variable que se llame “score” de la siguiente manera “`var score = 0;`”. Cada vez que el personaje se coma una capsula se incrementara esta variable en 1. “`score += 1;`”. De esta manera aparece esta variable en la vista del inspector y cuando se lanza el juego a medida que el personaje come capsulas se va incrementando ese contador. Al terminar la partida se reinicia el contador y se pone a 0.

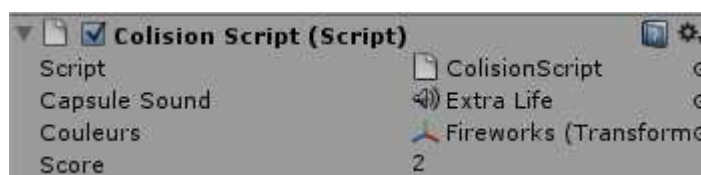


Ilustración 81: Contador del score



## 10.10. Tutorial: Encender/apagar la luz

Este tutorial nos va a permitir encender y apagar la luz haciendo clic derecho el ratón. Primero se crea un GameObject vacío y se procede a redactar el script que sigue:

```
var myLight : Light;  
var myLight1 : Light;  
  
- function Start(){  
    myLight = Light.FindObjectOfType(typeof(Light));  
}  
  
- function Update () {  
-     if(Input.GetButtonDown("Fire1")){  
        myLight.enabled = !myLight.enabled;  
        myLight1.enabled = !myLight.enabled;  
    }  
}
```

Explicación del código:

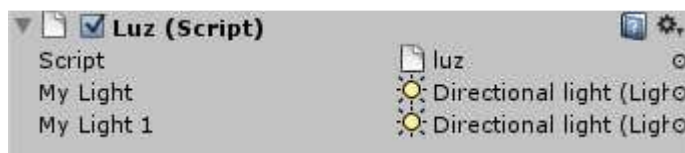
`myLight = Light.FindObjectOfType(typeof(Light));` → busca en la escena el objeto de tipo “light” (la luz de la escena).

`if(Input.GetButtonDown("Fire1"))` → cuando se pulsa el botón izquierdo del ratón.

`myLight.enabled = !myLight.enabled;` → cambia el estado de la luz, si está encendida, se apaga y viceversa.

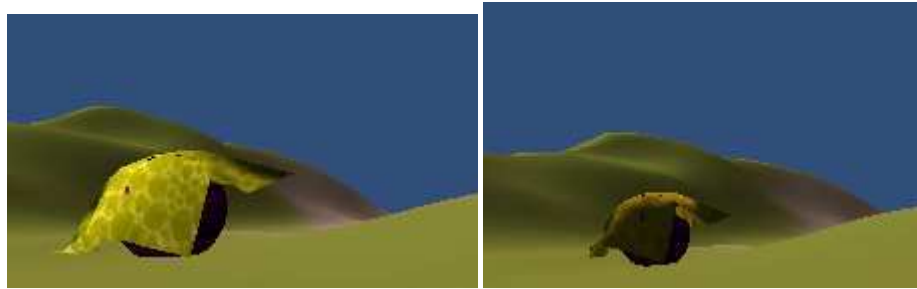
`myLight1.enabled = !myLight.enabled` → esta línea se duplica porque en nuestra escena existen 2 luces.

Una vez creado el script se adjudica al objeto vacío y en los parámetros que aparecen en la vista de inspector se arrastra cada una de las 2 luces para que quede de la siguiente manera:



*Ilustración 82: Script de luz*

Se lanza el juego y se prueba hacer clic en el botón izquierdo del ratón 2 veces seguidas y se puede ver que se van apagando las luces:



*Ilustración 83: Encendido y apagado de luz*

Igual que se usa el ratón, se puede usar cualquier tecla que Elija el usuario simplemente tiene que sustituir una línea de código por otra. Entre comillas se pone la tecla que se quiere usar como detonador, en el ejemplo se ha usado la tecla A.

`Input.GetButtonDown("Fire1")` por `Input.GetKeyDown("A")`



## 11. EL PRIMER SCRIPT:

Los scripts permiten la definición de la lógica del juego. A lo largo del tutorial se van a ver varios script sencillos como por ejemplo para definir en este caso el movimiento de rotación de las capsulas, para definir el comportamiento de unos enemigos (comportamiento de los fantasmas del Pacman), personalización sencilla de la interfaz del juego o para definir los movimientos de un personaje.

El funcionamiento es muy sencillo, primero se define el comportamiento deseado en un script y este se agrega como un componente al objeto para el cual queremos asignar ese comportamiento, incluso se puede agregar un script a un objeto vacío.

Existen 3 funciones básicas para los scripts:

- *Función Update( )* : función que define el bucle principal del juego por lo cual se ejecuta una vez por cada frame que renderize el dispositivo.
- *Función Awake( )* : Las variables definidas dentro de esta función se inicializarán justo antes de que empiece el juego. Esta función se ejecutará una sola vez durante el tiempo que dure la instancia del script.
- *Función Start( )* : Esta función es muy parecida a “Awake( )” pero tiene una diferencia vital; solamente se ejecutará si la instancia del script está habilitada. Esto nos permitirá retardar la inicialización de algunas variables de estado del juego. También debemos tener presente que la función “Start( )” siempre se ejecutará después de que haya terminado la función “Awake( )”.

Además Unity 3D dispone de un editor de scripts por defecto que se llama uniscite.

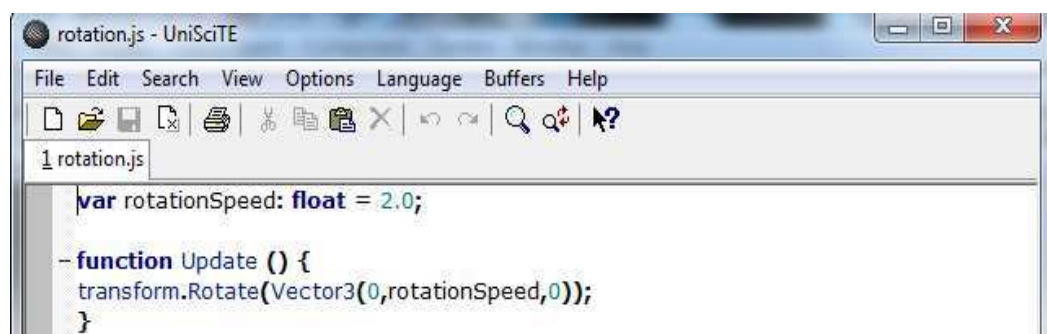


Ilustración 84: Editor de script uniscite



## 11.1. Tutorial: Script de rotación

Vamos a crear un script muy sencillo para hacer que las capsulas roten. En la parte superior izquierda de la vista de proyecto, haciendo clic en “Create” se despliega una lista en la que aparecen 3 opciones para la creación de scripts, C#, Boo o JavaScript. En nuestro caso vamos a usar JavaScript.

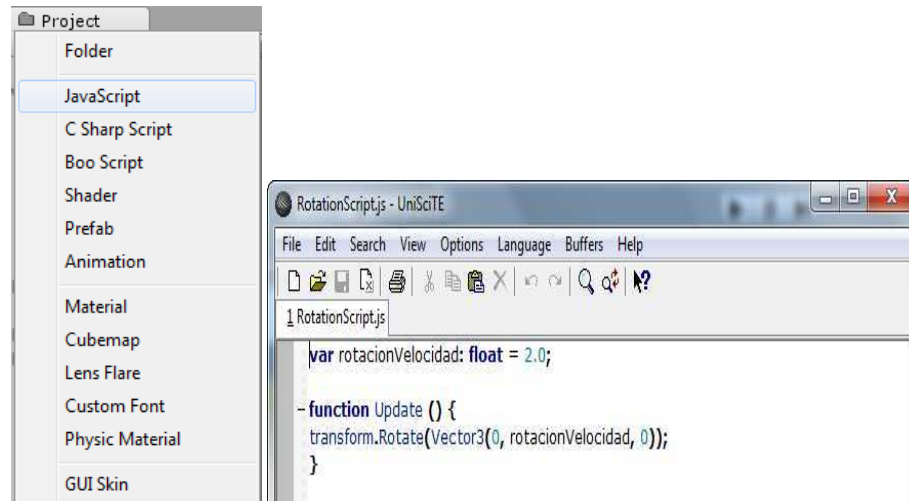


Ilustración 85: Creación de un javascript 1

### Explicación del código:

`var rotacionVelocidad: float = 2.0;` → “var” para declarar una variable que se llama “rotacionVelocidad” y de tipo “float” y es igual a “2.0”.

`transform.Rotate(Vector3(0, rotacionVelocidad, 0));` → “transform” porque se va a tocar un parámetro de la vista de inspector en “transform”.

→ Este parámetro será de rotación por lo cual “transform.Rotate”.

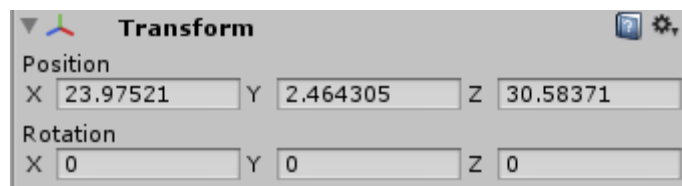
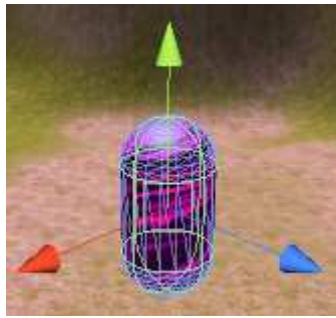


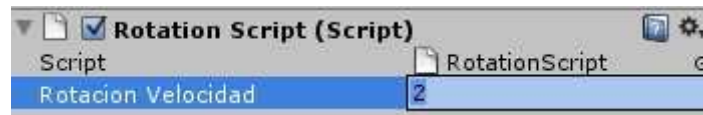
Ilustración 86: Creación de un javascript 2

→ “Vector3” así se llama a los 3 ejes que nos permiten movernos en un entorno 3D.




*Ilustración 87: Rotación de una capsula*

Se guardan los cambios para compilar este código y se arrastra el script para asociarlo a la capsula. La ventaja de declarar una variable de velocidad de rotación es el poder modificarla rápidamente desde la vista de inspector:



*Ilustración 88: velocidad de rotación*

Seguido se ejecuta el juego haciendo clic en  para ver como rota la capsula.





## 12. PREFAB:

Los prefab son objetos reutilizables, el desarrollador crea un objeto con una serie de características y lo define como un prefab en la vista de proyecto para después poder instanciar ese objeto tantas veces como lo desee y en cualquier escena del videojuego.

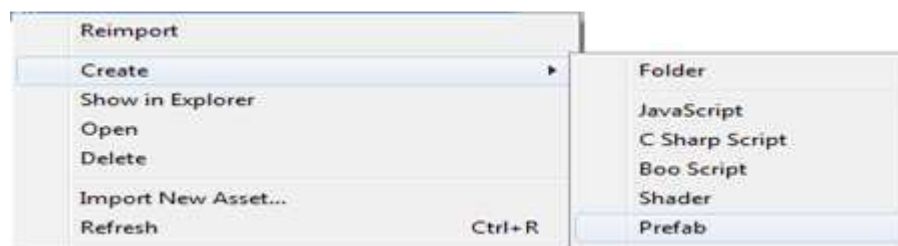
La creación de los prefab presenta diversas ventajas entre las cuales se pueden resaltar las siguientes:

- Cuando se instancia un prefab en cualquier sitio de la escena y se realizan modificaciones de sus propiedades, estas últimas se afectan a todas las demás instancias del prefab, no hay necesidad de modificar los elementos uno a uno.
- Los prefab también permiten la modificación de los atributos de un elemento sin que este cambio impacte a las demás instancias del prefab y sin necesidad de romper la conexión del prefab con esa instancia.
- En la vista de jerarquía el nombre de los objetos conectados a algún prefab se resalta en color azul.

### 12.1. Tutorial: Crear un prefabricado

A continuación se explica cómo se lleva a cabo la creación de los prefabricados en Unity 3D:

En la vista de proyecto se va a crear una carpeta llamada “Prefab”, situarse encima haciendo clic en el botón derecho: Create> Prefab.



*Ilustración 89: El prefab capsula*

Una vez que se le ha asignado a la capsula todas las características, se arrastra la capsula para asignarla al “Prefab” y se nombra.



*Ilustración 90: El prefab capsula*

Se borran las capsulas existentes en el juego y cada vez que se quiere añadir capsulas al mundo se usa el prefab. Se pueden añadir tantas capsulas



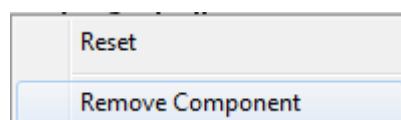
como se quiera y se puede observar que estas ya vienen definidas con todas las características del prefabricado (textura, script de rotación etc....).



*Ilustración 91: Utilización del prefab de capsulas*

## 12.2. Tutorial: Movimientos del personaje:

En unity 3D ya existe un script predefinido para el movimiento de los personajes pero la creación de uno propio permite un mejor conocimiento y manejo de las funcionalidades que lo componen. Se selecciona la cabeza del gusano para borrar el componente de movimiento aplicado anteriormente. En la vista de inspector se hace clic en el botón derecho y “RemoveComponent” del “Third Person Controller” y en el “Character Controller”.



*Ilustración 92: Remove Component*

Mantenemos el componente de “Smooth Follow” que nos permite seguir el personaje. Se crea un nuevo JavaScript y lo nombramos “mouvementScript”. El código del script es el siguiente:

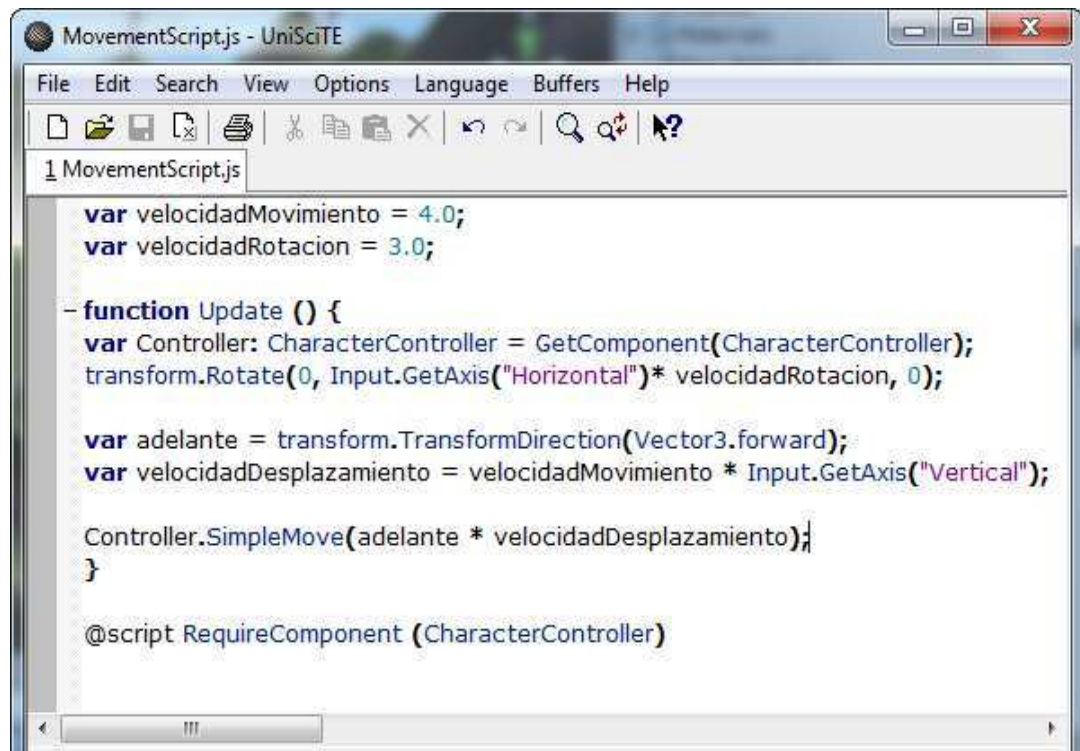


Ilustración 93: Script de movimiento

#### Explicación del código:

`var velocidadMovimiento = 4.0; var velocidadRotacion = 3.0;` → Las variables que se van a usar y que se pueden modificar fácilmente desde la vista de inspector.

`var Controller: CharacterController = GetComponent(CharacterController);` → variable que representa el componente CharacterController visto anteriormente y ya predefinido en Unity. Este componente es el que nos va a permitir mover el personaje.

`transform.Rotate(0, Input.GetAxis("Horizontal")* velocidadRotacion, 0);` → transform.Rotate igual que en el caso anterior porque se va a tocar esa parte de la vista de inspector. `Input.GetAxis("Horizontal")` para poder recuperar cuando el usuario teclee las flechas horizontales (movimiento izquierda, derecha). Esto significa que cuando el usuario teclee las flechas derecha o izquierda se va a modificar la rotación en el eje “y” mientras que en los ejes “X” e “Z” será nula.

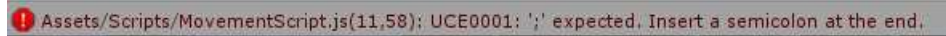
`var adelante = transform.TransformDirection(Vector3.forward);`  
`var velocidadDesplazamiento = velocidadMovimiento * Input.GetAxis("Vertical");` → Se declaran 2 nuevas variables para poder desplazar el personaje adelante o atrás.

`Controller.SimpleMove(adelante * velocidadDesplazamiento);` → permite mover el personaje.



`@script RequireComponent (CharacterController)` → definir una dependencia, el script necesita un componente definido en Unity que es el CharacterController.

Una vez creado el script, se compila y en caso de error el compilador de Unity lo muestra en la parte inferior izquierda. En el ejemplo que sigue, se puede ver el error que devuelve unity cuando falta un “;” al final de una sentencia en un script:

A screenshot of a Unity console error message. It shows a red error icon followed by the text: "Assets/Scripts/MovementScript.js(11,58): UCE0001: ';' expected. Insert a semicolon at the end,". The message is displayed on a dark background with a light border.

Assets/Scripts/MovementScript.js(11,58): UCE0001: ';' expected. Insert a semicolon at the end,

*Ilustración 94: Mensaje de error de compilación*

Cuando el script compila correctamente se puede asignar al gusano, guardar los cambios, ejecutar el juego y ya se puede ver como se mueve el gusano.



## 13. LOS COLLIDERS:

Los colliders son los componentes que permiten gestionar las colisiones entre elementos en unity 3D. Es una malla invisible alrededor del objeto encargado de detectar el contacto con otros objetos. La detección de colisiones agrupa tanto la detección como la recuperación de la información precedente de estas colisiones.

Cuando varios objetos entran en colisión en cualquier motor de juego, se dispone entonces de información relativa a ese evento. Consiguiendo información variada en el momento del impacto, el motor puede reaccionar de manera realista. En un juego que tiene en cuenta las leyes de la física por ejemplo, si un objeto se cae al suelo desde cierta altura, el motor necesita saber que parte del objeto tocara el suelo primero para controlar de manera realista la reacción del objeto a ese impacto.

Unity gestiona este tipo de colisiones y almacena información de manera a que el usuario solo tiene que recuperarla para definir la reacción que se tiene que producir. Los colliders se dividen en dos categorías, los estáticos y los dinámicos.

