UNITY y C#

DESARROLLO DE VIDEOJUEGOS

www

Desde **www.ra-ma.es** podrá descargar material adicional.

LUIS RUELAS

Ra-Ma®

UNITY y C#

DESARROLLO DE VIDEOJUEGOS

www

Desde **www.ra-ma.es** podrá descargar material adicional.

LUIS RUELAS

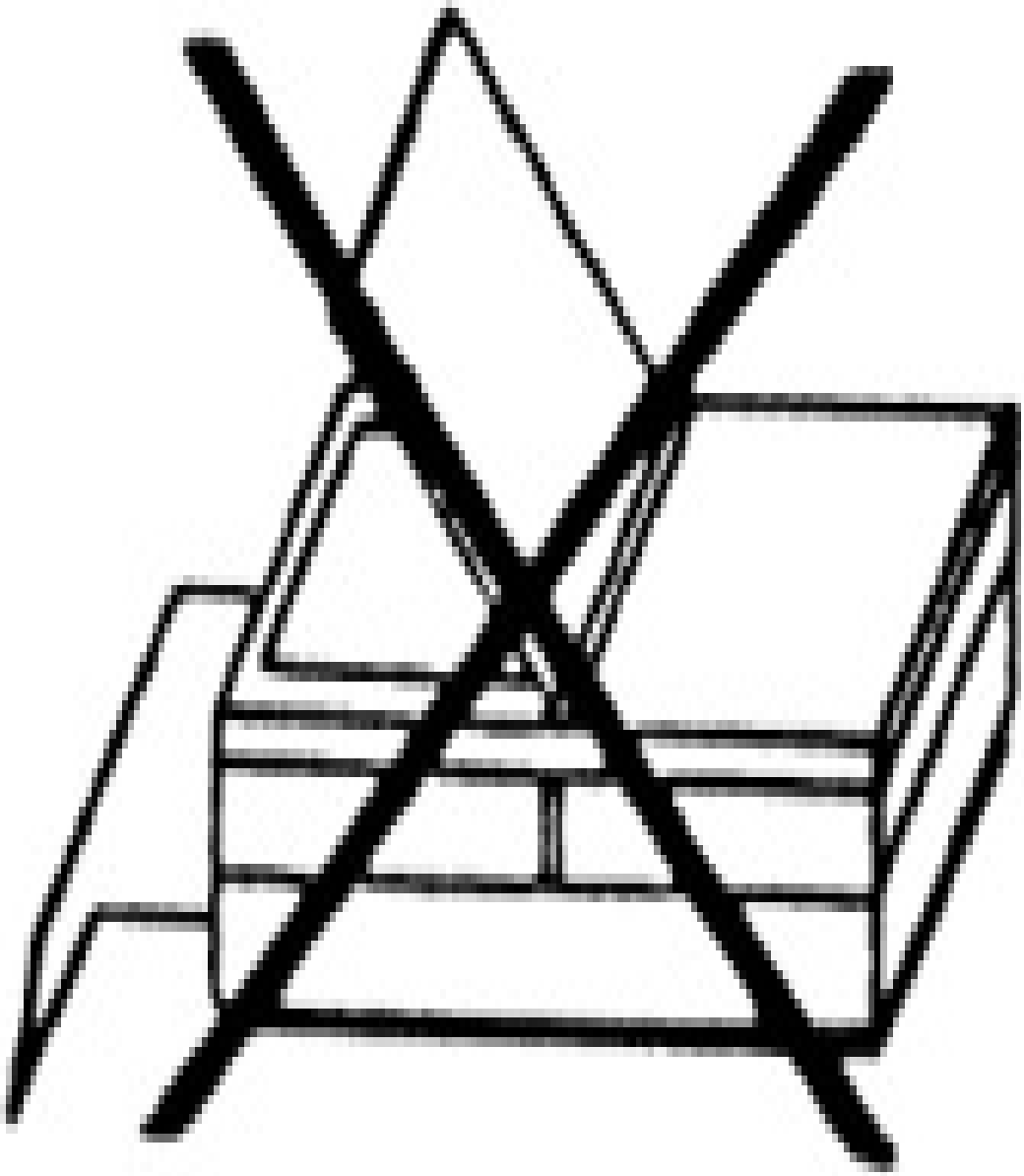
Ra-Ma®

Unity y C#

Desarrollo de videojuegos

Luis Ruelas

Ka-Ma®



La ley prohíbe

fotocopiar este libro

Unity y C#. Desarrollo de videojuegos

© Luis Ruelas

© De la edición: Ra-Ma 2018

MARCAS COMERCIALES. Las designaciones utilizadas por las empresas para distinguir sus productos (hardware, software, sistemas operativos, etc.) suelen ser marcas registradas. RA-MA ha intentado a lo largo de este libro distinguir las marcas comerciales de los términos descriptivos, siguiendo el estilo que utiliza el fabricante, sin intención de infringir la marca y solo en beneficio del propietario de la misma. Los datos de los ejemplos y pantallas son ficticios a no ser que se especifique lo contrario.

RA-MA es marca comercial registrada.

Se ha puesto el máximo empeño en ofrecer al lector una información completa y precisa. Sin embargo, RA-MA Editorial no asume ninguna responsabilidad derivada de su uso ni tampoco de cualquier violación de patentes ni otros derechos de terceras partes que pudieran ocurrir. Esta publicación tiene por objeto proporcionar unos conocimientos precisos y acreditados sobre el tema tratado. Su venta no supone para el editor ninguna forma de asistencia legal, administrativa o de ningún otro tipo. En caso de precisarse asesoría legal u otra forma de ayuda experta, deben buscarse los servicios de un profesional competente.