Los colliders estáticos permiten configurar las colisiones de los elementos (el personaje por ejemplo) del juego con los objetos estáticos de esta (los arboles por ejemplo). Existen los tipos de colliders estáticos que siguen:

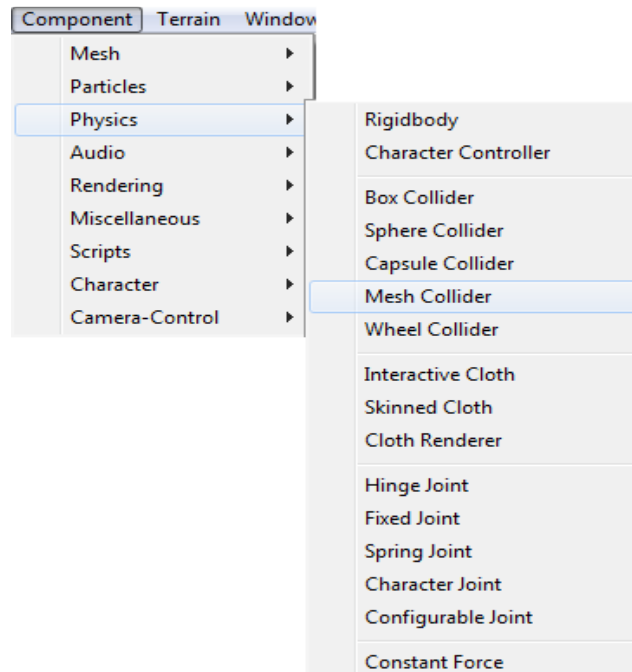
- Box Collider: Tiene la forma de cubo.
- Sphere Collider: en forma de esfera.
- Capsule Collider: este tiene forma de capsula.
- Mesh Collider: permite crear un collider con la misma forma que el objeto que va a englobar.
- Wheel Collider: este collider es muy específico y permite tratar las colisiones de las ruedas de los vehículos con el pavimento.
- Los colliders dinámicos se componen básicamente del “Character Controller colliders” y los “Rigidbodyes”.

El Character Controller colliders es exclusivamente para los personajes humanoides, les añade un sistema de colisiones que permite que el personaje se mueva por la escena y que inhibe el efecto del motor de físicas.

Los Rigidbodyes permiten que los elementos se vean afectados por el motor de físicas lo cual hace que el movimiento del objeto sea más realista.

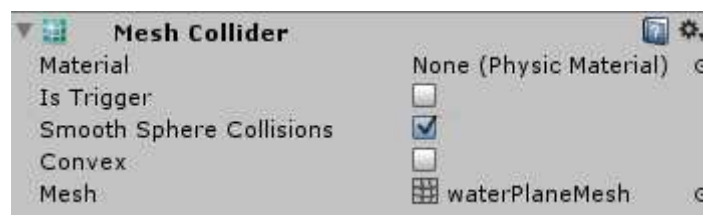
### 13.1. Tutorial: Objeto de colisión

Unity dispone de un sistema para interactuar con los elementos de manera muy sencilla. Se selecciona el agua creada anteriormente en el juego y se hace clic en Component>Physics>Mesh Collider. De esta manera el personaje puede colisionar con el agua. Si el gusano entra en colisión con el agua no podrá atravesarla.



*Ilustración 95: Mesh collider 1*

Si se hace clic en “Is Trigger” el agua sigue siendo un elemento de colisión pero se puede atravesar ya no es sólida.



*Ilustración 96: Mesh collider 2*

Haremos lo mismo con las capsulas que ya tienen un “Capsule Collider”, se activa la casilla “Is trigger”.



## 13.2. Tutorial: Recogida de las capsulas

Primero se importa un sonido para que suene al recoger las capsulas.

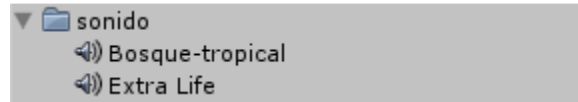


Ilustración 97: Inserción de sonido

A continuación, se procede a crear un script de colisión para hacer que el personaje pueda comerse las capsulas. En la vista de proyecto, botón derecho y se hace clic en javascript. Se escribe el código que sigue:

`var capsuleSound : AudioClip;` → Variable en la que se guarda el sonido que va sonar al comer las capsulas.

`var couleurs : Transform;` → Esta variable va permitir producir un efecto de explosión de colores al recoger la capsula.

`function OnTriggerEnter(objectInfo : Collider){` → Declaración de una función con un “objectInfo” que va a recuperar el nombre del objeto con el cual el personaje colisiona.

`if(objectInfo.gameObject.tag == "caps")` → si el objeto tocado tiene un tag “caps” (apartado anterior)

{

`audio.PlayOneShot(capsuleSound);` → se lanza el sonido cuyo nombre está indicado entre paréntesis para que suene una vez.

`Instantiate(couleurs, transform.position, transform.rotation);` → se instancia el objeto en la posiciones y rotación de la capsula que se ha tocado.

`Destroy(objectInfo.gameObject);` → Se destruye la capsula que ha sido tocada.

}

}

Se selecciona la cabeza del personaje y se le asigna el script de colisión entonces aparece el script en la vista del inspector.

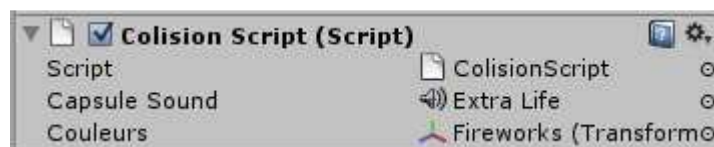
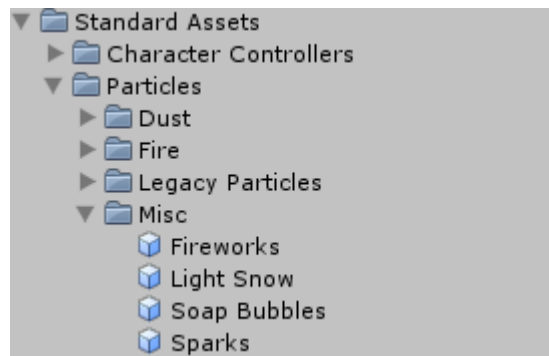


Ilustración 98: Añadir el script de colisión

Se arrastra el sonido anteriormente importado y se coloca en el campo “Capsule Sound”. Se accede a Standard Assets>Particules>Misc>Fireworks y se arrastra al campo “Couleurs”.





*Ilustración 99: Efecto explosión de colores*

Se lanza el juego para comprobar el funcionamiento y se puede ver como aparece en la imagen que el gusano se mueve por el terreno y cuando alcanza una capsula se la come y aparece una explosión de colores que desaparece poco a poco.



*Ilustración 100: Recogida de las capsulas*





## 14. SISTEMA DE PARTÍCULAS:

Los sistemas de partículas en Unity 3D se tratan como objetos. Estos sistemas están compuestos a su vez por 3 objetos:

- *Particle Emitter*: Sirve para generar las partículas.
- *Particle Animator*: Se utiliza para mover las partículas en el tiempo.
- *Particle Renderer*: Dibuja las partículas en pantalla.

En unity 3D existen varios tipos de partículas de los cuales se van a citar los más importantes:

- *Ellipsoid Particle Emitter*: Este sistema coloca partículas dentro de una esfera. Este es el sistema que aparece por defecto en la escena cuando se crea un sistema de partículas.
- *Mesh Particle Emitter*: Con este emisor se puede crear un sistema de partículas en una superficie concreta. Este sistema permite tener más control y por tanto más precisión en el diseño de las partículas y sus movimientos pero consume más recursos.
- *World Particle Collider*: tal como su nombre lo indica es un tipo de “collider” específico para los sistemas de partículas. Este sistema permite tratar las colisiones de las partículas con los demás elementos de la escena.

### 14.1. Tutorial: El fuego

En este apartado se van a ver dos sistemas de partículas con texturas diferentes y algunas propiedades específicas, el fuego y el humo.

Existen 2 maneras en Unity de crear un sistema de partículas: en la vista de jerarquía>Create>Particle System o en GameObject>Create Other>Particle System.



*Ilustración 101: Particle system*

Aparece en la escena un sistema de partículas con una textura y opciones predeterminadas. El sistema de partículas tal como se puede comprobar en la vista de inspector tiene 3 componentes: *Ellipsoide Particle Emitter*; *Particle Animator*; *Particle Renderer*. Se van a configurar las características del sistema a través del primer componente:



- **Min Size:** El tamaño mínimo → 0,3
- **Max Size:** El tamaño máximo. → 0,8
- **Min Energy:** Tiempo de vida mínimo de las partículas → 1,5
- **Max Energy:** Tiempo de vida máximo de las partículas → 3
- **Min Emission:** El número mínimo de partículas que se ven generadas.
- **Max Emission:** El número máximo de partículas que se ven emitidas.
- **World Velocity:** La velocidad global de las partículas en los 3 ejes de coordenadas. → Al poner la y: 1,5 se puede ver como sube el sistema de partículas.
- **Local Velocity:** La velocidad local de las partículas en los 3 ejes de coordenadas.
- **Rnd Velocity:** Añade velocidad aleatoria en cada uno de los ejes, lo que hace que las partículas no se comporten de la misma forma en cada uno de los 3 ejes. → Se ponen las 3 coordenadas X, Y e Z a 0,25.
- **Elipsoide:** Representa la esfera que está alrededor del sistema de partículas. → Se pone a 0.5 los 3 ejes de coordenadas X,Y e Z.
- **One Shot:** activar esta opción permite que las partículas aparezcan un rato y exploten. → en este caso se deja desactivada.

La vista del inspector queda como sigue:

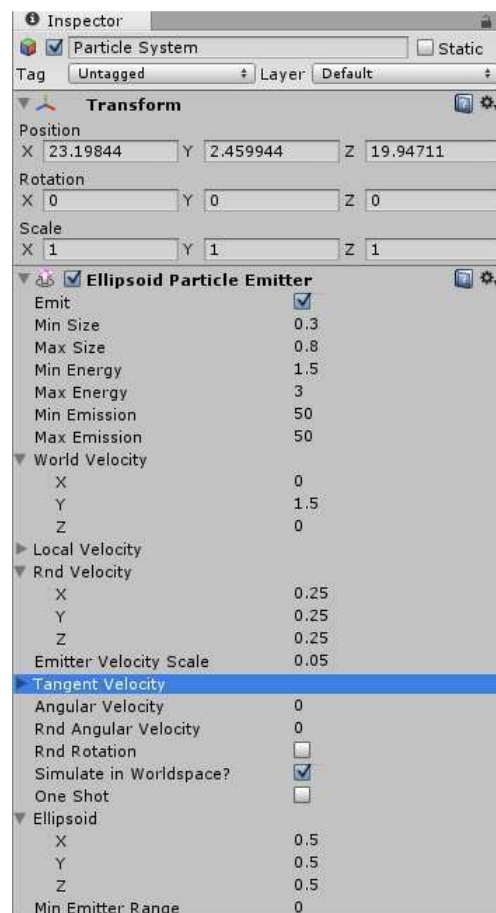


Ilustración 102: Elipsoide particle emitter

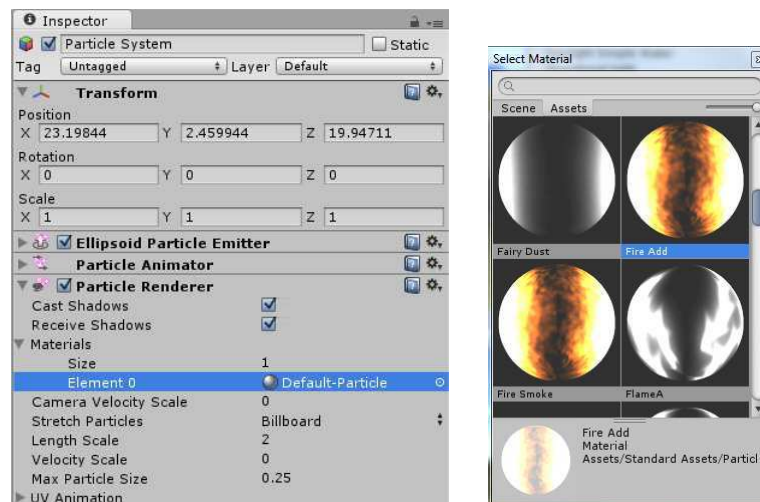


En la escena se puede observar un sistema de partículas como el que sigue:



*Ilustración 103: Resultado del sistema de partículas*

Seguimos con el sistema de partículas en la vista del inspector, se accede al componente Particle Renderer > Materials > Element 0 y se pulsa en el pequeño círculo de la derecha y se elige la textura de “Fire Add”.



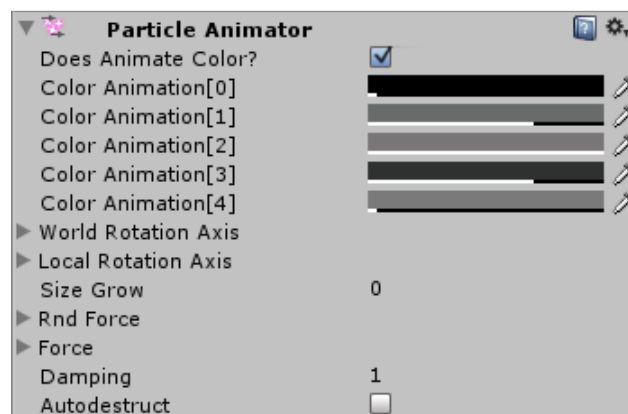
*Ilustración 104: Particle Renderer*

El resultado de realizar los pasos del este tutorial es el que se puede observar en la escena de la imagen que sigue:



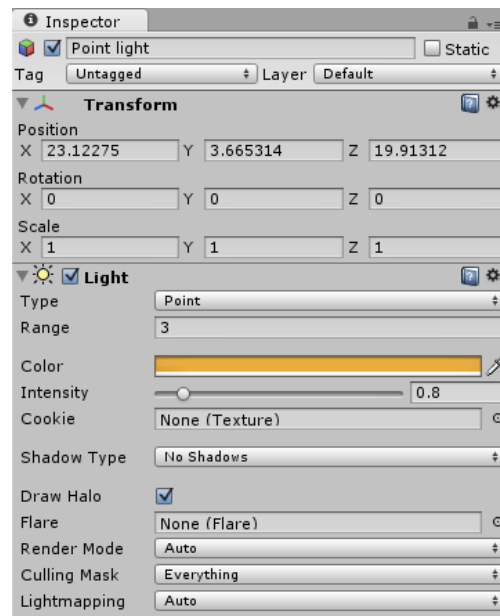
*Ilustración 105: El sistema fuego*

El componente que falta por ver es el Particle Animation, en este componente se puede ir probando con diferentes tonalidades de grises para crear un humo más realista.



*Ilustración 106: Particule Animator*

Se selecciona el sistema de partículas, se renombra en “fuego” y se crea una “Point light” (GameObject> Create Other> Point Light). Esta luz se coloca en el centro del fuego para iluminarlo y se procede a modificar los parámetros de esta: se elige un color de tono naranja y se activa el “Draw Halo” que permite arrojar luz al fuego. Como se puede observar que el fuego tiene mucho “Halo” se reduce el parámetro a 3 y se baja la intensidad “Intensity” a 0.8.



*Ilustración 107: Point light para el fuego*

Ya tenemos nuestro fuego con el siguiente aspecto:



*Ilustración 108: El fuego*

## 14.2. Tutorial: El Humo

Ahora vamos a ver el segundo sistema de partículas que va a representar el humo. Se crea de la manera vista más arriba un sistema de partículas colocado encima del fuego y se procede a cambiar las características de este sistema para que queden como se ve en la imagen:

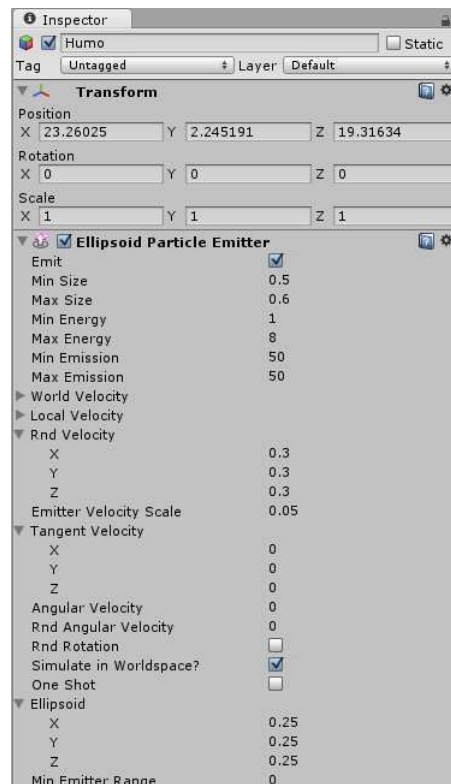


Ilustración 109: Características del humo 1

En el componente Particle Animator, se añaden grises igual que con el fuego hay que ir probando con las diferentes tonalidades. En el componente Particle Renderer se pone como material el Dust Material 1.

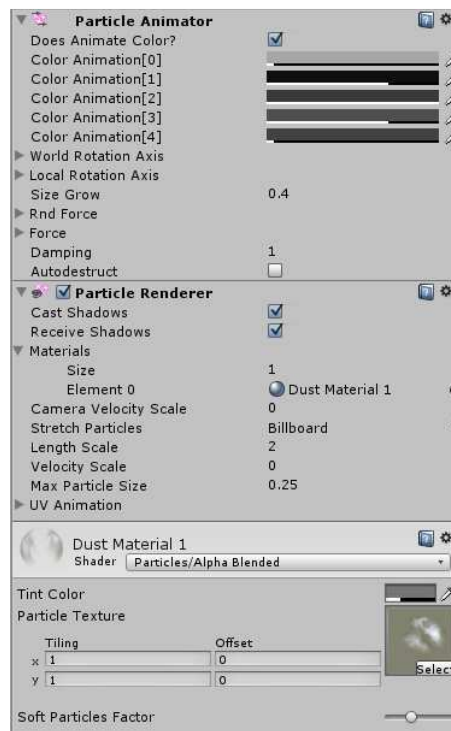


Ilustración 110: Características del humo 2



Finalmente se obtiene el resultado de la imagen que sigue:



*Ilustración 111: El humo*

Por último con los tres elementos creados, el fuego, el humo y la luz, se va a crear un Prefab para poder instanciarlo en cualquier sitio de la escena.



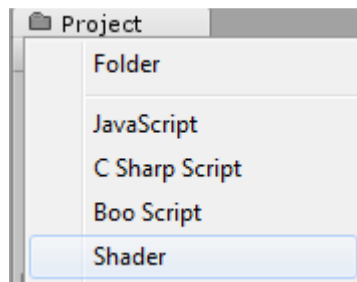
*Ilustración 112: Combinación fuego y humo*



## 15. SHADERS:

Unity permite la utilización de shaders y su aplicación sobre texturas. El uso de estos shaders depende de la tarjeta gráfica que se utiliza para renderizar el videojuego y se utilizan para que las texturas queden lo más realistas posible alterando la forma en que se ven afectados por la iluminación o para aplicar mapping por ejemplo.

Se pueden aplicar shaders de creación propia, los disponibles en web o los que ofrece unity. Para crear un shaders hay que acceder “Create>Shader” en la vista de proyecto:



*Ilustración 113: Los shaders*

Los shaders se utilizan para dar un aspecto preciso a los objetos sobre los cuales se aplican por decirlo de otra manera la aplicación de un material y para crear efectos especiales, como por ejemplo iluminación, fuego o niebla. Para la creación de los shaders se utilizan lenguajes de alto nivel tales como el ShaderLab con la extensión “.shader”.

A la hora de utilizar los shaders hay que asegurarse de su compatibilidad con el dispositivo para el cual se está creando el juego para visualizarlo de manera correcta.

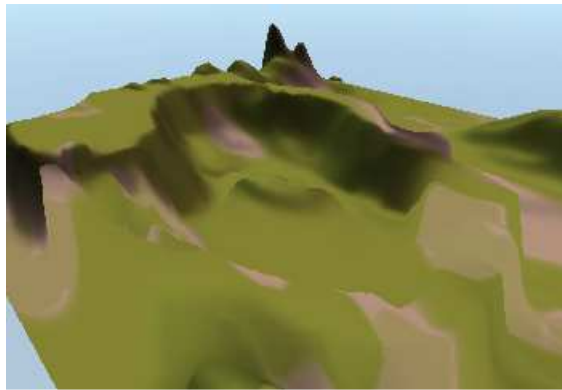
La versión profesional de Unity ofrece funciones añadidas como efectos de iluminación, efectos de post procesamiento.

En este manual no se ha creado ningún shaders pero para hacer una prueba y ver los efectos que se pueden obtener solo hay que descargar algún shaders disponible en web e importarlo. Una vez importado el shaders, este se aplica al objeto como si fuera una textura arrastrando.

### 15.1. Tutorial: Agua transparente

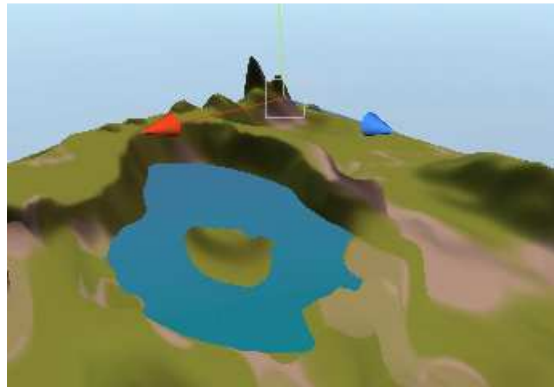
Se parte de un terreno con todos los detalles que se quieran incluir. Se crea en el terreno una estructura honda para un lago como se puede ver en la imagen:





*Ilustración 114: Terreno para lago*

El siguiente paso es incluir el agua accediendo a Standard Asset>Water>Daylight Simple Water y arrastrándola hasta el terreno. Se coloca y se manipula de manera a que quede definido el perímetro del lago.



*Ilustración 115: Aplicación del agua*

Se va a importar un shader de agua transparente, en este se ha usado un shader conseguido gracias a un usuario de la página de unityspain. También se pueden crear sus propios shaders y usarlos. Una vez importado el shader, se guarda en una carpeta previamente creada en la vista de proyecto.



*Ilustración 116: Shaders*

Se crea un material dentro de esta misma carpeta haciendo clic en el botón derecho y Create>Material. Se renombra el material (haciendo clic en F2) en agua transparente y aparecen estos datos en la vista del inspector.

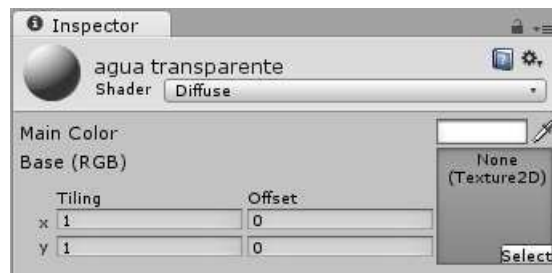


Ilustración 117: Agua transparente

Se accede a la opción indicado como shader y se selecciona la opción FX>WaterPlane (transparent).

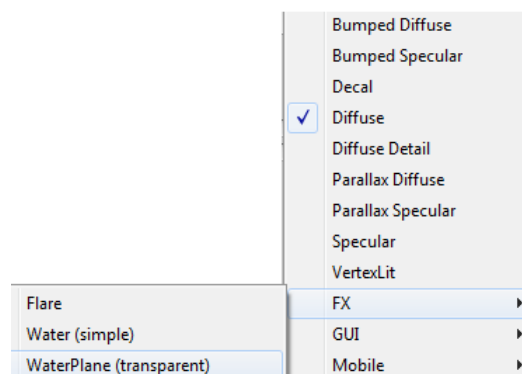


Ilustración 118: WaterPlane

Aparecen 4 apartados para colocar diferentes texturas. Se accede en los Standard Assets>Water>Source>Textures.

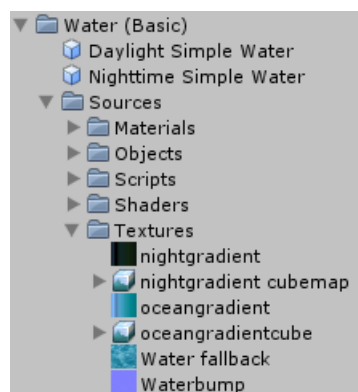
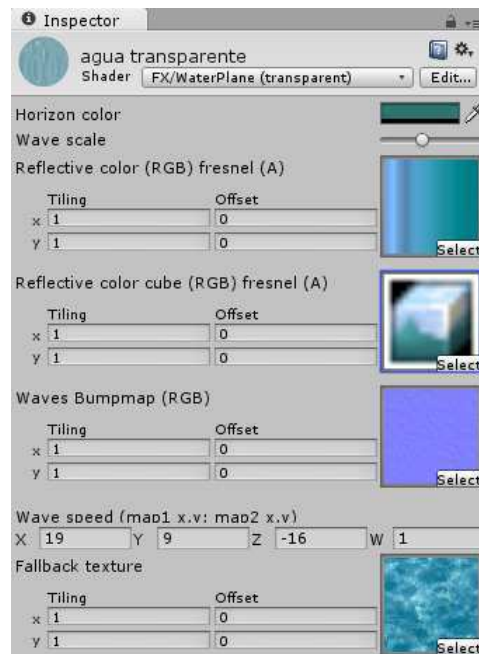


Ilustración 119: Texturas de agua

Se colocan las texturas Oceangradient, Oceangradient cubemap, Waterbump y Water fallback para que quede el material definido tal como sigue:



*Ilustración 120: Texturas WaterPlane*

Ahora vamos a aplicar el material al agua para ello se accede a la carpeta creada anteriormente Shaders>agua transparente y se arrastra hasta materials en la vista del inspector.



*Ilustración 121: Aplicar agua transparente*

Tal como se pudo comprobar en la escena, el aspecto del agua ha cambiado y ahora es transparente y se puede ver el fondo.



*Ilustración 122: Aspecto final del lago*

## 16. MATERIALES FÍSICOS:

En este apartado se va a detallar la manera de asignar a los elementos que se crean en unity propiedades de manera a que estos elementos tengan un comportamiento físico concreto.

### 16.1. Tutorial: Rigidbody

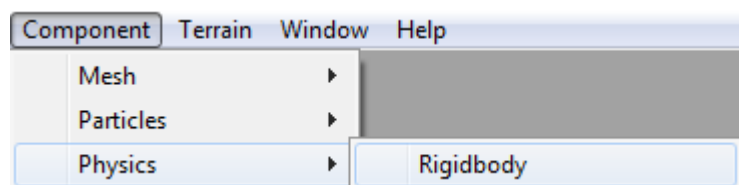
A modo de ejemplo, se crea una pelota mediante una esfera del modo usual GameObject> Create Other> Sphere. Cuando ya se obtiene la pelota, se texturiza y se le asigna un “Rigidbody”.



*Ilustración 123: Rigidbody*

El Rigidbody es un componente que ofrece unity 3D para que los objetos sean afectados por el motor de físicas. Puesto que este componente hace que los elementos se vean afectados por fuerzas lineales o de torsión, el comportamiento que se obtiene es muy realista.

Para añadir este componente a la pelota se accede a Component>Physics>Rigidbody.



*Ilustración 124: Componente Rigidbody*

Tal como se puede ver en la vista de inspector aparece el nuevo componente. Está compuesto por varios parámetros que permiten darle al objeto el comportamiento objetivo.



*Ilustración 125: Características del Rigidbody*

Las diferentes características que pueden afectar al elemento con el rigidbody:

- Drag: define la influencia del aire en el movimiento del objeto.
- Use Gravity: esta opción permite aplicar gravedad al objeto entonces si se pone la pelota encima del nivel del suelo esta calle.
- Is Kinematic: permite que las leyes de física solo afecten al elemento que lo tiene activado cuando se realiza alguna transformación sobre este.

## 16.2. Tutorial: Propiedades de materiales

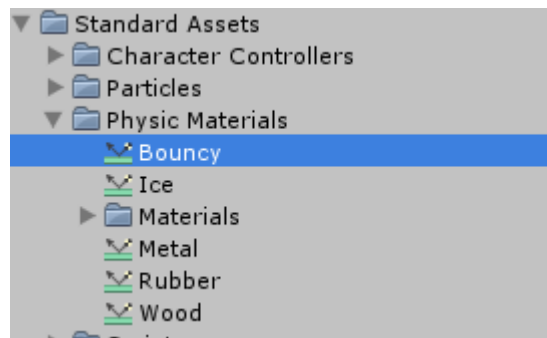
Seguimos en la vista de inspector pero en otro componente que tiene la pelota “Sphere Collider”. En este componente se puede ver que hay un material que no tiene que ver con la textura que se le ha asignado a la pelota. De nuevo se coloca la pelota por encima del nivel del suelo, se accede a los “Standard Asset>Physics Materials>Bouncy” en la vista de proyecto y se arrastra este material al “Sphere Collider”:



*Ilustración 126: Añadir material 1*

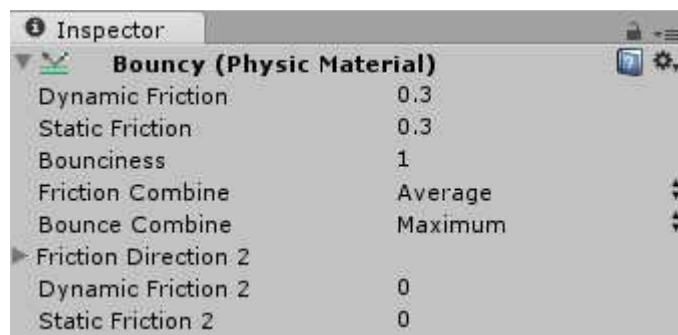
Por último, se lanza el juego para observar que ahora la pelota bota en la escena.

En los “Standard Asset>Physics Materials” se puede observar que existen varios materiales:



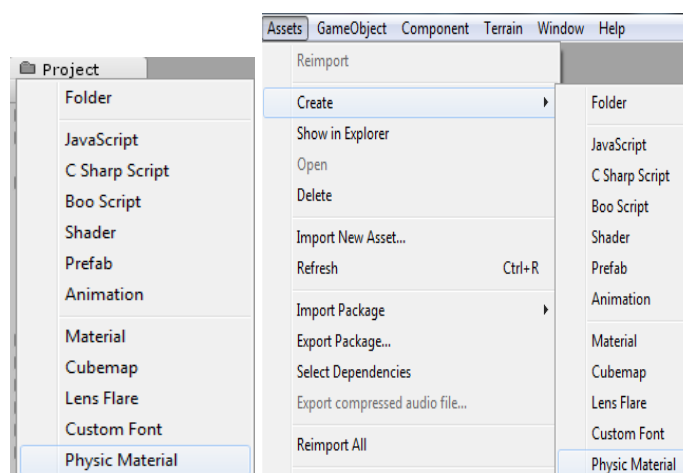
*Ilustración 127: Materiales existentes*

Cada uno de estos materiales viene definido con unas características concretas, las cuales se pueden ver seleccionándolo:



*Ilustración 128: Propiedades de los materiales*

El usuario puede también crear su propio material accediendo a “Create>Physics Materials” en la vista de proyecto o en la barra del menú “Asset>Create>Physics Materials”:



*Ilustración 129: Creación de un material*

Aparecen los parámetros por defecto del material en la vista del inspector para que el usuario pueda modificarlos.



### 16.3. Tutorial: Fixed Joint

Para este tutorial vamos a utilizar la pelota creada anteriormente quitándole el material “Bouncy”. Se crea un cilindro cuyos parámetros se modifican para que quede como en la imagen y se le asigna un rigidbody:



Ilustración 130: Fixed Joint 1

Ahora a la pelota se le va a añadir un “Fixed Joint”, este componente permite que los movimientos de la pelota varíen en función del cilindro. Se accede a Component>Physics>Fixed Joint.

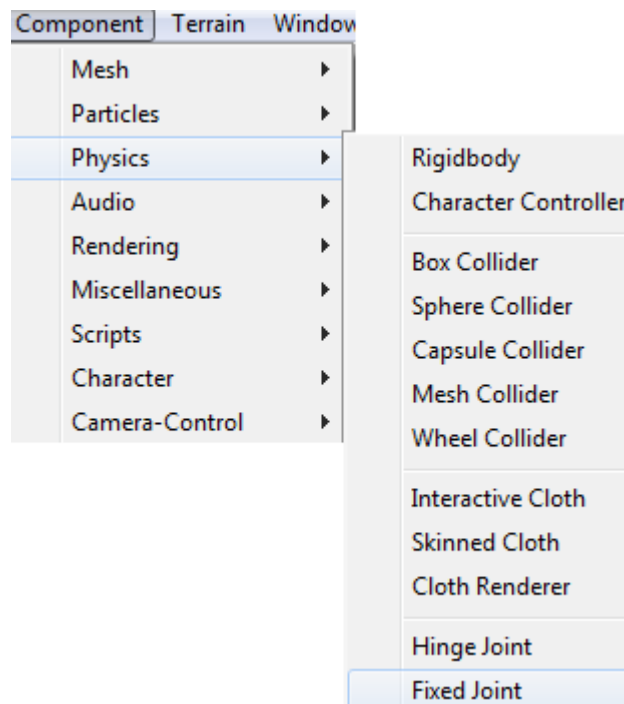


Ilustración 131: Fixed Joint 2



Una vez hechos estos pasos, aparece el nuevo componente en la vista del inspector se rellena el “connected body” con la barra (el cilindro).



Ilustración 132: Elemento conectado con Fixed Joint

Se lanza el juego y se puede comprobar que la pelota se mueve en función del movimiento de la barra.

## 16.4. Tutorial: Hinge Joint

Seguimos con el ejemplo anterior y vamos a crear una capsula con un tamaño parecido al cilindro y se le adjudica un rigidbody. Mientras que la barra se le asigna un nuevo componente “Hinge Joint”.

Este componente permite que los elementos queden conectados entre sí y sus movimientos dependientes los unos de los otros. Para añadirlo se accede a Component>Physics>Hinge Joint y aparece en la vista de inspector para que se le conecte el elemento que se quiere en “connected Body”:

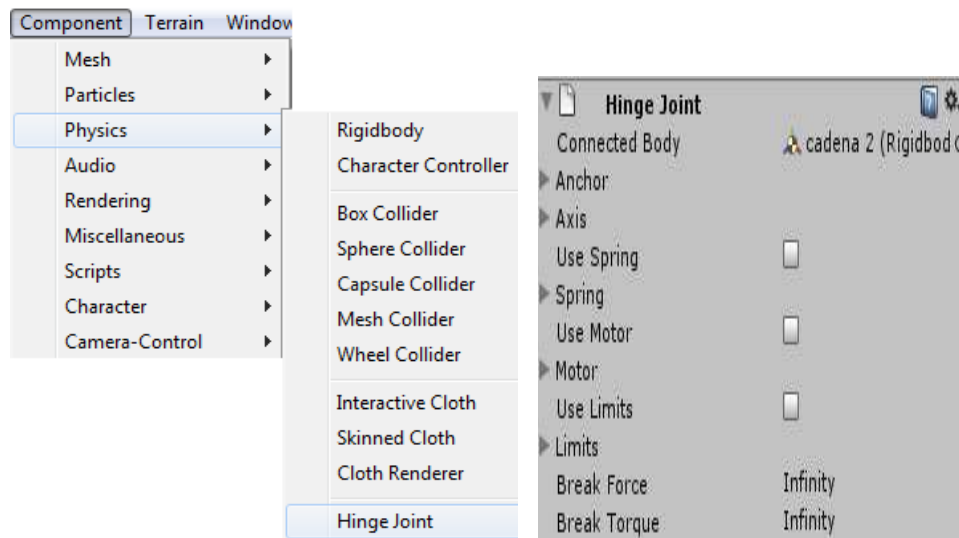


Ilustración 133: Hinge Joint

Finalmente el ultimo trozo de la cadena se le pone un “Hinge Joint” dejando el campo “Connected Body” vacío para que quede conectado con el universo. Se unen todos los elementos en un objeto vacío que se nombra como péndulo, se coloca en una posición más alta que el suelo, se gira de manera a que quede de lado y se lanza el juego. Se puede observar que el movimiento de este objeto es el de un péndulo que va balanceándose de un lado a otro perdiendo fuerza:



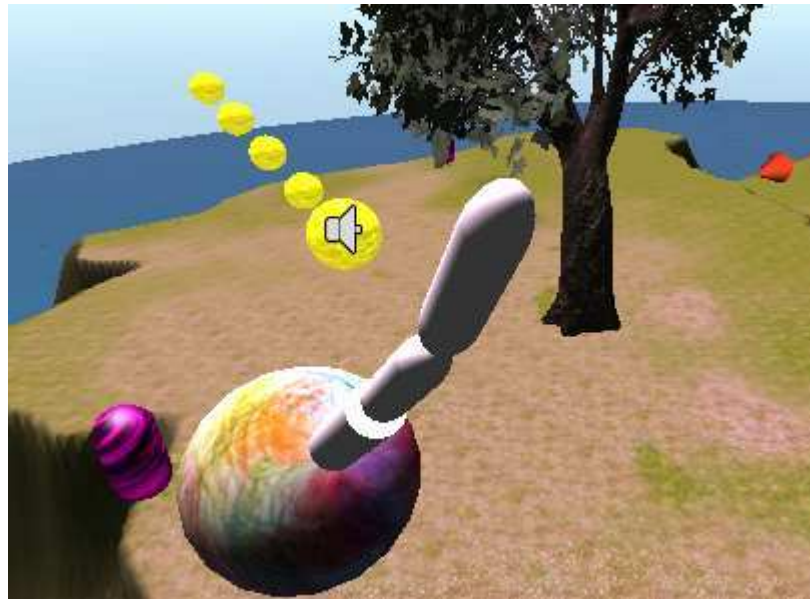


Ilustración 134: Movimiento del péndulo

## 16.5. Tutorial: Simulación de tela

En este apartado se va a explicar cómo simular los movimientos de una tela en Unity 3d.

### Alfombra:

Primero se va a crear una esfera que se deja en el suelo y un plano que cuelga encima de esfera que hará las veces de alfombra. Se les adjudica el tamaño adecuado y una textura y seguido se le añade a la alfombra un “Interactive Cloth” accediendo a Component>Physics>Interactive Cloth.