Reservados todos los derechos de publicación en cualquier idioma.

Según lo dispuesto en el Código Penal vigente, ninguna parte de este libro puede ser reproducida, grabada en sistema de almacenamiento o transmitida en forma alguna ni por cualquier procedimiento, ya sea electrónico, mecánico, reprográfico, magnético o cualquier otro sin autorización previa y por escrito de RA-MA; su contenido está protegido por la ley vigente, que establece penas de prisión y/o multas a quienes, intencionadamente, reprodujeren o plagiaren, en todo o en parte, una obra literaria, artística o científica.

Editado por:

RA-MA Editorial

Calle Jarama, 3A, Polígono Industrial Igarsa

28860 PARACUELLOS DE JARAMA, Madrid

Teléfono: 91 658 42 80

Fax: 91 662 81 39

Correo electrónico: editorial@ra-ma.com

Internet: www.ra-ma.es y www.ra-ma.com

ISBN: 978-84-9964-764-7

Depósito legal: M-28962-2018

Maquetación: Antonio García Tomé

Diseño de portada: Antonio García Tomé

Filmación e impresión: Safekat

Impreso en España en noviembre de 2018

Para mi hijo Axel E. Ruelas

“No sé el camino profesional que llegarás a escoger,pero en todo lo que tu desees y en lo que tu corazón te guíe,siempre estaré ahí apoyándote.Este libro es para ti hijito.”