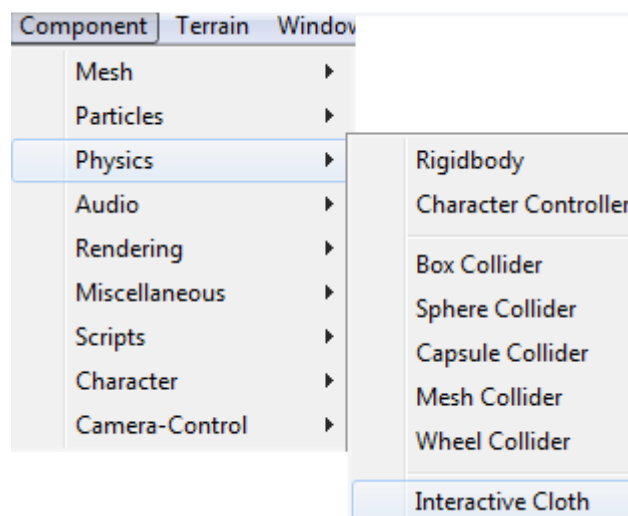


Ilustración 135: Interactive Cloth



Este componente aparece en la vista del inspector y se pueden configurar sus características. Se pueden configurar varias opciones rigidez de flexión (Bending Stiffness), rigidez de estiramiento (Stretching Stiffness), doblamiento (Damping), grosor (Thickness), gravedad, etc...

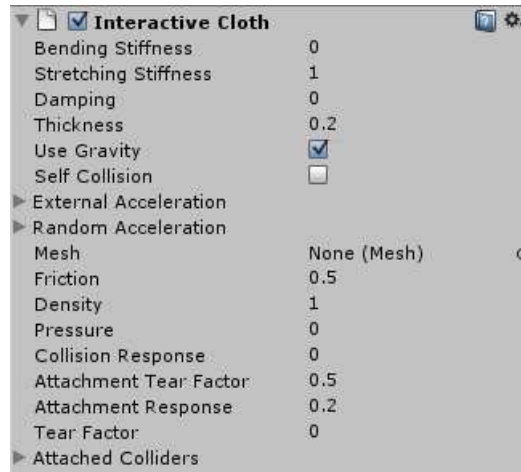


Ilustración 136: Características del Interactive Cloth

Estos pasos se pueden realizar de otra manera creando un GameObject “Cloth”. Este GameObject viene definido con dos componentes el “Interactive Cloth” y el “Cloth Renderer”. El “Cloth Renderer” permite asignar un material al objeto y que este pueda recibir o arrojar sombras.

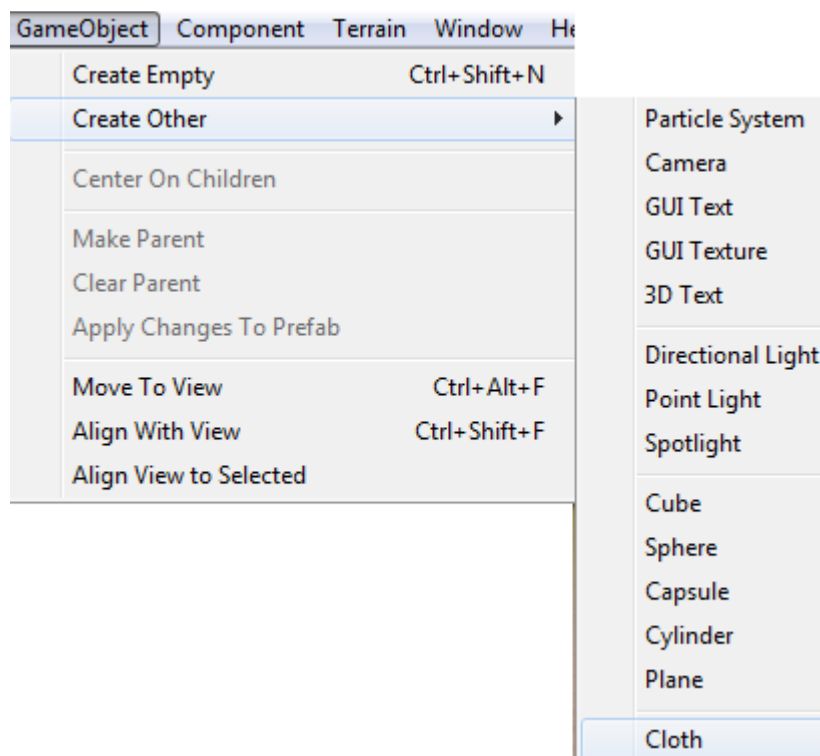
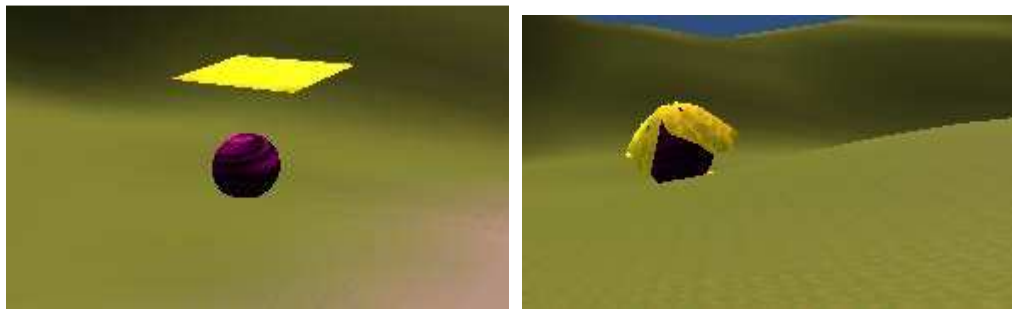


Ilustración 137: Gameobject Cloth

Se van a ir modificando las características del “Interactive Cloth” para ver lo que permite cada opción. A modo de ejemplo se activa el efecto de gravedad (Use Gravity) y el efecto de mojado (Damping a 1) y se lanza el juego para ver como la alfombra cae sobre la esfera como si fuera mojada y van apareciendo dobleces.

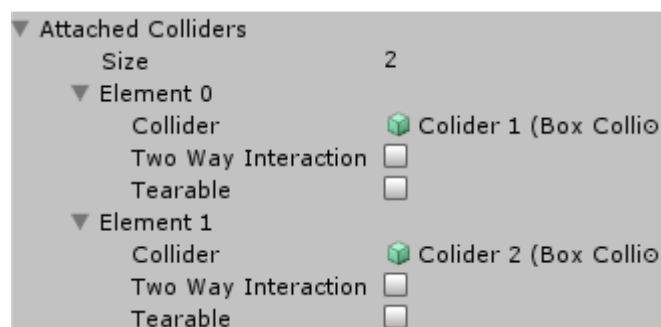


*Ilustración 138: Caída de la alfombra*

#### Bandera:

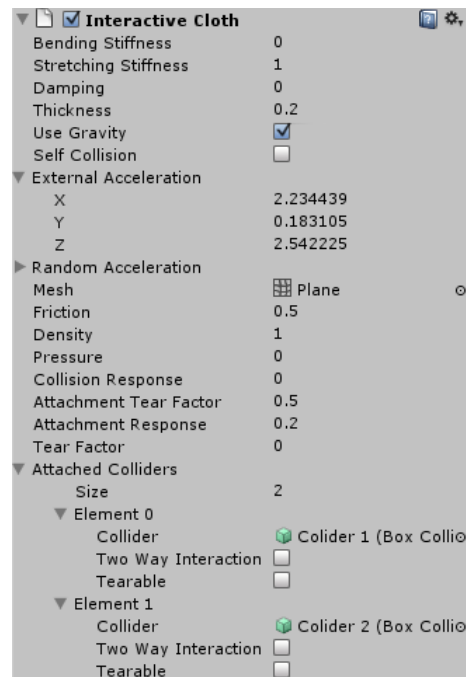
Se crea una bandera de manera muy sencilla, con un cilindro se obtiene el mástil que la va a sujetar y con objeto “Cloth” la bandera. Se crean dos objetos vacíos (Create Empty) y en cada uno se asigna un collider que van a permitir sujetar la bandera con el mástil (esto se hace para que la bandera no se caiga si la opción de gravedad está activada).

Puesto que la bandera es un objeto “Cloth” viene con un componente “Interactive Cloth” que tiene un parámetro llamado “Attached Colliders”. Como queremos juntar la bandera con 2 elementos ponemos el sub-parámetro “size” a 2 y aparecen 2 elementos. Arrastramos los 2 colliders creados anteriormente a estos elementos:



*Ilustración 139: Inserción de colliders*

Primero hay que asegurarse que los elementos: bandera, mástil y colliders están realmente sujetos. Seguido se van modificando las coordenadas de “External Acceleration”:



*Ilustración 140: Propiedades bandera*

Se prueba el juego hasta obtener el efecto de los movimientos de una bandera. Se puede comprobar que a pesar de estar activada la opción de gravedad, la bandera no se caiga.



*Ilustración 141: Movimiento de la bandera*



## 17. PERSONALIZAR LA INTERFAZ:

Para personalizar la interfaz existen 2 opciones en Unity 3D:

- Opción 1: componentes GUI Texture y control de eventos por scripts. Este método consiste en crear objetos GUI Texture y mostrar sus texturas con la ayuda de scripts basado en los eventos del ratón: recorrer un menú, hacer clic, soltar el botón. No hace falta mucho código para la creación de los botones pero todas las acciones se controlan por el script.
- Opción 2: Clase UnityGUI y recursos GUI skin. Con este método, contrariamente a la opción anterior el entero menú se crea con la ayuda de scripts. La creación de elementos de menú requiere más código en un principio pero se pueden crear recursos GUI skin para definir la apariencia y el comportamiento de los elementos en el panel de inspector.

En general los desarrolladores prefieren la opción 2 para la creación de los menús completos para un juego por su flexibilidad. Los elementos del menú se crean en el script pero su aspecto se define con la ayuda de componentes GUI.

Además los parámetros de los componentes GUI de Unity son muy sencillos de utilizar porque han sido creados para usuarios que no tienen gran conocimiento previo.

### 17.1. Tutorial: Barra de desplazamiento:

Se inserta un cubo en la escena y se abre un Javascript en el cual se escribe el siguiente código:

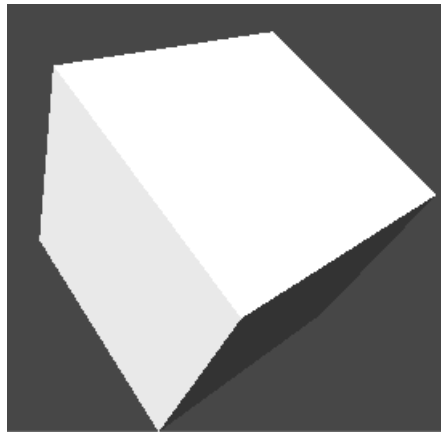
```
static var vitesse = 10.0;

function Update () {
    transform.Rotate(Vector3.right*Time.deltaTime*vitesse);
}
```

Explicación del código:

`static var vitesse = 1.0;` → variable de velocidad  
`transform.Rotate(Vector3.right*Time.deltaTime*vitesse);` → permite rotar el cubo en el eje x.

Se guarda este script para compilar el código y se comprueba que no hay error. Se asigna el script al cubo y se ejecuta el juego. De momento en la escena aparece el cubo rotando.



*Ilustración 142: Cubo rotando*

Se vuelve a abrir el scripts y se añade una función de la siguiente manera:

```
- function OnGUI(){  
  GUI.Box(Rect(20,90,80,70), "Modificar la velocidad de rotacion");  
  vitesse = GUI.HorizontalSlider (Rect(22, 130, 75, 25),vitesse, -150.0, 150.0);  
}
```

Explicación del código:

GUI.Box(Rect(20,90,80,70), “Modificar la velocidad de rotación”); → Se crea una caja (Box) que va a ser rectangular (Rect) con estas características:

- Posición: 20,90 → esta posición está definida por el desplazamiento en pixeles.
- 80 pixeles de largo
- 70 pixeles de alto.
- Se muestra el mensaje: “Modificar la velocidad de rotación”.

Vitesse = GUI.HorizontalSlider(Rect(22,130,75,25), vitesse, -150.0, 150.0) ; → esta línea permite la creación de una barra de deslizamiento.

Se compila el código para comprobar que no hay errores y se lanza el juego. En la escena ahora aparece una barra de desplazamiento que permite modificar la velocidad de rotación del cubo. Al desplazar la barra se puede ver que la velocidad de rotación del cubo cambia, al mover la barra hacia la derecha la velocidad aumenta y en el sentido contrario la velocidad baja.



Ilustración 143: Barra de desplazamiento

## 17.2. Tutorial: Cronometro:

Se va a crear otro JavaScript llamado “TimerScript” con el código que sigue:

```
private var timer : int = 0;
private var attente: float = 1.0;
private var actualiser : boolean = false;

- function Update () {
-   if (!actualiser){
-       StartCoroutine("Pause");
-   }
- }

- function OnGUI(){
  GUI.Label(new Rect (15, 15, 100, 20), timer.ToString());
}

- function Pause(){
  actualiser = true;
  yield new WaitForSeconds(attente);
  timer ++;
  actualiser = false;
}
```

### Explicación del código:

`private var timer : int = 0;` → variable que se va incrementando con el paso del tiempo.

`private var attente: float = 1.0;` → variable que permite esperar un segundo antes de incrementar el timer.

`StartCoroutine("Pause");` → lanzar una rutina llamada “Pause” que se va a crear más adelante.

`GUI.Label(new Rect (15, 15, 100, 20), timer.ToString());` → creación de elemento de interfaz donde va a aparecer el timer.



`yield new WaitForSeconds(attente);` → esta función permite esperar durante el tiempo indicado 1 segundo.

Se guarda este código, se añade el script a un cubo y se ejecuta el juego. Se puede observar que aparece un timer en la esquina superior izquierda de la escena y que se va incrementando cada segundo.



*Ilustración 144: TimerScript*

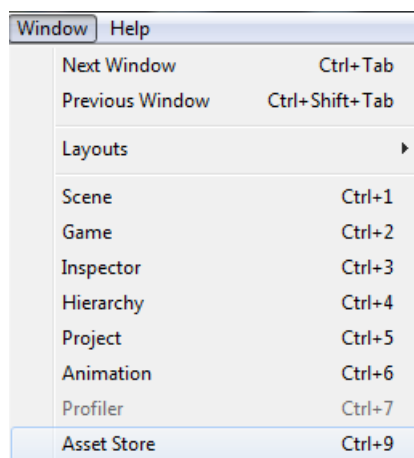




## 18. IMPORTACIÓN DE MODELOS:

Unity 3D admite la importación de elementos 3D siempre que estos tengan formato “.fbx”. Para la creación de los modelos 3D, existen dos opciones. La primera es la utilización de cualquier aplicación que permita la exportación en formato “.fbx” como blender. La segunda opción es utilizar “Autodesk FBX converter”, este software permite transformar los demás formatos 3D a formato “.fbx” y es totalmente gratuito.

Además Unity proporciona una gran variedad de modelos 3D y algunos son totalmente gratuitos. Para importar uno de los objetos 3D que proporciona Unity 3D, se accede en el menú a Window>Asset Store y entonces se abre una nueva ventana.



*Ilustración 145: Asset Store 1*

El Asset Store es como el Apple Store, contiene objetos 3D, proyectos completos, paquetes de scripts y extensiones para Unity. Se puede explorar para descubrir todas las posibilidades que ofrece el Asset Store.



*Ilustración 146: Asset Store 2*



## 18.1. Tutorial: Importar un personaje

Se va a usar un modelo 3D gratuito definido en Unity, para ello se accede al Asset Store. Se accede a Character Controller> Humanoides:

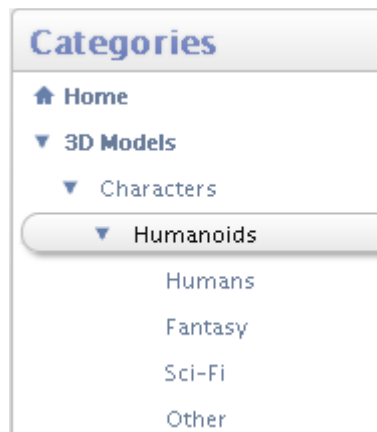


Ilustración 147: Modelos 3D

Se puede ver que existen muchos objetos 3D, la mayoría son de pago pero existen algunos gratuitos.

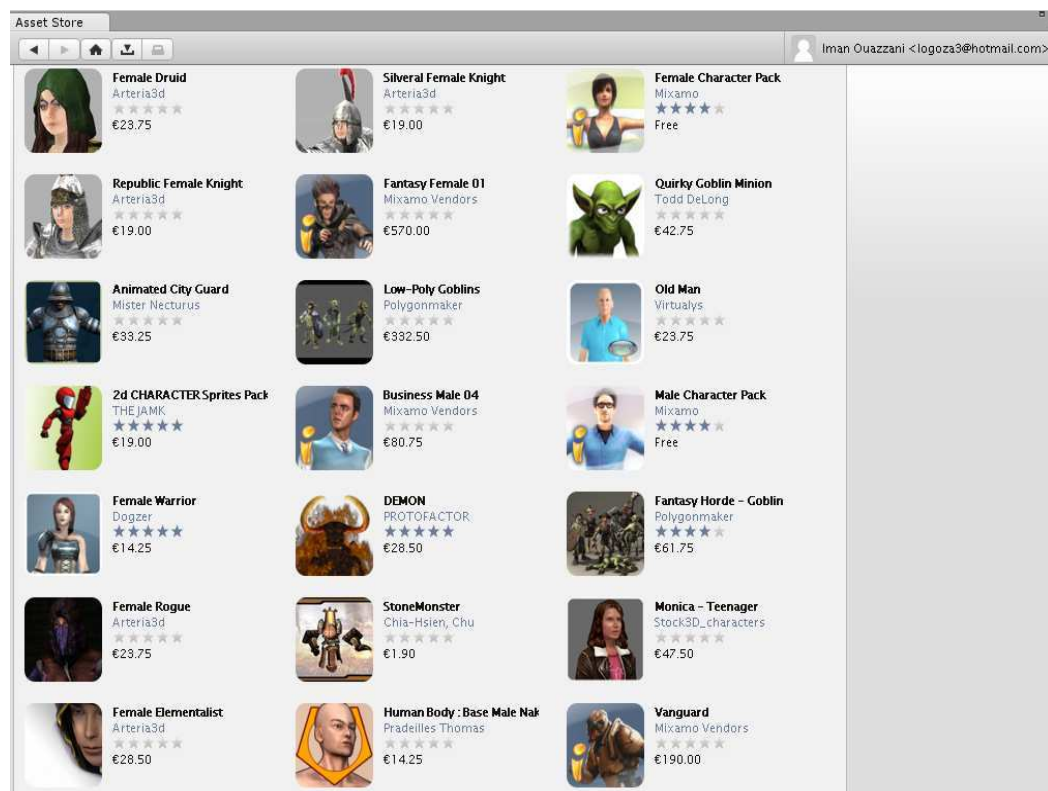
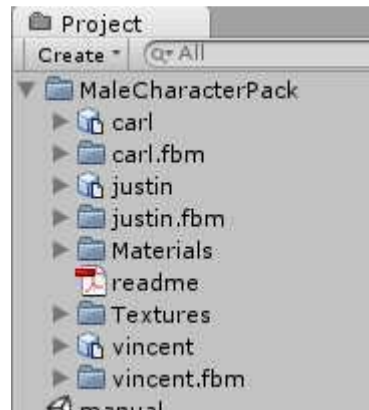


Ilustración 148: Modelos humanoides

Se va a importar el “MaleCharacterPack” después de registrarse. En este paquete vienen definidos tres personaje de los cual solo se va a utilizar uno. Una vez la importación acabada, nos aparece una carpeta nueva



“MaleCharacterPack” en la vista de proyecto. Esta carpeta contiene los personajes y sus texturas.