índice

SOBRE EL AUTOR 11

TRABAJO CON UNITY 13

INTRODUCCIÓN 13

INSTALACIÓN DE UNITY 14

DESCARGAR EL MATERIAL DE APOYO 15

PREPARACION DEL PROYECTO 16

INTERFAZ DE UNITY 18

Ventana “Scene” 18

Ventana “Project” 19

Ventana “Hierarchy” 20

Ventana “Inspector” 20

EDITORES DE SCRIPT 21

Ventana “Console”21

CONSTRUCCION DE UNA PLATAFORMA BÁSICA 23

RETO AL LECTOR 32

TRABAJO CON PLAYER 37

INTRODUCCIÓN 37

PREPARACION DEL PLAYER 37

PROGRAMACION DEL MOVIMIENTO HORIZONTAL 49

PROGRAMACION DEL VOLTEO DEL PLAYER 59

ANIMACION IDLE DEL PLAYER 63

ANIMACION RUNNING DEL PLAYER CON UN BLEND TREE 69

PROGRAMACIÓN DE LA CAMARA 78

PROGRAMACIÓN DE SALTO DEL PLAYER 87

PROGRAMACIÓN DE LA ANIMACION DEL SALTO 95

PROGRAMACIÓN DEL ATAQUE DEL PLAYER 106

PROGRAMACIÓN DE LA ANIMACION DE ATAQUE DEL PLAYER 117

RETO AL LECTOR 128

TRABAJO CON ENEMIGO 01 131

INTRODUCCIÓN 131

PREPARACION DEL ENEMIGO 01 131

USO DEL COMPONENTE “NAV MESH AGENT” 137

ANIMACION DEL CAMINADO DEL ENEMIGO 01 144

CREACION DE UN GAME MANAGER 151

PROGRAMACION DEL ATAQUE DEL ENEMIGO 01 155

PROGRAMACION DEL DAÑO AL PLAYER 159

PROGRAMACION DEL DAÑO AL ENEMIGO 168

CREACIÓN DE UNA BARRA DE ENERGIA PARA EL PLAYER 176

AÑADIR CLIPS DE AUDIO 185

RETO AL LECTOR 188

CONCLUSION 188

TRABAJO CON ENEMIGO SHOOTER 189

INTRODUCCIÓN 189

PREPARACIÓN DEL ENEMIGO SHOOTER 189

PROGRAMACIÓN DE ATAQUE DEL ENEMIGO SHOOTER 197

PROGRAMACIÓN DEL PROYECTIL DEL ENEMIGO SHOOTER 200

PROGRAMACIÓN DEL DAÑOS 203

RETO AL ESTUDIANTE 209

ITEMS 211

INTRODUCCIÓN 211

CREACIÓN DE UN ITEM 2D DE SALUD 211

CREACIÓN DE UN ITEM 3D DE INVENCIBILIDAD 219

APLICACIÓN DE ANIMACIONES PARA LA TOMA DE ITEMS 232

PROGRAMACIÓN DE ENEMIGOS QUE ARROJAN ITEMS 236

RETO AL ESTUDIANTE 241

DETALLES AVANZADOS 243

INTRODUCCIÓN 243

Creación y preparación de la plataforma móvil 244

Programación de la plataforma móvil vertical 248

Creación de un “KillBox” 252

Reaparición del “Player” I 254

Reaparición del “Player” II 256

Reaparición del “Player” III 258

Programación de más Checkpoints 261

Creación de un sistema de vidas para “Player” 264

Creación de una pantalla de inicio 269

Creación de una pantalla de GameOver 281

Creación y Programación de un Item de vida 286

TRABAJO CON ENEMIGO SPAWNER 289

INTRODUCCIÓN 289

Creación de un enemigo tipo “Spawner” 289

Creación de las animaciones para el enemigo tipo “Spawner” 293

Programación de las animaciones para el enemigo tipo “Spawner” 300

Creación y programación del enemigo tipo “Spawn” 306

Generación de enemigos 314

Añadir música al juego 318

Programación de la funcionalidad de Pausa 320

BATALLA CON EL JEFE FINAL 325

INTRODUCCIÓN 325

Preparación del Jefe Final 326

Programación de un BossCheckpoint 333

Despertar al Jefe I 336

Trabajo con cámara para batalla final 340

Creación de las animaciones del Jefe Final 344

Reactivación de Player con Animation Event 347

Preparación de la condición de ataque del Jefe 348

Creación y programación de la barra de salud del jefe 354

Programación del primer ataque del jefe y reseteo de comportamientos 365

Programación del segundo y tercer ataque del jefe 372

Programación de la generación de rocas cayendo 377

Programación aleatoria de los ataques 386

Funcionalidad de daño y muerte del jefe 389

Detalles Finales 394

RETO AL ESTUDIANTE 400

IMPLEMENTACIÓN EN ANDROID 401

INTRODUCCIÓN 401

Insertar video en Unity 401

Exportación del juego para pc 404

Preparación de elementos para plataforma Android 406

Programación de controles para dispositivos móviles 418

Implementación de Adds 433

Exportación del juego para plataforma Android 438

material adicional443

SOBRE EL AUTOR

Luis Ruelas

Especialista en Animación 3D y Desarrollo de Videojuegos

Con 18 años de experiencia impartiendo clases en Universidades e Institutos del norte de México en: Desarrollo de Videojuegos con Unity, Animación y modelado 3D con Autodesk Maya y ZBrush, Metodología de la programación, Creación de multimedia y efectos visuales con Adobe Suite.

Proveedor de servicios de Animación 3D, capacitación, desarrollo de videojuegos y multimedia.

Instructor en Udemy con alumnos a nivel nacional de más de 38 países e impartiendo “Master Class” en Desarrollo de Videojuegos con Unity en diversos estados de México.

Creador de “Cineanimación”, empresa dedicada a impartir cursos especializados en el ámbito del desarrollo de videojuegos y animación 3D.

www.cineanimacion.com

creacionvideojuegos@gmail.com

1

TRABAJO CON UNITY

INTRODUCCIÓN

Unity es un motor de videojuegos creado por Unity Technologies disponible para Windows y OS X.

Con Unity podrás crear juegos en 2D y 3D para diversas plataformas como Windows, OS X, XBOX ONE, PS4, Nintendo Switch, Android y IOS como principales plataformas.

A su vez, también permite crear videojuegos y recorridos con realidad virtual y crear proyectos profesionales con realidad aumentada.

Los lenguajes de programación que acepta Unity son C#, JavaScript y Boo.

En este libro desarrollaremos scripts en el lenguaje C# y aprenderemos la metodología de la programación desde el inicio. No necesitas tener ninguna base para tomar este curso.

En este libro tutorial podrás aprender Unity desde cero creando un videojuego tipo plataforma 3D con todos los elementos necesarios de un videojuego profesional.

Creación de entornos 3D

Programación y animación de movimientos de personajes

Programación de funcionalidades de los personajes

Lógicas de diversos ataques de enemigos

Generación y programación de ítems

Funcionalidades profesionales de videojuegos como pantalla de inicio, pantalla de “Game Over”, música, efectos sonoros, explosiones, pausa, barras de vida, cinemáticos y mas

Exportación de tu videojuego a plataforma Android, publicación y comercialización de tus videojuegos