*Ilustración 149: Importación del modelo 3D*

Se elige uno de los 3 personajes definidos como prefab (carl, justin y vincent), se arrastra a la escena y ya tenemos nuestro personaje:



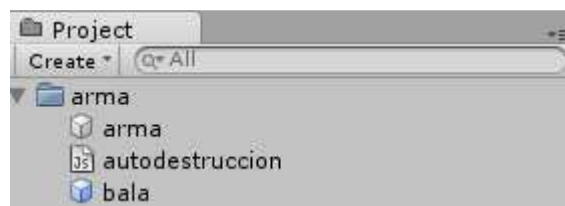
*Ilustración 150: Inserción del modelo 3D*



## 19. INTELIGENCIA ARTIFICIAL:

Se va a crear un arma que dispara proyectiles contra los enemigo. El arma estará compuesta por un soporte y una metralleta que rota y se dispara automáticamente mediante un script de inteligencia artificial y las bala disparadas se autodestruyen en 5 segundos.

Primero se crea una carpeta en la vista de proyecto que va a contener todos los elementos que se van a utilizar y se nombra “arma”. Se definen 2 prefab en esta carpeta, uno para una bala y otro para el arma.



*Ilustración 151: Creación del arma*

### 19.1. Tutorial: Una bala

Se crea una esfera en GameObject>Create Other> Sphere, se le da el tamaño adecuado para una bala y se le aplica el Rigidbody en Component> Physics> Rigidbody.

Puesto que las balas se van a disparar sin cesar y estarán constantemente en la escena, se implementa un Javascript para que se destruyan en 5 segundos con el siguiente código:

```
var duracionVida : float = 5;  
  
- function Awake() {  
    Destroy(gameObject, duracionVida);  
}
```

Se adjudica el script a la esfera y seguido se coloca esta esfera en el prefab creado anteriormente para ello.



*Ilustración 152: Creación de la bala*

## 19.2. Tutorial: Un arma

El arma va a ser muy sencilla compuesta por 2 cilindros y una esfera. La base es un cilindro vertical sobre el cual se pone una esfera de la cual tiende un cilindro horizontal. La base es estática mientras que la esfera es giratoria lo cual permite tener un ángulo de visión muy amplio. Una vez creada la estructura se le pone una textura metálica y se adjudica al prefab anteriormente creado (arma). El resultado es el que sigue:



*Ilustración 153: Creación de la cabeza rotatoria*

A continuación se crea un script que se va a llamar IA. Este script va a definir el comportamiento del arma de manera a que la base este estática y que el soporte giratorio se va moviendo disparando balas cada 5 segundos.

```
var objetivo : Transform;
var bala : Transform;
var tirarFuego;

function Update () {

    var rotate = Quaternion.LookRotation(objetivo.position - transform.position);
    transform.rotation = Quaternion.Slerp(transform.rotation, rotate, Time.deltaTime * 1);

    var temps : int = Time.time;
    var cadence = (temps % 4);

    if(cadence){
        tir(temps);
    }
}

function tir(temps){
    if(temps != tirarFuego){
        var tirer = Instantiate(bala, transform.Find("salida").transform.position, Quaternion.identity);
        tirer.rigidbody.AddForce(transform.forward * 3000);
        tirarFuego = temps;
    }
}
```





### Explicación del código:

`var objetivo : Transform;` → el objetivo sobre el cual se va a disparar el arma.

`var bala : Transform;` → la bala que será disparada por el arma.

`var tirarFuego;` → para controlar el momento en el que se puede disparar.

`var rotate = Quaternion.LookRotation(objetivo.position - transform.position);`  
→ Esta variable permite hacer rotar la parte de arriba del arma mirando el objetivo. Con la resta se obtiene una línea recta que describe la trayectoria de las balas.

`transform.rotation = Quaternion.Slerp(transform.rotation, rotate, Time.deltaTime * 1);` → permite ralentizar un poco la rotación del arma y no matar el personaje a la primera.

`var temps : int = Time.time;` → declaración de un cronómetro.

`var cadence = (temps % 4);` → definición del ritmo de disparos.

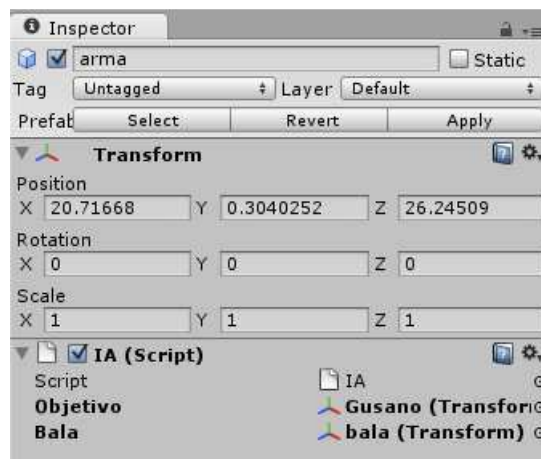
`tirarFuego;` → función definida más abajo y que depende del tiempo.

`var tirar = Instantiate(bala, transform.Find("salida").transform.position, Quaternion.identity);` → Se instancia la bala, su punto de salida definido anteriormente y se le da a bala la orientación adecuada.

`tirer.rigidbody.AddForce(transform.forward * 3000);` → se le adjudica a la bala una fuerza que permite su proyección hacia el objetivo. No hay que poner un valor demasiado alto porque si las balas se lanzan a gran velocidad no se podrán visualizar en el campo de visión.

`tirarFuego = temps;` → se reinicializa el tiempo para lanzar el bucle una vez más.

Se compila el script para comprobar que no hay errores y se adjudica al arma. Una vez que se utiliza el prefab para poner el arma en la escena hay que definir los parámetros que aparecen en la vista de inspector. El objetivo es el gusano (el personaje de los anteriores tutoriales) y en la bala se pone el prefab de la bala.



*Ilustración 154: Script de inteligencia artificial*

Se lanza el juego y se puede comprobar cómo va rotando el arma disparando en dirección del gusano a intervalos variables.



*Ilustración 155: Los disparos del arma*

### 19.3. Tutorial: Movimiento de los enemigos

Este tutorial nos va a permitir controlar los movimientos de los enemigos, haciendo que se muevan siguiendo un camino definido previamente.

Primero se procede a crear un personaje formado por una capsula y un cilindro que nos va a permitir detectar la orientación que sigue el movimiento. A este personaje se le adjudica un Rigidbody y queda como sigue:



*Ilustración 156: El enemigo*

Ahora se va a crear un objeto vacío y se llamará camino. Dentro de este camino se van a crear otros objetos vacíos que se van a nombrar zona 1, zona 2 y zona 3. Estas 3 zonas se colocan allí donde se quiera definir cada zona y asegurarse que están encima del suelo.

Se procede a crear un Javascript con el código que aparece a continuación:



```
var pathPoint : Transform[];  
private var speed = 5;  
private var currentPathPoint : int;  
  
function Awake(){ pathPoint[0] = transform; }  
  
- function Update(){  
-  
-     if (currentPathPoint < pathPoint.length){  
-  
-         var zone : Vector3 = pathPoint[currentPathPoint].position;  
-         var movingTo : Vector3 = zone - transform.position;  
-         var velocity = rigidbody.velocity;  
-  
-         if(movingTo.magnitude < 1){ currentPathPoint++; }  
-         else {velocity = movingTo.normalized * speed; }  
-     }  
-  
-     else{  
-         if(1){ currentPathPoint = 0; }  
-         else{ velocity = Vector3.zero; }  
-     }  
  
-     rigidbody.velocity = velocity;  
-     transform.LookAt(zone);  
- }  
}
```

#### Explicación del código:

`var pathPoint : Transform[];` → una variable que representa un punto del camino y que será una tabla.

`private var speed = 5;` → variable que representa la velocidad.

`private var currentPathPoint : int;` → variable que representa el punto del camino actual y no puede tener un valor con coma.

`function Awake(){ pathPoint[0] = transform; }` → función que se lanza al comienzo de la partida y que permite inicializar el punto de partida con el valor de la posición inicial definida anteriormente.

`if (currentPathPoint < pathPoint.length)` → si el punto actual es menor que el número total de puntos.

`var zone : Vector3 = pathPoint[currentPathPoint].position;` → esta línea permite declarar la zona en la nos encontramos y la zona a la cual nos vamos a desplazar.

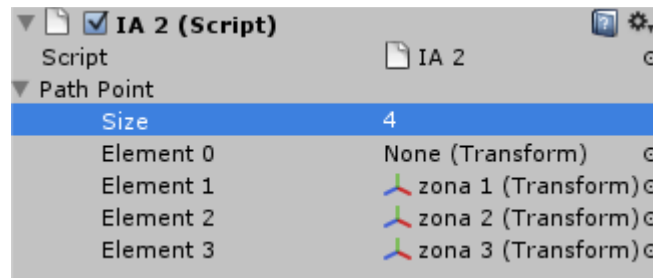
`var movingTo : Vector3 = zone - transform.position;` → permite desplazarnos hacia una posición de la zona a la cual nos queremos dirigir.





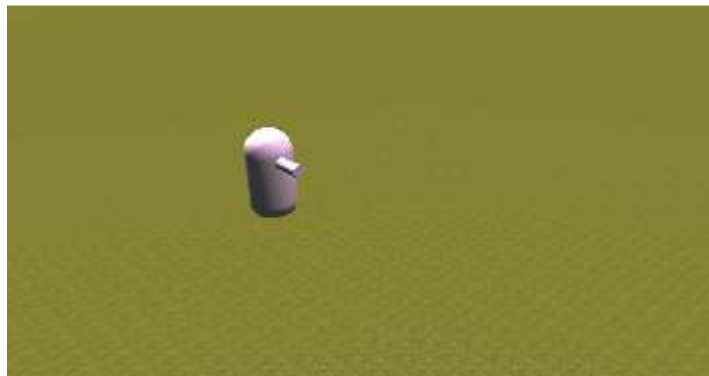
`if(movingTo.magnitude < 1){ currentPathPoint++; } →` si nos encontramos al final de la zona, nos trasladamos a la siguiente zona.

Se adjudica el script de inteligencia artificial al personaje que es el enemigo, entonces en la vista de inspector aparece el script con la variable “Path Point” se pone el size a 4 y entonces nos aparecen 4 elementos. Dejamos el primer elemento vacío y colocamos en los elementos restantes las 3 zonas creadas anteriormente.



*Ilustración 157: zonas de movimiento*

Se guardan los cambios y se lanza el juego. Se puede comprobar que el enemigo se mueve desplazando de una zona a otra.



*Ilustración 158: Movimiento de los enemigos*



## 20. PACMAN:

En este apartado se van a explicar una serie de tutoriales de funcionalidades que han sido implementadas en el videojuego prototipo.

### 20.1. Tutorial: Poner varias cámaras en la escena:

Se va a explicar cómo poner varias cámaras en una escena y como ir cambiando de vista de una cámara a otra. Se ponen 3 cámaras en la escena accediendo a GameObject> Create Other>Camera. Estas cámaras van a ofrecer al usuario 3 vistas diferentes de la escena, una vista en primera persona, una vista en tercera persona y una vista desde la parte superior de la escena.

Se crea un objetos vacío que se va a llamar “cámaras”: GameObject> Create Empty. Seguido se crea un JavaScript como ya se viene haciendo para todos los scripts: Vista de Project> Create> Javascript y se escribe el siguiente código:

```
var CamaraPrincipal : GameObject;  
var Camara : GameObject;  
var CamaraTerceraPersona : GameObject;
```

➔ Estas 3 variables representan las 3 cámaras de la escena.

```
- function Start(){  
    CamaraPrincipal.active = true;  
    Camara.active = false;  
    CamaraTerceraPersona.active = false;  
}
```

➔ Por defecto se activa la cámara principal mediante la función Start.

```
- function Update () {  
    if (Input.GetKeyDown("1"))  
    {  
        CamaraPrincipal.active = false;  
        Camara.active = true;  
        CamaraTerceraPersona.active = false;  
    }  
}
```

➔ La función Update es la que permite controlar el evento de tecla pulsada y activar una cámara u otra según lo que incluya el usuario. La línea de código Input.GetKeyDown("") es la que permite recoger lo que el usuario introduce por teclado.

- ➔ Si el usuario pulsa la tecla 1, se activa la cámara principal y se desactivan las otras 2. Esta es la vista que se obtiene de la escena:

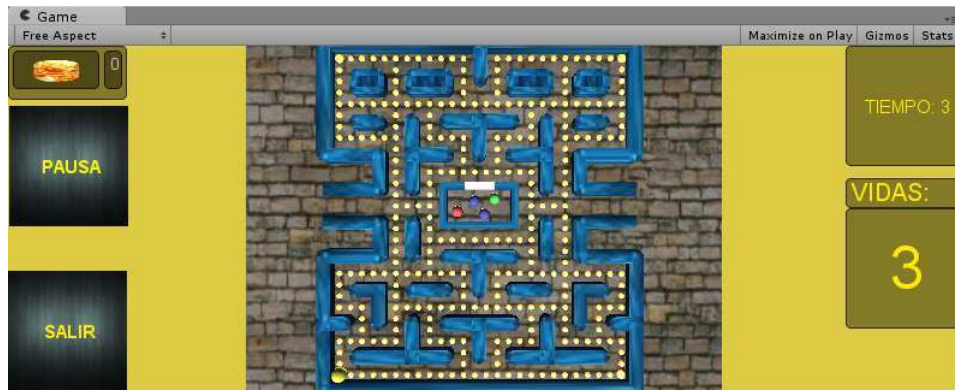


Ilustración 159: Vista desde arriba

```
- if(Input.GetKeyDown("1")){  
    Camara.active = false;  
    CamaraPrincipal.active = true;  
    CamaraTerceraPersona.active = false;  
}
```

- ➔ Si el usuario pulsa la tecla 2, se activa la segunda cámara y se desactivan las otras 2. Esta es la vista que se obtiene de la escena:

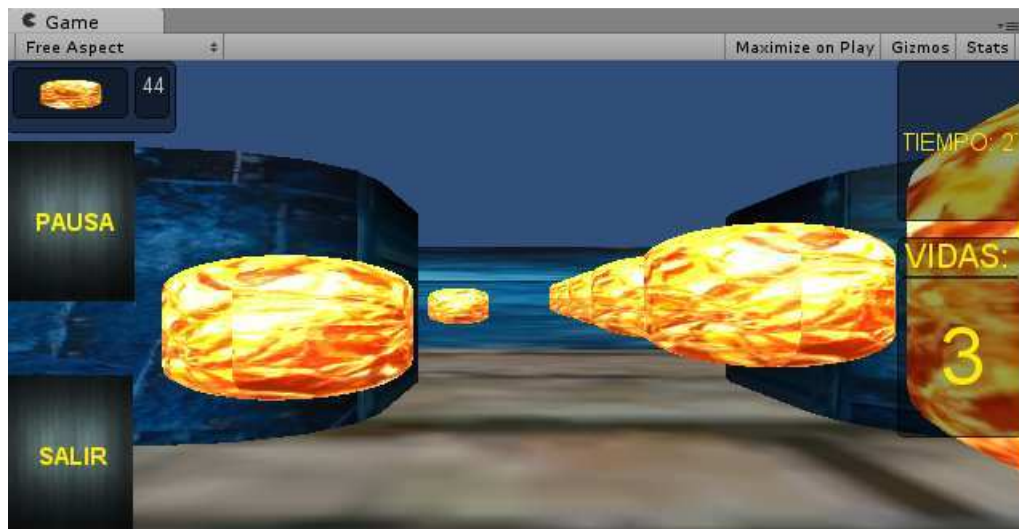
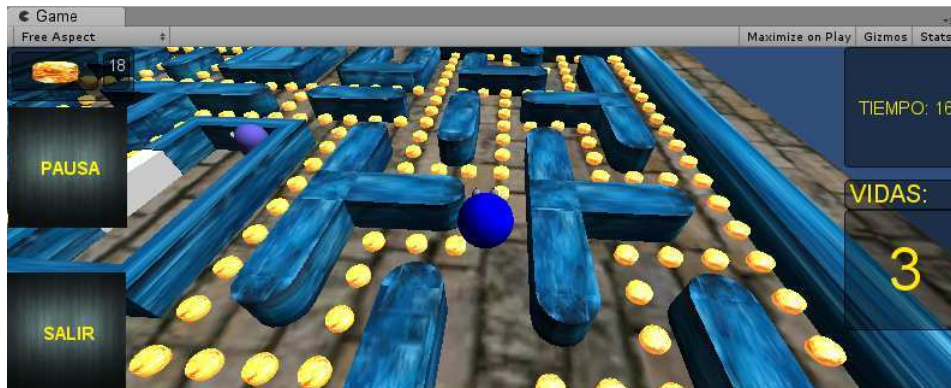


Ilustración 160: Vista en primera persona

```
- if(Input.GetKeyDown("2")){  
    Camara.active = false;  
    CamaraPrincipal.active = false;  
    CamaraTerceraPersona.active = true;  
}
```

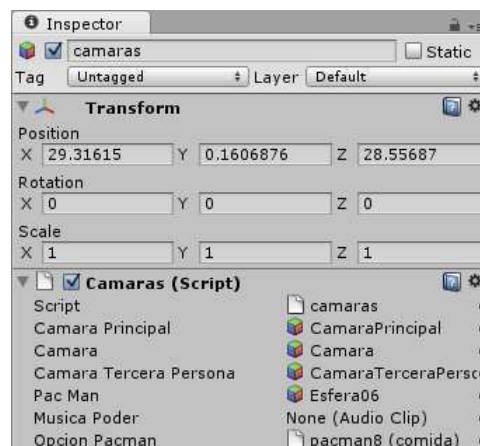


→ Si el usuario pulsa la tecla 3, se activa la tercera cámara y se desactivan las otras 2. Esta es la vista que se obtiene de la escena:



*Ilustración 161: Vista en tercera persona*

Se adjunta el script al objeto “cámaras” anteriormente creado y nos aparecen 3 variables nuevas en la vista del inspector:

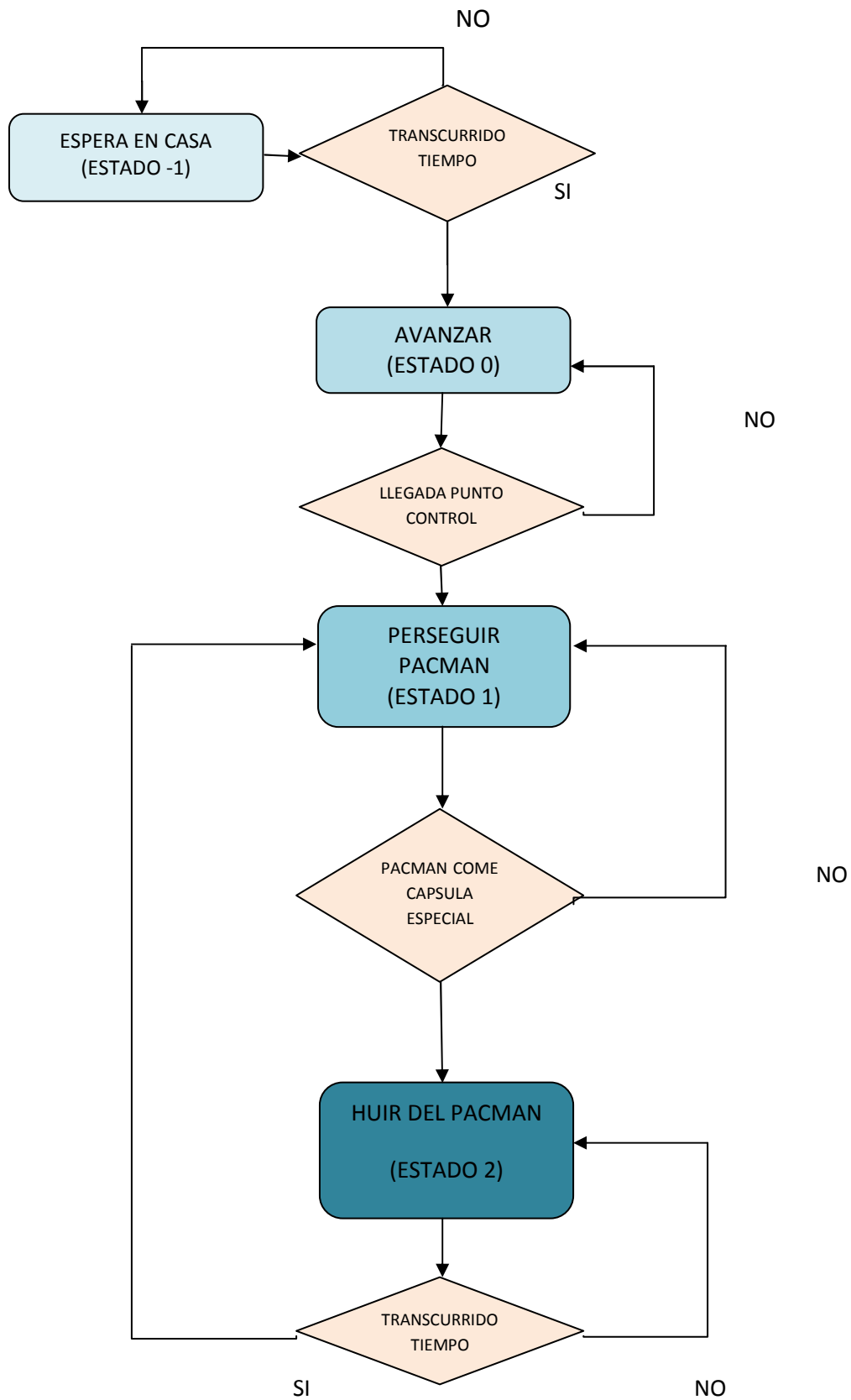


*Ilustración 162: Comportamiento de las cámaras*

A cada variable se le adjudica la cámara correspondiente, para ello se va a la vista de jerarquía, se selecciona la cámara y se arrastra hasta colocarla en la variable. Se guardan los cambios y se lanza el juego y se comprueba que las cámaras de la escena se comportan del modo definido en el script. Por defecto se ve la vista ofrecida por la cámara principal y según se van pulsando las teclas 1, 2 o 3 se cambia de una vista a otra sin que se pare el juego. Además los cambios de punto de vista no alteran la posición alcanzada en la escena y seguimos en el mismo punto.