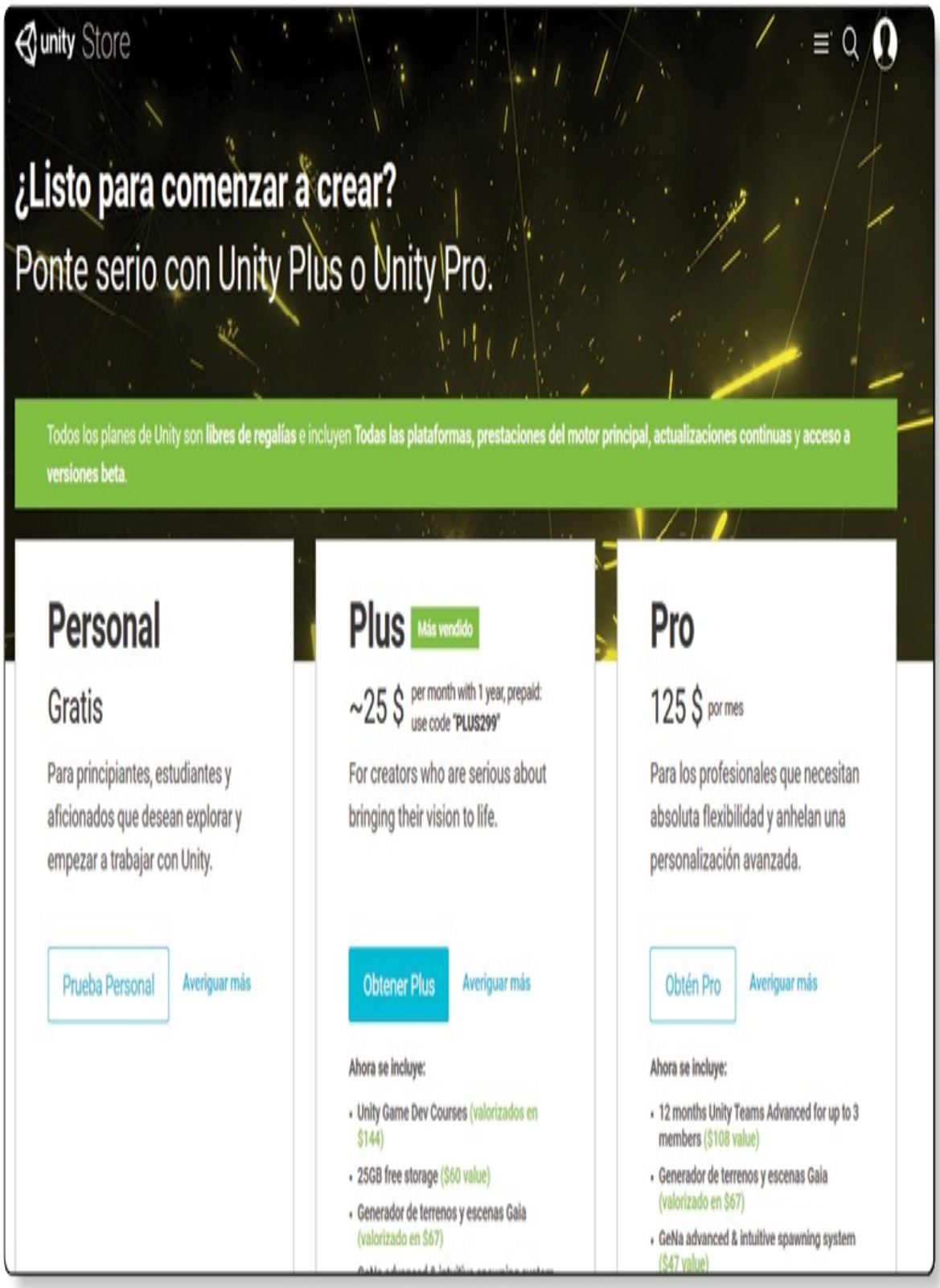
INSTALACIÓN DE UNITY

Como primer paso para comenzar a desarrollar los ejercicios de este libro deberá de descargar el software UNITY.

Vaya a https://unity3d.com/es

Clic en “Consigue Unity”.

Encontrarás 3 versiones: Personal, Plus y Pro. Para los ejercicios de este libro utilizaremos la versión gratuita Personal.


unity Store
¿Listo para comenzar a crear?
Ponte serio con Unity Plus o Unity Pro.
Todos los planes de Unity son libres de regalías e incluyen Todas las plataformas, prestaciones del motor principal, actualizaciones continuas y acceso a versiones beta.
Personal
Gratis
Para principiantes, estudiantes y aficionados que desean explorar y empezar a trabajar con Unity.
Prueba Personal
Averiguar más
Plus
Más vendido
~25 $ per month with 1 year, prepaid: use code "PLUS299"
For creators who are serious about bringing their vision to life.
Obtener Plus
Averiguar más
Ahora se incluye:
Unity Game Dev Courses (valorizados en $144)
25GB free storage ($60 value)
Generador de terrenos y escenas Gaia (valorizado en $67)
Pro
125 $ por mes
Para los profesionales que necesitan absoluta flexibilidad y anhelan una personalización avanzada.
Obtén Pro
Averiguar más
Ahora se incluye:
12 months Unity Teams Advanced for up to 3 members ($108 value)
Generador de terrenos y escenas Gaia (valorizado en $67)
GeNa advanced & intuitive spawning system ($47 value)

Tipos de cuentas de Unity

Descarga la versión de tu ordenador

Instala el software

Se te pedirá que crees una cuenta de Unity, misma que estará ligada a tus proyectos.

Unity 2018.1.0f2
unity
Create a Unity ID
If you already have a Unity ID, please sign in here.
Email
Password
Username
Full Name
I agree to the Unity Terms of Use and Privacy Policy
I understand that checking this box that I am agreeing to receive promotional materials from Unity
Already have a Unity ID?
Create a Unity ID
Or
Sign in with google
Sign in with facebook

Pantalla para crear una identidad de Unity.

Una vez creada se te pedirá que confirmes tu email con un enlace en tu cuenta de correo

Inicia Unity con un Unity ID

DESCARGAR EL MATERIAL DE APOYO

Este ejercicio contiene material de apoyo mismo que se encuentra dividido por capítulos.

Los “Assets” que están incluidos en este proyecto son objetos 3D y personajes con sus respectivas animaciones de la página https://shop.bitgem3d.com/

https://shop.bitgem3d.com

HAND PAINTED 3D | FACETED SERIES | SMASHY CRAFT SERIES | GAME DEV FORUM | 2D PIXEL ART | LOG IN / SIGN UP | 0

BITGEM | 3D MODELS | GAME TEXTURES | FREE MODELS & TEXTURES | ON SALE | Search

LOW POLY 3D MODELS AND TEXTURES FOR GAME DEVELOPERS

We are a small team of industry veterans, working first and foremost on building a high quality, style consistent low poly 3d model and texture inventory for game developers looking to kick start a new project!

WISHLIST

5 GHOUL MONSTERS
$49.00

DEMON CREW BUNDLE
$59.00

SKELETON CREW BUNDLE
$59.00

MINI SKELETON SWARM PACK
$39.00

MAYAN TEMPLE SET
$39.00

PIXEL DUNGEON SET
$19.00 ~~$49.00~~

HAUNTED MANSION SET
$59.00

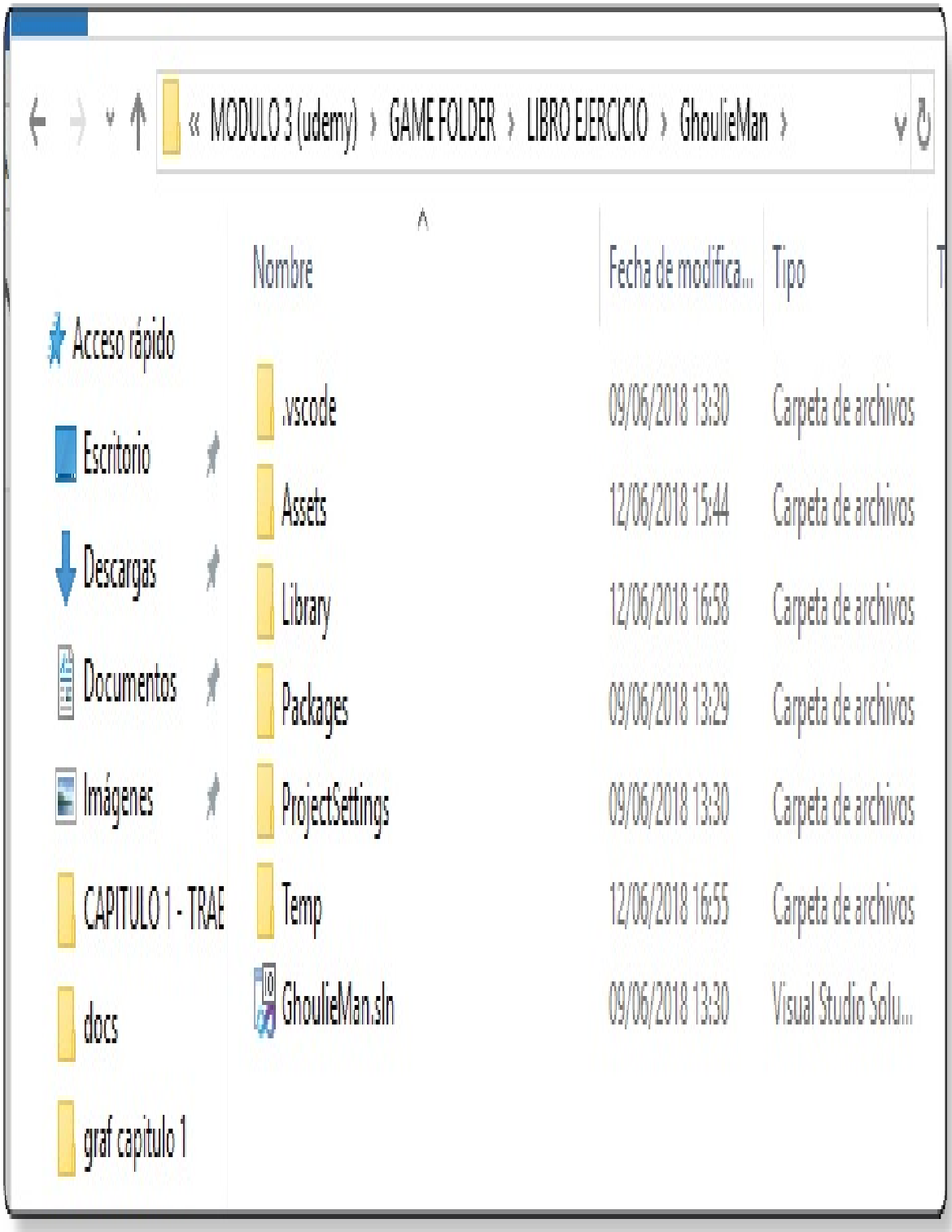
DUNGEON STARTER SET
$59.00

Dentro de la carpeta “resources” ubicada en el sitio “www.ra-ma.es” se encuentran todos los elementos en 3D, texturas, y audios que utilizarás en cada capítulo.

PREPARACIÓN DEL PROYECTO

Lo primero que tenemos que hacer es hacer un “Set up” del proyecto.