## 20.2. Tutorial: Máquina de estados:

En este apartado se va a explicar la máquina de estados que se ha implementado para el movimiento de los enemigos y el script implementado para ello.





### Explicación de la máquina de estados:

Los fantasmas van a ir pasando de un estado a otro a medida que se cumplan ciertas condiciones. El control de las condiciones se hace mediante la función Update en la cual aparece:

`private var estado = -1;` → una variable de control de estado.

```
var xPacman = pacman.transform.position.x;  
var zPacman = pacman.transform.position.z;  
var xFantasma = controller.transform.position.x;  
var zFantasma = controller.transform.position.z;
```

→ Variables que representan las coordenadas X y Z del pacman y del fantasma.

Los estados por los cuales un fantasma puede pasar son los que siguen:

- ESTADO -1: El estado por defecto, el fantasma se encuentra en movimiento horizontal de un lado hacia otro dentro de su casa. Este movimiento se repite hasta que transcurra un tiempo determinado para cambiar de estado.

```
if(estado == -1){  
    if(controller.velocity.x == 0)  
        enAvant2 = -enAvant2;  
}
```

- ESTADO 0: El fantasma tiene un movimiento en vertical hasta llegar a un punto de control donde habrá salido completamente de “casa”. El punto de control es una coordenada concreta del eje z. Para que el fantasma pueda salir de su casa se tiene que modificar el objeto que representa la puerta.

`Puerta.collider.isTrigger = false;`  
Seguido se modifica el estado del fantasma.  
`estado = 1;`

```
if(estado == 0){  
  
    enAvant2 = transform.TransformDirection(Vector3(0,0,1));  
    controller.transform.position.x = 28.25;  
  
    if((zFantasma > 29) && (salida == 0)){  
        salida = 1;  
        puerta.collider.isTrigger = false;  
        estado = 1;  
    }  
}
```



- ESTADO 1: Se detecta la posición del Pacman para que el fantasma pueda realizar un movimiento de aproximación a este. En este estado el fantasma se va moviendo por el laberinto según el movimiento del pacman hasta llegar a una posición en la que tiene que quedarse quieto porque no encuentra salida. En este caso, transcurrido un tiempo de 50 segundos se pasa al estado 2.

```
if(((controller.velocity.x == 0) || (controller.velocity.z == 0)) && (salida == 1) && (estado == 1)){  
    //print("Entro en estado 1");  
    if(xFantasma > xPacman){ //El fantasma se encuentra más a la derecha que el pacman  
        if(zFantasma > zPacman){ // El fantasma se encuentra más arriba que el pacman  
  
            enAvant2 = transform.TransformDirection(Vector3(-1,0,-1));  
  
        }else if (zFantasma < zPacman){ //El fantasma se encuentra más abajo que el pacman  
  
            enAvant2 = transform.TransformDirection(Vector3(-1,0,1));  
        }else{  
            enAvant2 = transform.TransformDirection(Vector3(-1,0,0));  
        }  
    }else if(xFantasma < xPacman){ //El fantasma se encuentra más a la izquierda que el pacman  
        if(zFantasma > zPacman){ // El fantasma se encuentra más arriba que el pacman  
  
            enAvant2 = transform.TransformDirection(Vector3(1,0,-1));  
  
        }else if(zFantasma < zPacman){ //El fantasma se encuentra más abajo que el pacman  
  
            enAvant2 = transform.TransformDirection(Vector3(1,0,1));  
        }else{  
            enAvant2 = transform.TransformDirection(Vector3(1,0,0));  
        }  
    }else{  
        if(zFantasma > zPacman){ // El fantasma se encuentra más arriba que el pacman  
  
            enAvant2 = transform.TransformDirection(Vector3(0,0,-1));  
        }else if(zFantasma < zPacman){ //El fantasma se encuentra más abajo que el pacman  
  
            enAvant2 = transform.TransformDirection(Vector3(0,0,1));  
        }else{  
            enAvant2 = transform.TransformDirection(Vector3(0,0,0));  
        }  
    }  
    }  
  
    if((controller.velocity.x == 0) && (controller.velocity.z == 0)){  
        parado++;  
    }  
  
    if(parado > 50)  
        estado = 2;  
}
```





- **ESTADO 2:** En este estado, el fantasma se mueve alejándose de la posición del pacman. Este movimiento se realiza de manera controlada mediante una variable “intentos” hasta que transcurran 50 intentos. Pasado ese tiempo se vuelve al estado 1 y se reinicializa la variable.

```
//Estado en que el enemigo se encuentra bloqueado
if((salida == 1) && (estado == 2)){
    parado = 0;
    intentos++;
    //print("Entro en estado 2 y " + intentos);

    if(xFantasma > xPacman){ //El fantasma se encuentra más a la derecha que el pacman
        if(zFantasma > zPacman){ // El fantasma se encuentra más arriba que el pacman

            enAvant2 = transform.TransformDirection(Vector3(1,0,1));

        }else if (zFantasma < zPacman){ //El fantasma se encuentra más abajo que el pacman

            enAvant2 = transform.TransformDirection(Vector3(1,0,-1));
        }else{
            enAvant2 = transform.TransformDirection(Vector3(1,0,0));
        }

    }else if(xFantasma < xPacman){ //El fantasma se encuentra más a la izquierda que el pacman
        if(zFantasma > zPacman){ // El fantasma se encuentra más arriba que el pacman

            enAvant2 = transform.TransformDirection(Vector3(-1,0,1));

        }else if(zFantasma < zPacman){ //El fantasma se encuentra más abajo que el pacman

            enAvant2 = transform.TransformDirection(Vector3(-1,0,-1));
        }else{
            enAvant2 = transform.TransformDirection(Vector3(-1,0,0));
        }

    }else{
        if(zFantasma > zPacman){ // El fantasma se encuentra más arriba que el pacman

            enAvant2 = transform.TransformDirection(Vector3(0,0,1));
        }else if(zFantasma < zPacman){ //El fantasma se encuentra más abajo que el pacman

            enAvant2 = transform.TransformDirection(Vector3(0,0,-1));
        }else{
            enAvant2 = transform.TransformDirection(Vector3(0,0,0));
        }

    }

    if(intentos > 50){
        //print("paso al estado 1");
        estado = 1;
        intentos = 0;
    }
}
```



### 20.3. Tutorial: Puertas invisibles:

El juego del Pacman se desarrolla dentro de un laberinto por lo cual cuando el personaje alcanza la salida tiene que volver a la entrada. En este apartado se va a explicar cómo se controlan los movimientos del pacman para que no salga del laberinto y en caso de llegar a uno de los 2 extremos de este último que vuelva a entrar por el otro.

Primero se definen puertas invisibles en cada uno de los extremos del laberinto. Las puertas invisibles son simplemente unos GameObject tipo cubo sin textura y con las medidas adecuadas para que ocupen toda la salida.



*Ilustración 163: Puertas invisibles*

Puesto que el pacman solo alcanzara estas 2 posiciones si está comiendo las capsulas, el comportamiento que conlleva se define en el script de comida.

El script de comida.js es el que permite controlar el contacto del pacman con los diversos elementos que componen el juego, capsulas, fantasmas, puertas invisibles, el cambio de nivel, etc...

Cuando el pacman alcanza la puerta salida debe ser tele-transportado al otro extremo detrás de la puerta entrada. La parte de código que realiza este movimiento es la que sigue y consiste en un simple cambio de coordenadas.

```
}else if(objetoInfo.gameObject.name == "salida"){  
    comecoco1.transform.position.x = 30.9;  
    comecoco1.transform.position.y = 0.3885764 ;  
    comecoco1.transform.position.z = 28.48 ;  
}
```



Lo mismo ocurre en el otro sentido, si el pacman alcanza la puerta de entrada este vuelve a entrar por la puerta de salida con un cambio de coordenadas.

```
}else if(objetInfo.gameObject.name == "entrada"){
    comecoco1.transform.position.x = 25.7;
    comecoco1.transform.position.y = 0.3885764;
    comecoco1.transform.position.z = 28.48;
```

## 20.4. Tutorial: Introducir un botón de interfaz:

En este tutorial se va a explicar cómo se ha introducido el botón de “pausar/continuar” del pacman. Se crea un script con la función OnGUI:

```
- function OnGUI () {
    //estilo.fontSize = 20;

    GUILayout.BeginHorizontal("box",GUILayout.Width(Screen.width/8), GUILayout.Height(45));
    GUILayout.Box(puntos,GUILayout.Height(puntos.height),GUILayout.Width((Screen.width/8) * 0.45));
    GUILayout.Box(variable.score+"",GUILayout.Height(puntos.height), GUILayout.Width((Screen.width/8) * 0.35));
    GUILayout.EndHorizontal();

    if(GUILayout.Button (pausaPlay,estilo, GUILayout.Height(100),  GUILayout.Width(Screen.width/8))) {

        if(tipoBoton == 0){
            movPacman.vitesseMouvement = 0;
            movPacman.vitesseRotation = 0;
            movEnemigo1.avance = 0;
            movEnemigo2.avance = 0;
            movEnemigo3.avance = 0;
            movEnemigo4.avance = 0;
            pausaPlay = "CONTINUAR";
            tipoBoton = 1;
            musica.audio.mute = true;

        }else if(tipoBoton == 1){
            movPacman.vitesseMouvement = 4;
            movPacman.vitesseRotation = 3;
            movEnemigo1.avance = 0.5;
            movEnemigo2.avance = 0.5;
            movEnemigo3.avance = 0.5;
            movEnemigo4.avance = 0.5;
            pausaPlay = "PAUSA";
            tipoBoton = 0;
            musica.audio.mute = false;

        }
    }
}
```

### Explicación del código:

if (GUILayout.Button (pausaPlay,estilo, GUILayout.Height(100),  
GUILayout.Width(Screen.width/8))) → Permite crear el botón definiendo su  
altura y anchura, el if permite indicar si se ha pulsado el botón.

```
if(tipoBoton == 0){
    movPacman.vitesseMouvement = 0;
    movPacman.vitesseRotation = 0;
```



```
movEnemigo1.avance = 0;
movEnemigo2.avance = 0;
movEnemigo3.avance = 0;
movEnemigo4.avance = 0;
pausaPlay = "CONTINUAR";
tipoBoton = 1;
musica.audio.mute = true; → Este código se ejecuta si el usuario pulsa
“Pausa” entonces se ponen todas las variables de movimiento del pacman y de
los enemigos a cero de esta manera se para el juego en el punto en el que lo ha
dejado el jugador. El botón expone la opción de “Continuar”.
```

```
else if(tipoBoton == 1){
    movPacman.vitesseMouvement = 4;
    movPacman.vitesseRotation = 3;
    movEnemigo1.avance = 0.5;
    movEnemigo2.avance = 0.5;
    movEnemigo3.avance = 0.5;
    movEnemigo4.avance = 0.5;
    pausaPlay = "PAUSA";
    tipoBoton = 0;
    musica.audio.mute = false; → Cuando el usuario pulsa el botón “Continuar”
se reanuda el juego en el punto en el que se había dejado.
```

Finalmente se obtiene un botón de interfaz como el que se ve en la imagen. Primero aparece como “Pausa” mientras se está jugando y cuando el usuario pausa el juego se cambia el mensaje a “Continuar”.



*Ilustración 164: El botón pausar/continuar*

## 20.5. Tutorial: Cambio de propiedades en ejecución:

En este apartado se va a explicar cómo se cambian las propiedades de un objeto en ejecución. Cuando pacman come una capsula grande, esto le permite ser inmune a los fantasmas durante un tiempo definido, este poder se refleja con el cambio de color del pacman que pasa de ser amarillo al color azul.

Hay un script llamado “comida” y definido para la recogida de las capsulas por el pacman, en el cual se puede ver todo el procedimiento de comida, incremento del score y todo lo que conlleva. En un script que se lanza



en paralelo llamado “cámaras” y que permite el control del movimiento de las cámaras se implementan las siguientes líneas de código:

```
if(opcionPacman.opcionEnemigo == 1){  
    //audio.PlayOneShot(musicaPoder);  
    pacman.renderer.material.color = Color.blue;  
    if(contadorPoderPacman < 400){  
        contadorPoderPacman++;  
    }else{  
        opcionPacman.opcionEnemigo = 0;  
        contadorPoderPacman = 0;  
        //Cambio de color del Pacman a su color  
        pacman.renderer.material.color = Color.yellow;  
    }  
}
```

Explicación del código:

`pacman.renderer.material.color = Color.blue;` → Cuando el pacman se puede comer los fantasmas pasa al estado 1 de la máquina de estados en ese caso se cambia su color al azul:

```
if(contadorPoderPacman < 400){  
    contadorPoderPacman++;  
}
```

→ Cuando el pacman pasa al color azul, esta posición de poder solo le dura un tiempo concreto que se controla con este contador que se va incrementando hasta llegar a 400.

```
opcionPacman.opcionEnemigo = 0;  
contadorPoderPacman = 0;
```

`pacman.renderer.material.color = Color.yellow;` → Se vuelve al estado 0 de nuevo en el que el pacman no se puede comer los fantasmas y entonces su color pasa a ser de nuevo amarillo.

## 20.6. Tutorial: Pasar de una escena a otra:

Tal como se ha visto hasta ahora, en unity se trabaja a nivel de escenas por lo cual tiene que existir un mecanismo simple que permita el paso de una escena a la siguiente. Este procedimiento es muy simple y se hace utilizando una línea de código:

```
if(GUI.Button(new Rect(0, Screen.height-100, Screen.width/8, 100), "SALIR", stile)){  
    Application.LoadLevel("MainPacman");  
}
```

Este código hace que cuando el usuario pulsa el botón “salir” se carga la escena que representa el menú del videojuego, esta escena se llama MainPacman y tiene el siguiente aspecto:



*Ilustración 165: Menú del pacman*

En otros casos el paso de una escena a la siguiente se hace automáticamente sin pulsar ningún botón. Cuando el pacman se ha comido las 296 capsulas, se carga automáticamente el nivel 2 del juego. Del mismo modo cuando en el nivel 2 el pacman ya se ha comido las 280 capsulas se carga la escena que representa el final ganador del pacman.

```
if(Application.loadedLevelName=="pacman1"){  
    if(score == 296)  
        Application.LoadLevel("pacman2"); //Llamada a la pantalla del nivel 2  
}else if(Application.loadedLevelName=="pacman2"){  
    if(score == 280)  
        Application.LoadLevel("FinPacmanBien"); //Llamada a la pantalla fin juego bien  
}
```

## 20.7. Tutorial: Cerrar la aplicación:

El cierre de la aplicación igual que en los demás casos es muy simple y se hace con una línea de código. Cuando el usuario pulsa el botón “Cerrar Pacman” entonces se ejecuta la función `Application.Quit()` que permite cerrar la aplicación.