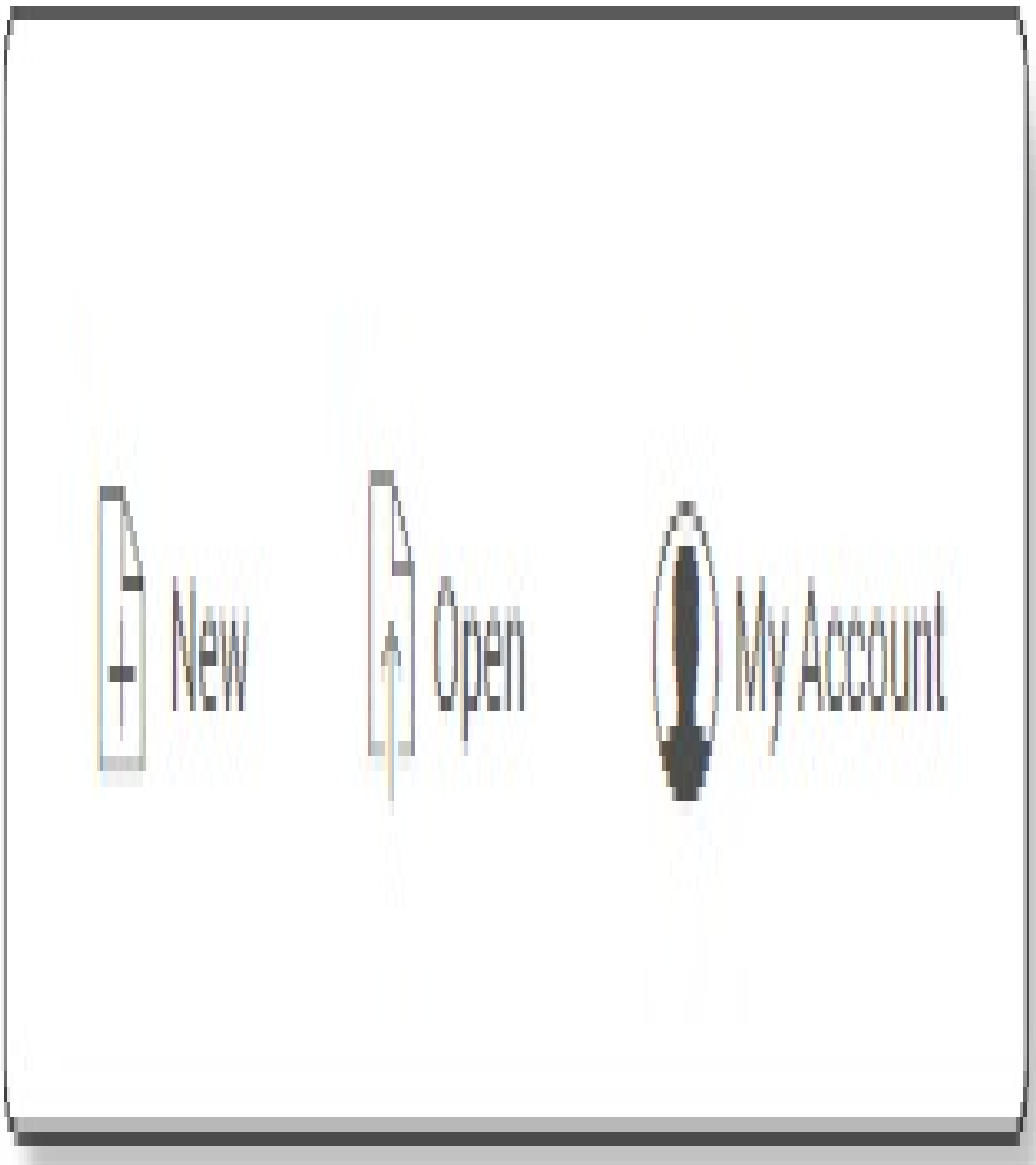
Al momento de crear un nuevo proyecto se creará una carpeta maestra para nuestro proyecto, misma que contendrá las carpetas necesarias para manejar nuestro proyecto.


« MODULO 3 (udemy) › GAME FOLDER › LIBRO EJERCICIO › GhoulieMan ›
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
CAPITULO 1 - TRAE
docs
graf capitulo 1
Nombre
Fecha de modifica...
Tipo
.vscode
09/06/2018 13:30
Carpeta de archivos
Assets
12/06/2018 15:44
Carpeta de archivos
Library
12/06/2018 16:58
Carpeta de archivos
Packages
09/06/2018 13:29
Carpeta de archivos
ProjectSettings
09/06/2018 13:30
Carpeta de archivos
Temp
12/06/2018 16:55
Carpeta de archivos
GhoulieMan.sln
09/06/2018 13:30
Visual Studio Solu...

NOTA:

Cuando quiera exportar su proyecto completo; deberá de mover toda la carpeta maestra para que no se pierda ningún elemento.

Clic en el ícono “New”


New
Open
My Account

Nuevo proyecto

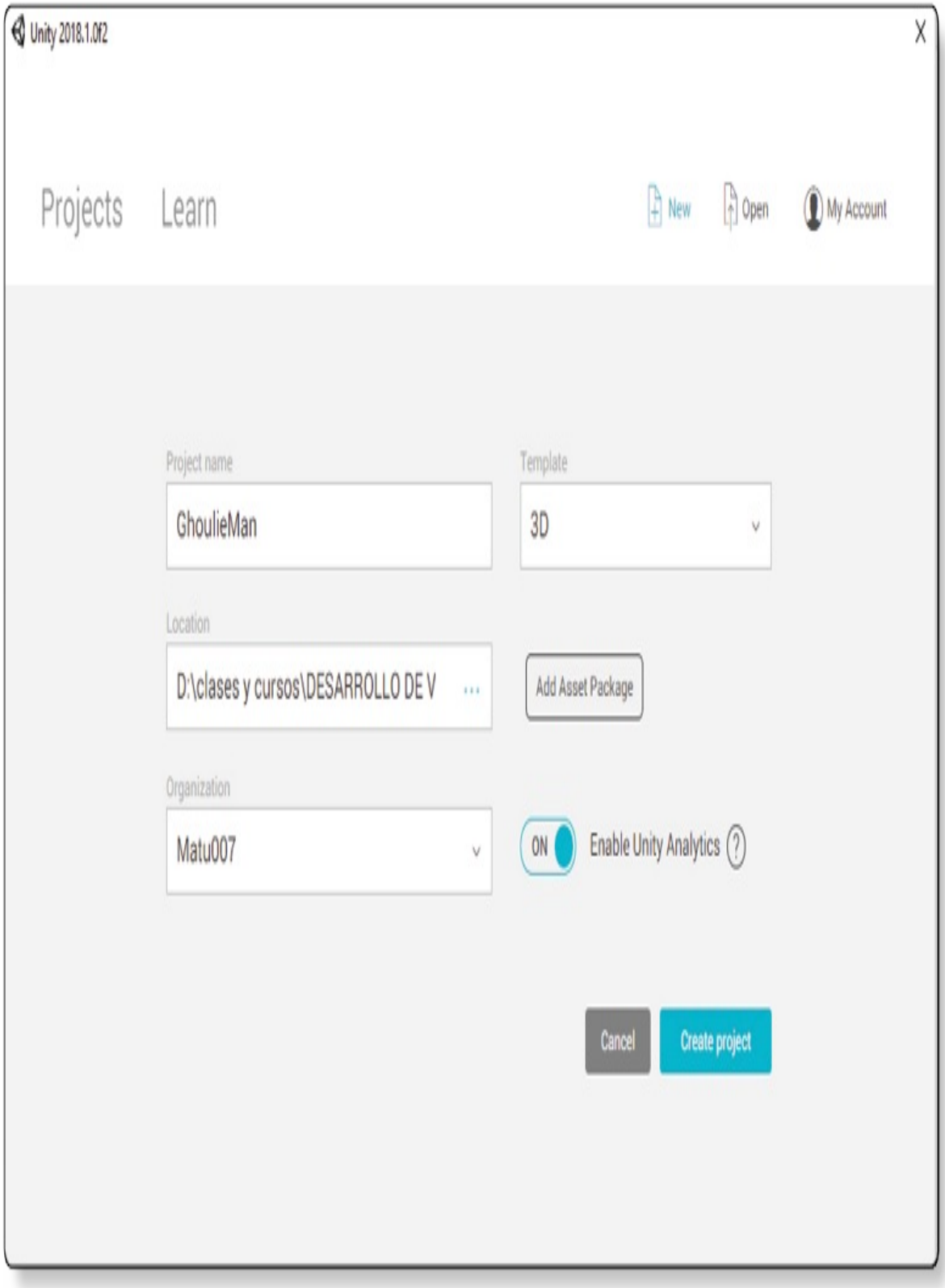
En la siguiente pantalla se te pedirá lo siguiente:

Project name: Deberás de asignar un nombre para tu proyecto, en el caso de este ejercicio lo llamaré “GhoulieMan”

Template: Podrás elegir un proyecto 3D o 2D, para este proyecto elegiremos 3D

Location: La localización dentro de tu unidad donde estará alojado tu carpeta de proyecto

Organization: El nombre de tu organización personal


Unity 2018.1.0f2
Projects
Learn
New
Open
My Account
Project name
GhoulieMan
Template
3D
Location
D:\clases y cursos\DESARROLLO DE V
Add Asset Package
Organization
Matu007
ON
Enable Unity Analytics
Cancel
Create project

Parámetros para el nuevo proyecto.

Clic en “Create Proyect”

INTERFAZ DE UNITY

En este apartado revisaremos las ventanas principales para el inicio de nuestro trabajo dentro de Unity.

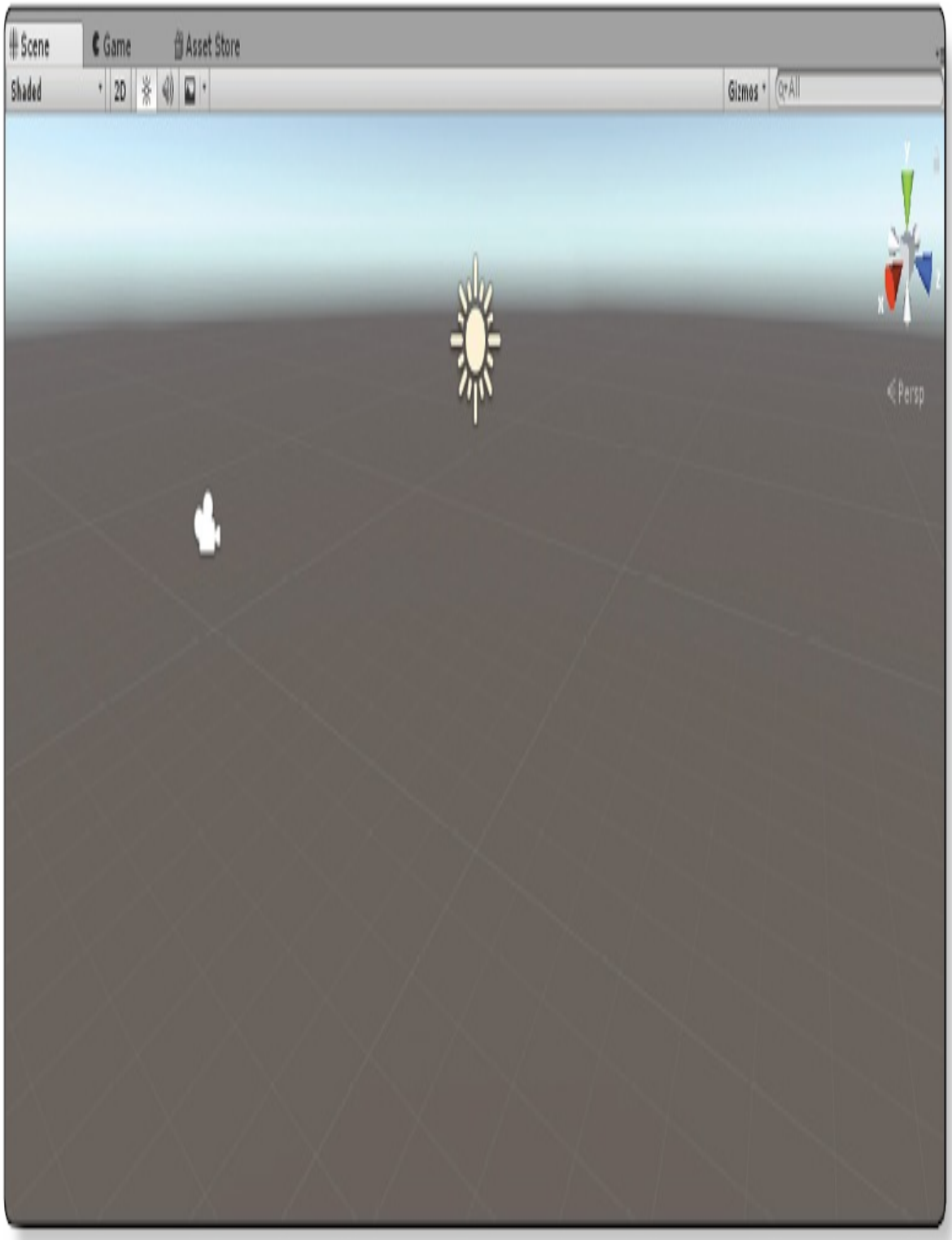
Ventana “Scene”

La ventana “Scene” es nuestro monitor visual de todos los elementos y entorno de nuestro videojuego.