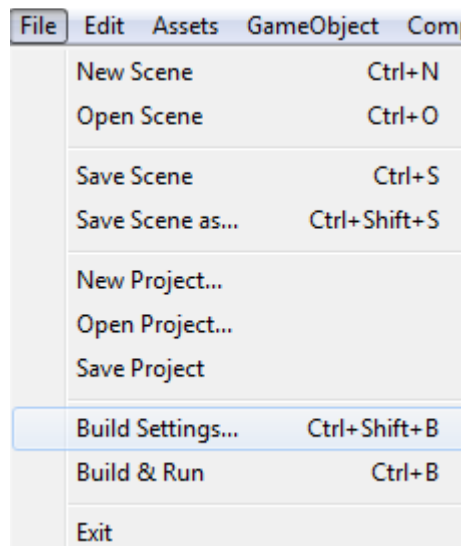
```
if(GUI.Button(new Rect(Screen.width / 2 - 100, Screen.height/2+100, 200, 100), "CERRAR PACMAN")){  
    Application.Quit();  
}
```





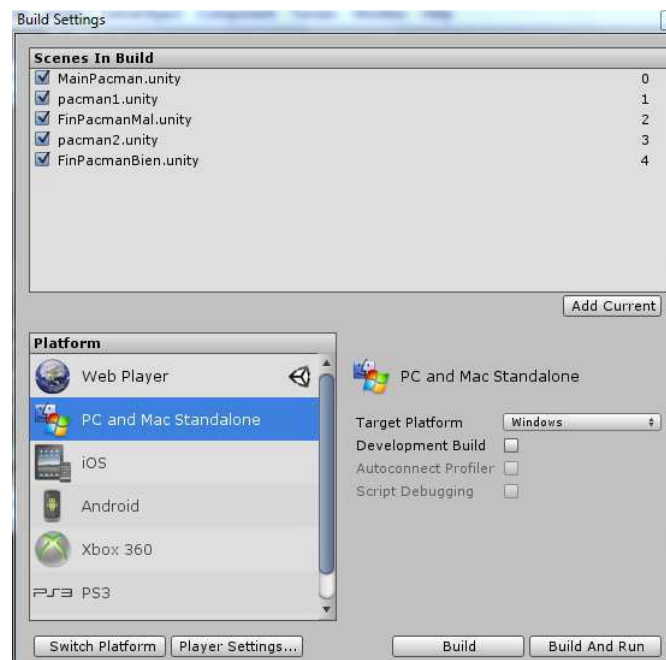
## 21. COMPILACIÓN PARA WEBPLAYER:

La generación de una versión del juego “IPacman” para Webplayer se realiza de manera sencilla y rápida. Se accede a “File>Build Setting”:



*Ilustración 166: Build Settings*

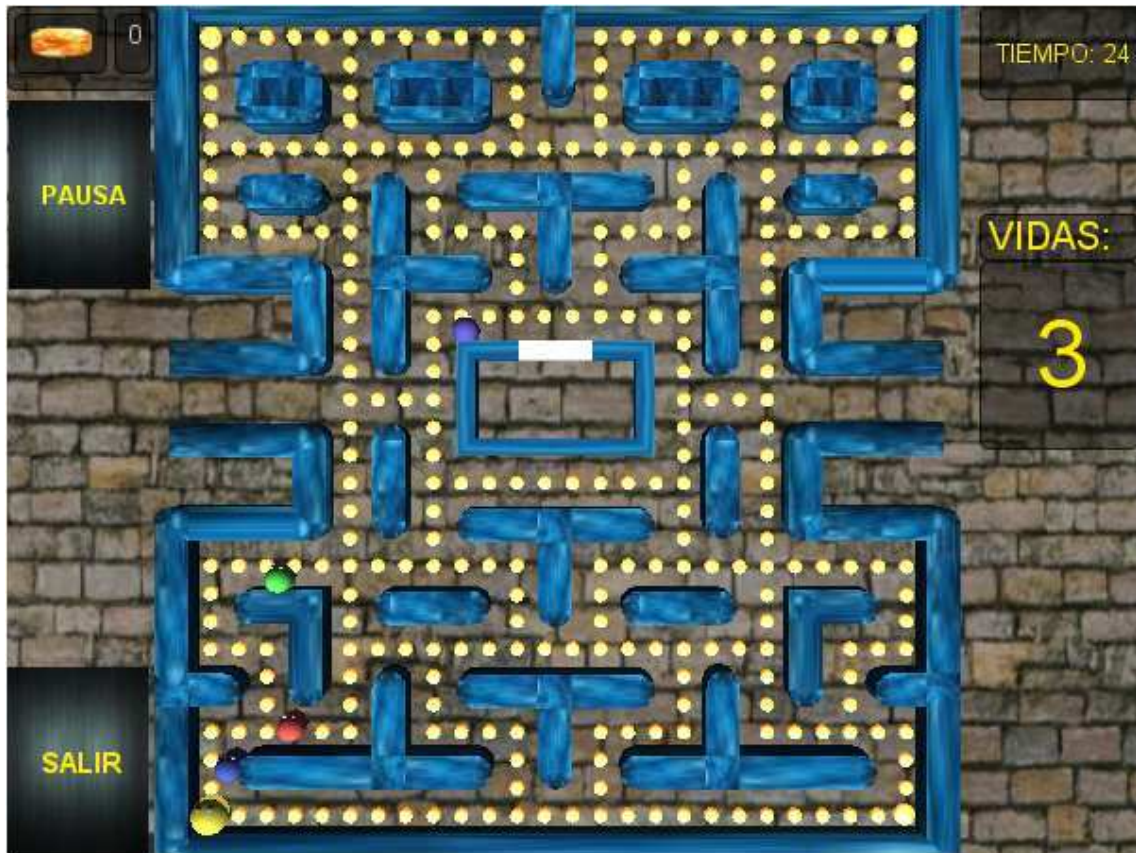
Una vez realizados estos pasos, aparece la pantalla que sigue y en la que se puede observar que unity no da opción a compilar el juego para varias plataformas PC, IOS, Android, Xbox 360, etc aunque en la mayoría de los casos es previo pago.



En el caso de Web Player es totalmente gratuito y solo hay que seleccionar esta opción y hacer clic en “Build” entonces se genera una carpeta llamada “ Webplayer”



dentro del proyecto con el html y los archivos necesarios para su ejecución. Se lanza el html y ya tenemos nuestro juego en una página Web.





## 22. GESTIÓN DEL PROYECTO.

### 22.1. Diagramas de Gantt.

Este apartado está dedicado a la planificación del proyecto. Primero la planificación inicial hecha antes de la realización del proyecto fin de carrera y basada en la experiencia adquirida con proyectos anteriores. Seguido la planificación real que se ha ido actualizando a lo largo de la realización del proyecto.

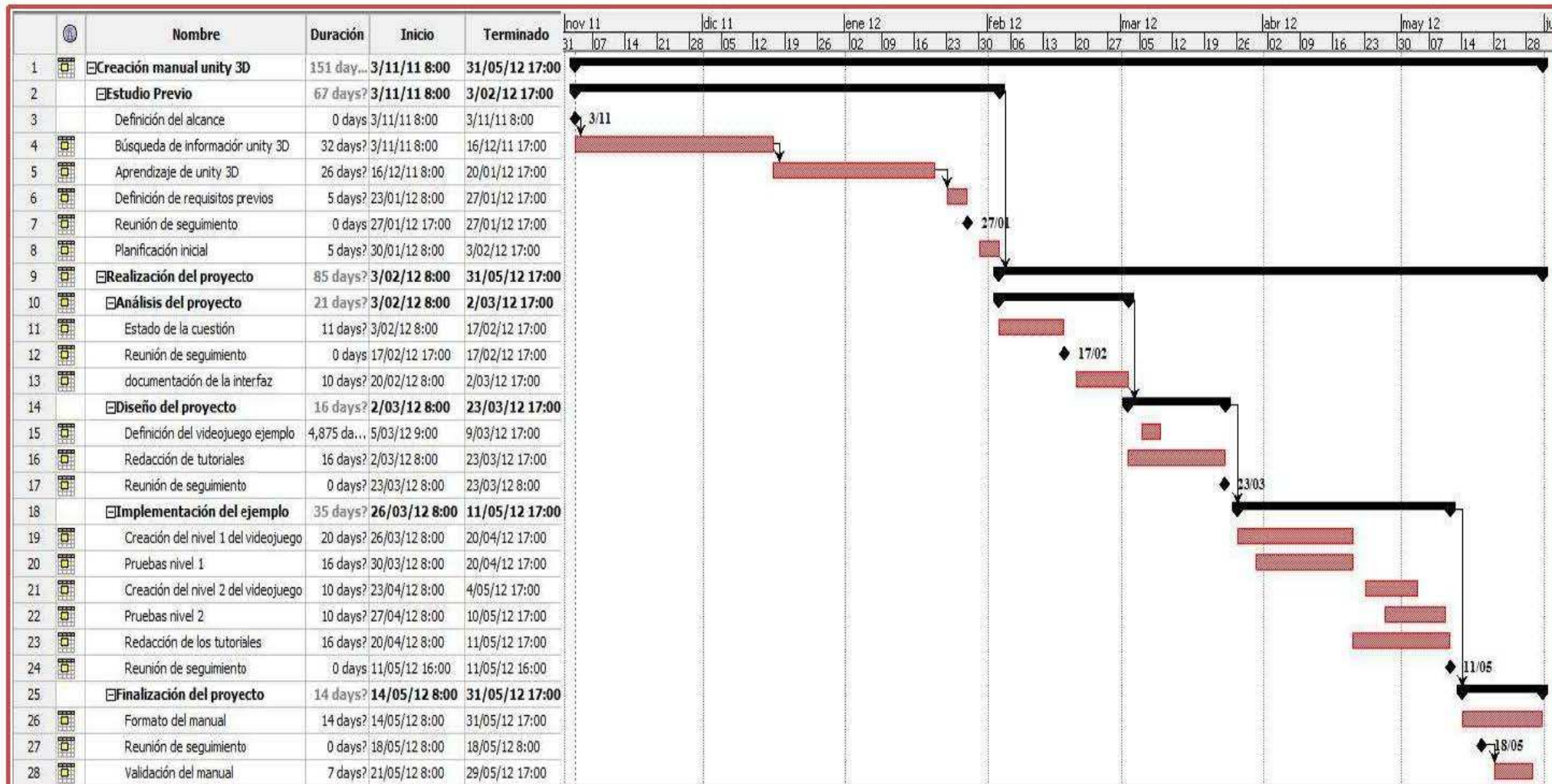
En el diagrama van a aparecer las diferentes fases en las que se ha dividido el proyecto y las tareas correspondientes a cada fase junto con el tiempo estimado o real para llevar a cabo la tarea.

Para la realización de la planificación se va a utilizar un diagrama de Gantt con el OpenProj.

### 22.2. Planificación estimada.

Esta planificación estimada se ha realizado al comienzo del proyecto y se basa en la experiencia adquirida en proyectos anteriores. Se ha hecho una división del proyecto en fases y tareas que se han visto necesarias y el tiempo que se espera consumir para la culminación de esas tareas.





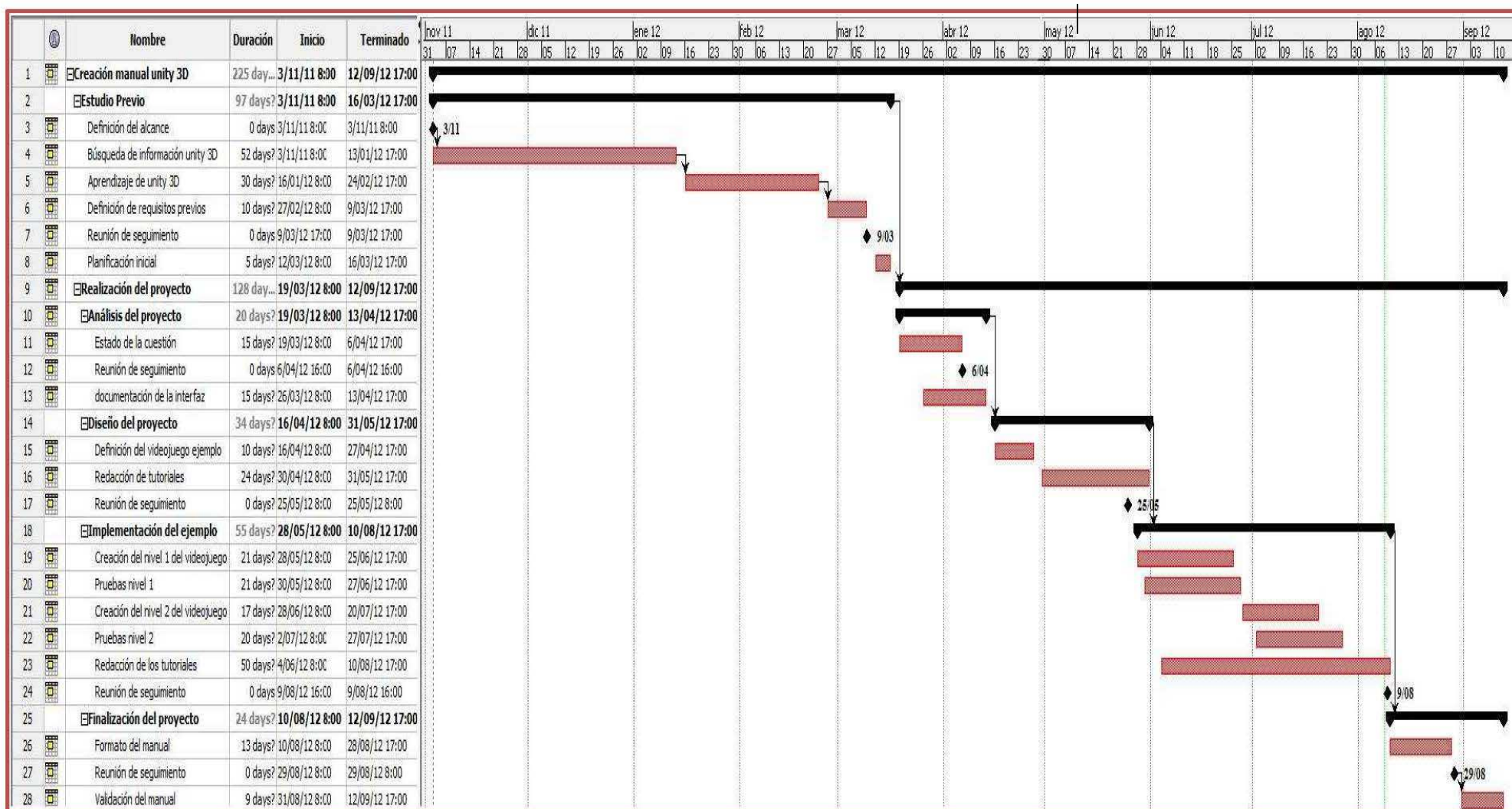


El proyecto se ha dividido en 2 grandes fases que engloban lo siguiente:

- El estudio previo:
  - En él se ha definido el alcance que tiene que tener el manual.
  - La parte de búsqueda de información, documentación y toma de contacto con el motor.
  - Una macro-planificación inicial.
- El proyecto de realización:
  - Análisis del proyecto: documentación sobre los motores y unity en concreto.
  - Diseño e implementación del proyecto: definición del tipo de videojuego que se va a crear como ejemplo y su implementación.
  - Finalización del proyecto: últimos ajustes en el manual, su formateado y mejoras.

### 22.3. Planificación real:

A lo largo de la realización del proyecto, se ha ido actualizando la planificación inicial conforme al progreso real de las tareas para obtener el diagrama de Gantt que aparece a continuación:





## 22.4. Análisis de la planificación:

El proyecto se ha realizado a lo largo de 255 días con una media de trabajo diaria de 4 horas diarias.

Tal como se puede observar en los anteriores diagramas de Gantt, la planificación estimada y la planificación real difieren en 104 días. Para llevar a cabo el proyecto se han consumido 3 meses más de los planificados al inicio del proyecto.

Los motivos de este desfase en los tiempos se debe a:

- Dificultad para encontrar documentación:
  - Unity 3D es un motor que no lleva mucho tiempo en el mercado, esto hace que aún no exista mucha documentación.
- Evolución del caso práctico:
  - Un cambio en la elección del videojuego por dificultades temporales.
  - Complicaciones surgidas a lo largo del desarrollo.
- Experiencia nula en videojuegos y por consiguiente en los motores.



## 23. PRESUPUESTO

En esta sección, se va a calcular el presupuesto que haría falta en caso de llevar a cabo el proyecto de creación de una manual de usuario de Unity 3D a vistas de comercializarlo.

En este presupuesto se van a tomar en cuenta todos los recursos ya sean humanos o materiales y todas las fases, aprendizaje, creación de un mini-videojuego, los correspondientes test así como la redacción del manual en sí.

### AUTOR:

Iman Ouazzani

### DEPARTAMENTO:

Departamento de informática.

### DESCRIPCIÓN DEL PROYECTO.

**Título:** MANUAL PARA LA CREACIÓN DE VIDEOJUEGOS MEDIANTE EL MOTOR UNITY 3D.

**Duración:** 6 meses.

**Tasa de costes indirectos:** 20%.

### PRESUPUESTO DETALLADO DEL PROYECTO.

#### RECURSOS HUMANOS:

En esta tabla se especifican los costes relativos a las personas que involucradas en el proyecto.

Persona involucrada	Categoría	Dedicación (meses)	Coste H/M	Coste €
Juan Peralta Donate	Ingeniero Senior	1	4.289,54	4.289,54
Iman Ouazzani	Ingeniero	6	2.694,39	16.166,34
<b>Total</b>				<b>20.455,88</b>

*Tabla 5: Recursos humanos*



### RECURSOS MATERIALES:

A continuación se van a detallar los equipos utilizados para la realización de este proyecto y el coste correspondiente.

Descripción	Coste €	% Uso proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
ACER Aspire 5920	600,00	100	6	6	60,00
DELL Vostro V131	1.099,00	100	6	6	109,90
Disco duro externo	100,00	100	6	6	10,00
Multifunción Escáner + Impresora	300,00	100	6	6	30,00
<b>Total</b>					<b>209,90</b>

*Tabla 6: Recursos materiales*

### DESCRIPCIÓN DE LOS EQUIPOS:

- DELL Vostro V131- Ordenador para la creación de la documentación del proyecto:
  - Sistema operativo: Windows 7 Home Premium 64Bit
  - Tamaño:  $329.3 \times 237.6 \times 16.05$  – 21 milímetros
  - Peso: 1.83 kilos con batería de seis celdas
  - Procesador Intel Core i5 2410M Ultra bajo consumo a 2.30Ghz
  - Memoria: 4GB DDR3
  - Gráfica: Intel HD2000 integrada
  - Pantalla: 13.3 pulgadas antirreflejos con resolución  $1366 \times 768$  píxeles
  - Disco duro: 500GB (450 aprovechables de fábrica)
  - Baterías: 65Whr de seis celdas
  - Chips Mobile Intel® HM 67
  - Conjunto de chips: Mobile Intel® HM 67
  - Audio y altavoces:





- Altavoces, Conector para auriculares estéreo, Conector para micrófono y micrófonos digitales integrados, Cámara web de alta definición integrada de 1,0 megapíxeles.
  - Software Dell Webcam Central, Software SRS Premium Voice Pro.
- Unidad óptica : DVD+/-RW 8x externo opcional
- Alimentación: Batería de iones de litio de 6 celdas inteligente (65 W/h)
- Conectividad:
  - Conexión con cables integrada: Gigabit Ethernet (NIC 10/100/1000)
  - LAN inalámbrica: Intel® 802.11 b/g/n; Intel Centrino Advanced-N 6230, Dell 802.11 b/g/n;
  - Bluetooth: Bluetooth v3.0 + alta velocidad
  - Banda ancha móvil: Dell 5530/5630 (requiere la contratación de servicios de operador de red móvil)
- Puertos, ranuras y chasis: Conector de red (RJ45), USB 3.0 (2), Puerto combinado USB 2.0/eSATA (1), Conector para micrófono, salida de auriculares/altavoces, Lector de tarjetas 8 en 1 HDMI Conector de vídeo VGA de 15 patillas.
- Disco duro externo.- Disco duro externo con conexión USB para realizar copias de seguridad del proyecto.
  - Modelo: FreeCom.
  - Capacidad: 80 Gb de almacenamiento de datos.
- Multifunción impresora + escáner.- Utilizada para la impresión y escáner de la documentación necesaria para el proyecto.
  - Modelo: HP photosmart C4180.
- DELL Aspire 5920- Ordenador para la creación del videojuego prototipo.
  - Pantalla: 15,4" TFT LCD WXGA (1280 \* 800 píxeles) con refresco de pixeles cada 8 milisegundos.
  - Peso y dimensiones: 364 \* 270.2 \* 43.7 mm (30.8 unidad principal) 2,8 kilogramos.
  - Procesador: Intel Core Duo T7300 Santa Rosa 2GHz



- Memoria RAM: 2GB (1024 – 1024)
- Disco duro: 160 GB
- Sistema operativo: Windows Vista Home Premium
- Controles:
  - Teclado QWERTY Touchpad
  - Teclas de acceso directo: E-mail, navegador web, WiFi, Bluetooth
  - Controles táctiles de reproducción y acceso directo a NTI CD&DVD Maker y Acer Arcade Deluxe.
- Conectividad y entradas compatibles:
  - 4 puertos USB
  - Puerto HDMI (soporta HDCP)
  - Ranura 5 \* 1 para tarjetas de memoria (SD/MMC/MS/MS PRO/xD)
  - Salida VGA
  - Salida S-Video/TV
  - Salida auriculares y altavoces
  - Salida de línea telefónica y Conexión Ethernet
  - Entrada de micrófono
  - Sincables: WiFi y Bluetooth
- Tarjeta gráfica:
  - NVIDIA® GeForce® Go X8600 GT 1024 MB (Turbocaché)
  - Memoria de video 256 MB
- Lector: lectura y escritura de CD y DVD (lector HD DVD opcional)
- Cámara Interna: CrystalEye, 30 imagines por segundo.

#### COSTES AÑADIDOS:

En esta tabla, se detallan otros costes relativos a la realización del proyecto.

Descripción	Empresa	Coste imputable
Funjible		200,00
Viajes		100,00
Licencia Windows 7 Professional 64 bits: 2	Microsoft Store	599,98
Licencia Microsoft Office 2007 Basic	Microsoft Store	149,99
<b>Total</b>		<b>899,98</b>

*Tabla 7: Costes añadidos*





- Fungibles.- Material de oficina utilizado en el desarrollo del proyecto fin de carrera y el presente documento.
- Transporte.- Coste de desplazamiento para la realización de las reuniones de seguimiento.
- Licencias.- Licencias de software necesarias para los equipos utilizados en el proyecto.

#### PRESUPUESTO TOTAL DEL PROYECTO:

En la siguiente tabla se recogen todos los costes detallados anteriormente y por consiguiente el coste total del proyecto.

Presupuesto Costes Totales	
Personal	20.456
Amortización	210
Subcontratación de tareas	0
Costes de funcionamiento	1.050
Costes Indirectos	4.343
<b>Total</b>	<b>26.059</b>

*Tabla 8: Presupuesto total del proyecto*



## 24. COMERCIALIZACIÓN DEL PROYECTO:

Existe la posibilidad de comercializar un videojuego creado con el motor Unity 3D de diversas maneras. Primero estaría las opciones existentes para la mayoría de los videojuegos creado con cualquier motor 3D:

- Niveles exclusivos: consistiría en comercializar unos niveles exclusivos para algún juego creado con todos los elementos de este y ampliando sus propiedades.
- Elementos exclusivos: dar a acceso a unos elementos concretos en un videojuego previo pago para la compra de claves. La introducción de estas claves en el videojuego permitiría su uso en el juego. Los elementos pueden ser cualquier tipo de objeto del juego (armas, personajes, vehículos, etc.).
- Mejoras de las propiedades del juego: se pueden mejorar y actualizar un videojuego para comercializarlo. Por ejemplo se pueden dotar las armas con más poder, hacer mejoras en los niveles de vida del personaje, etc.

Además no hay que olvidar que Unity 3d es un motor multiplataforma que permite la compilación de los juegos creados para androide, iphone, wii, etc. Por lo cual también se pueden comercializar los juegos para esas plataformas, subiendo el juego por ejemplo al market de android.



## 25. CONCLUSIONES:

Llegados a este punto y echando la vista atrás se puede comprobar si se han cumplido los objetivos del proyecto y hacer una valoración global.

Finalizado el proyecto, se puede concluir que los motores de videojuegos son herramientas muy útiles que facilitan el trabajo de los creadores permitiéndoles centrarse en los aspectos creativos y no tanto en los aspectos matemáticos, físicos, etc.

Unity 3D es uno de los motores más recientes en el mercado, es completo y muy potente. Además ha ido adaptándose a los avances versión a versión y a las nuevas tendencias como la creación de juegos para diferentes soportes de allí que sea multiplataforma.

Esta característica es una de las ventajas más importantes de unity 3D. En efecto, las tendencias actuales hacen que un juego tenga que poder jugarse en una consola de salón, en el móvil o en el PC para adaptarse a unos consumidores muy exigentes y una competencia incansable. Como hemos podido comprobar en el manual, la compilación del videojuego ejemplo para Web se ha realizado de una manera muy rápida y sencilla.

Para llevar a cabo el proyecto se ha utilizado la versión gratuita de Unity 3D que no por ser gratuita es menos potente como se ha podido comprobar. Unity ofrece un entorno simple e intuitivo que permite la creación de un juego muy rápidamente y puesto que no es necesario tener conocimientos profundos de programación cualquiera lo puede utilizar con unas nociones básicas.

Por último, hay que recodar dos puntos más, por un lado que Unity no solo permite la creación de videojuegos, también es una herramienta muy apreciada por los arquitectos y diseñadores puesto que permite crear entornos 3D de manera sencilla y rápida, y por otro lado, Unity 3D está en evolución constante y cada cierto tiempo se presenta una nueva versión que incluye mejoras y avances.



## 26. LÍNEAS FUTURAS:

Se va a hacer una reflexión sobre las posibles líneas a investigar y a realizar más adelante. Puesto que la realización de este proyecto tenía tiempo y recursos limitados quedan muchas cosas por descubrir de Unity 3D:

- Aprovechamiento de numerosas funcionalidades no explotadas de Unity3d.
- Adaptación del videojuego a todas las plataformas que ofrece Unity, compilar el juego para androide por ejemplo.
- Existen versiones de Unity dedicadas exclusivamente para un tipo de plataforma como Iphone que se podría utilizar para la creación de un videojuego.
- El uso de diversos asset y paquetes que ofrece Unity como los “Standard Asset Movil”.
- Mejorar la inteligencia artificial de los enemigos, que sus movimientos sean más inteligentes y de esta manera que sea más difícil para el nivel.
- Investigar las herramientas que ofrece Unity para la creación de entornos de diseño arquitectónico.



## 27. REFERENCIAS:

- [1] Página oficial de Unity 3d: <http://unity3d.com/unity/>
- [2] Página oficial de Unity España: <http://www.unityspain.com/>
- [3] Página para búsqueda de videos sobre Unity 3D: <http://www.youtube.com/>
- [4] Página con un mini manual de Unity: [http://cronicasdenix.hostzi.com/?page\\_id=56](http://cronicasdenix.hostzi.com/?page_id=56)
- [5] Página para importar texturas variadas: <http://www.cgtextures.com/>
- [6] Página de la comunidad Unity 3d Francia:  
<http://www.unity3d-france.com/unity/category/tutoriels/tutoriels-videos-tutoriels/>
- [7] foro con definición del comportamiento de los fantasmas:  
<http://www.desadaptados.net/video-juegos/descubre-la-personalidad-de-los-fantasmas-de-pacman/>
- [8] Página para importar sonidos:  
[http://efectos-de-sonido.anuncios-radio.com/gratis/index.php?option=com\\_weblinks&catid=68&Itemid=4](http://efectos-de-sonido.anuncios-radio.com/gratis/index.php?option=com_weblinks&catid=68&Itemid=4)
- [9] Página con un manual de Unity 3D:  
<http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/ProgramacionVideoJuegos/Unity3D/paso2b.html>
- [10] Página con video tutoriales de Unity 3D:  
<http://www.spotnik-tv.com/dotclear/index.php?Tutos-unity-3d>
- [11] Página con información de Unity 3D:  
<http://as3.miguelmoraleda.com/es/tag/unity3d/>
- [12] La wiki: [http://en.wikipedia.org/wiki/Unity\\_\(game\\_engine\)](http://en.wikipedia.org/wiki/Unity_(game_engine))
- [13] Página no oficial de los usuarios Unity Francia:  
<http://www.unity3d-france.com/unity/>
- [14] Foro en el que se habla de Unity 3D:  
<http://forum.canardpc.com/threads/42472-Cr%C3%A9er-son-jeu-avec-Unity3D-!>
- [15] Ejemplos de proyectos realizados con Unity 3D:  
[http://www.youtube.com/watch?v=0XQHeHxJU\\_w](http://www.youtube.com/watch?v=0XQHeHxJU_w)
- [16] Página de contenido variado con una sección dedicada a Unity 3D:  
<http://www.5axe.com/blog/2010/07/21/creer-un-jeu-video-pour-toutes-les-plateformes-avec-unity/>
- [17] Listado de videojuegos creados con Unity 3D:  
<http://unity3d.com/gallery/made-with-unity/game-list>
- [18] Presentación de un libro para crear videojuegos con Unity 3D:  
<http://www.pearson.fr/livre/?GCOI=27440100039450>



- [19] Listado de las mejoras aportadas a la versión 3 de Unity:  
<http://www.macgeneration.com/news/voir/146571/gdc-unity-enclenche-la-troisieme>
- [20] tutoriales sobre Unity: <http://unity3d.com/support/resources/tutorials/>
- [21] Wiki de los motores de videojuegos: [http://fr.wikipedia.org/wiki/Moteur\\_de\\_jeu](http://fr.wikipedia.org/wiki/Moteur_de_jeu)
- [22] Artículo sobre el mercado de videojuegos:  
<http://www.01net.com/editorial/564881/2015-les-jeux-video-pourraient-valoir-plus-de-60-milliards-d-euros/>
- [23] Historia del Pacman: <http://www.pac-man.fr/histoire-pacman.php>
- [24] Videos tutoriales en YouTube: <http://www.youtube.com/watch?v=i89IIyqM9VE>
- [25] Página dedicada a los creadores de videojuegos:  
<http://www.games-creators.org/wiki/Accueil>
- [26] Listado de los motores de videojuegos:  
[http://fr.wikipedia.org/wiki/Liste\\_de\\_moteurs\\_de\\_jeu](http://fr.wikipedia.org/wiki/Liste_de_moteurs_de_jeu)
- [27] Página latina con un apartado dedicado a Unity:  
<http://www.ecured.cu/index.php/Unity3D>
- [28] Página dedicada a la creación de videojuegos:  
<http://www.creation-jeux-video.com/les-moteurs-de-jeu-ou-moteurs-3d/>
- [29] Videos tutoriales de Unity 3D:  
<http://formation-facile.fr/unity35.php>
- [30] Tutoriales Web sobre Unity:  
<http://webtutoriaux.com/tutoriaux-Unity3D.html>
- [31] Página para importar texturas: <http://www.cgtextures.com/>
- [32] Comportamiento de los fantasmas:  
<http://www.desadaptados.net/video-juegos/descubre-la-personalidad-de-los-fantasmas-de-pacman/>
- [33] Manual de referencias Unity: [http://cronicasdenix.hostzi.com/?page\\_id=56](http://cronicasdenix.hostzi.com/?page_id=56)
- [34] Definición de movimientos de enemigos:  
<http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/ProgramacionVideoJuegos/Unity3D/paso2b.html>
- [35] Ejemplo de proyectos unity:  
<http://unity3d.com/support/resources/example-projects/>
- [36] Página de desarrolladores de videojuegos:  
<http://www.genbetadev.com/herramientas/unity-3d-desarrollo-de-videojuegos-para-ios-y-android-gratis-hasta-el-8-de-abril>



[37] Tutorial de unity 3D:

<http://trinit.es/unity/tutoriales/10%20-%20Shaders%20Tutorial/10%20-%20Shaders%20Tutorial.pdf>

[38] Página de un usuario de unity 3D, tutoriales en video:

<http://fenixdiscom.ekiwi.es/index.php/unity.html>

[39] Manual de iniciación en unity: “Empezando con unity 3D”

<http://es.scribd.com/doc/49807377/2/INTRODUCIENDO-UNITY3D>

[40] Tutorial de terrenos en unity:

<http://es.scribd.com/doc/62192939/Tutorial-Terrenos-de-Unity-3D>

[50] Tutorial de arboles en unity: “Unity Tree”

<http://docs.unity3d.com/Documentation/Components/terrain-Trees.html>

[51] Tutorial de audio en unity:

[http://cronicasdenix.hostzi.com/?page\\_id=56](http://cronicasdenix.hostzi.com/?page_id=56)



## 28. ANEXOS:

### 28.1. Glosario de términos:

En la tabla que sigue se van a explicar los términos y acrónimos que han sido utilizados en este documento:

<b>Término</b>	<b>Definición</b>
2D	Dos dimensiones
3D	Tres dimensiones
Game Engine (motor de videojuegos)	Motor de videojuegos. Conjunto de herramientas proporcionadas para el desarrollo de videojuegos.
IA	Se denomina inteligencia artificial a la capacidad de razonar de un agente no vivo. Se le adjudica al objeto un código que le dota de cierta inteligencia y hace que se comporte de una manera u otra en función de la situación en la que se encuentra.
Lenguaje de scripting	Tipo de lenguaje de programación que es generalmente interpretado.
Nivel	Modelo en 3D del mundo creado mediante el editor.
script	Una secuencia de instrucciones que se lanzan como un lote y se ejecutan una tras otra.
Javascript	JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, <sup>3</sup> basado en prototipos, imperativo, débilmente tipado y dinámico.
Collider	Un collider es la estructura que hace sólidos a los objetos y permite controlar las colisiones entre objetos.
Escena	La escena hace referencia a la conjugación de los elementos que conforman la imagen a saber: decorados, iluminación, texturas, sonido, etc.
GameObject	Representa un objeto, un elemento de la escena del juego y tiene propiedades y componentes.
Sistema de partículas (Particle System)	Se refiere a una técnica de computación gráfica que utiliza un gran número de pequeños elementos para simular algún fenómeno como el fuego o el humo.
Uniscite	Editor estándar de scripts en Unity 3D y predeterminado hasta la versión 3.4.
<u>MonoDevelop</u>	Es un entorno de desarrollo integrado libre y gratuito, diseñado primordialmente para C# y otros lenguajes .NET como Nemerle, Boo, Java
<u>Boo</u>	Boo es un lenguaje de programación orientado a objetos, de tipos estáticos para la Common Language Infrastructure con una sintaxis inspirada en Python y un énfasis en la extensibilidad del lenguaje y su compilador
<u>Asset</u>	Usualmente se denomina asset a cualquier recurso gráfico que se carga





	externamente
plugins	Es un complemento o una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.
Android	Android es un sistema operativo móvil basado en Linux, que junto con aplicaciones middleware está enfocado para ser utilizado en dispositivos móviles como teléfonos inteligentes, tabletas, Google TV y otros dispositivos.
3ds Max	Es un programa de creación de gráficos y animación 3D desarrollado por Autodesk y es uno de los programas de animación 3D más utilizado
Blender	es un programa informático multiplataforma, dedicado especialmente al modelado, animación y creación de gráficos tridimensionales
OpenGL	(Open Graphics Library) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.
Shaders	La tecnología shaders es cualquier unidad escrita en un lenguaje de sombreado que se puede compilar independientemente.
NVIDIA® PhysX®	Es GPU con una tecnología que añade un elemento de realismo en los juegos. Permite obtener efectos de física dinámicos impresionantes como las explosiones o el movimiento natural del agua y los personajes.
SDK	Un kit de desarrollo de software es un conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto.
API	(Application Programming) Interface es una serie de servicios o funciones que el sistema operativo ofrece al programador.
IDE	(Integrated development environment) es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien poder utilizarse para varios.
MonoDevelop	Es un entorno de desarrollo diseñado principalmente para C # y otros lenguajes NET. Permite a los desarrolladores escribir rápidamente aplicaciones de escritorio y Web ASP.NET en Linux, Windows y Mac OSX.
Framework	Es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de <i>software</i> concretos, con base a la cual otro proyecto de <i>software</i> puede ser más fácilmente organizado y desarrollado

Tabla 9: Glosario de términos



## 28.2. Guía del juego “Pacman”:

En este apartado se va a explicar el funcionamiento del juego “pacman” para que el usuario pueda jugar. El funcionamiento es bastante parecido a cualquier pacman estándar existente en el mercado pero simplificado.

Cuando el usuario lanza el juego se encuentra en la escena 1 que representa el primer nivel y consiste en un laberinto relleno de capsulas, con el personaje que es el pacman o comecocos y los fantasmas. Las capsulas pueden ser de 2 tipos, las normales que son las de tamaño más pequeño y representan la mayoría, las grandes que hay en las esquinas del laberinto y que son 4.

Si el pacman se come las capsulas pequeñas va sumando puntos en el contador e incrementa el score pero cuando se coma una capsula grande además de aumentar el contador se le cambia el color de amarillo a azul, este cambio dura un tiempo preciso durante el cual aunque sea alcanzado por un fantasma no solo no se ve afectado por estos sino que el fantasma vuelve a su punto de partida (su casa).

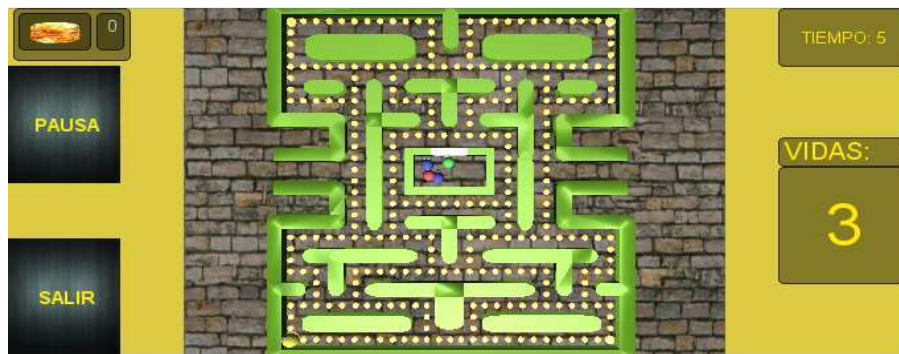
Los fantasmas son 4 azul, rojo, verde y morado. Al principio del juego están en su casa moviéndose de un lado a otro y van saliendo de su casa de manera escalonada cada cierto tiempo. Su comportamiento es bastante sencillo, van detectando la posición del pacman en todo momento y se mueven por el laberinto en dirección del pacman hasta llegar a algún punto del laberinto cerrado y que no pueda avanzar entonces se mueven en sentido contrario.

El pacman tiene que comerse todas las capsulas para poder alcanzar el siguiente nivel. Para ello dispone de un contador en la esquina superior izquierda que se va incrementando mostrándole al usuario el número de capsulas que ha comido. Dispone de un contador de tiempo que empieza 0 y se va incrementando en la parte superior derecha. En la parte derecha de la pantalla también aparece el botón vidas que empieza en 3 que es el número de vidas de las cuales dispone el pacman para acabar este nivel, este número se des-incrementa cada vez que el pacman es alcanzado por un fantasma y no está de color azul. Además el usuario dispone de la tecla pausar/continuar que le permite parar o seguir con el juego en la parte izquierda de la pantalla y la tecla salir que le permite abandonar el juego.





El nivel dos tiene un funcionamiento similar al nivel 1 con algunas modificaciones que han sido introducidas. Las diferencias aportadas al nivel dos con respecto al nivel son la modificación de algunas zonas del laberinto, así como la textura y el aumento del número de capsulas. Igual que en el caso anterior para poder superar este nivel, el pacman tiene que comerse todas las capsulas antes de que se le agoten las vidas para ello tiene que evitar los fantasmas.



Además durante todo el desarrollo del juego, el usuario puede ir cambiando la vista del juego mediante las teclas 1,2 y 3. Estas teclas le van a permitir tener una vista del juego desde arriba, en primera persona o tercera persona. Estas opciones le ofrecen al usuario una mejor visibilidad y diferentes puntos de vistas del juego.

Por último, el jugador puede probar la versión del Pacman compilada para Webplayer y así poder jugar online.