Para poderte mover a través de ella en el modo 3D:

Con Alt + botón izquierdo del ratón podrás rotar la vista

Con Alt + botón derecho del ratón podrás hacer acercamiento o alejamiento de la vista


Scene
Game
Asset Store
Shaded
2D
Gizmos
Persp

Ventana Scene

La barra de herramientas nos ofrece herramientas para movernos y modificar elementos dentro de la ventana “Scene” mismos que iremos viendo conforme a su utilidad.

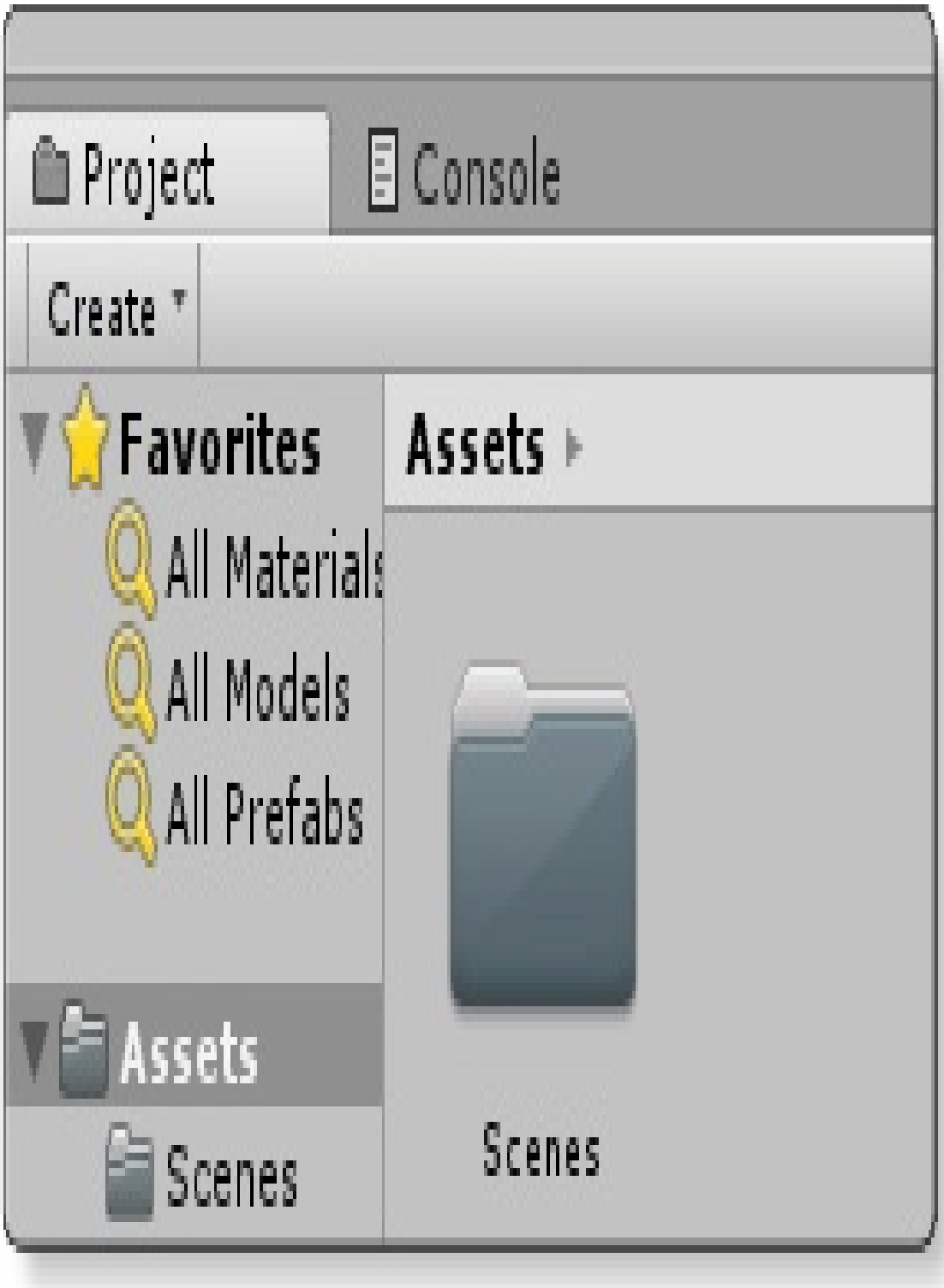
La primera herramienta es “Hand Tool” (acceso directo tecla “Q” en el teclado) nos sirve para plantear y movilizarnos en la ventana “Scene”.

File Edit Assets GameObject C
Hand Tool
Create
Q▾All

Barra de Herramientas

Ventana “Project”

En la ventana “Project” será donde se importen o se creen los elementos (“Assets”) que se añadirán en el videojuego.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Scenes
Assets
Scenes

Ventana Project

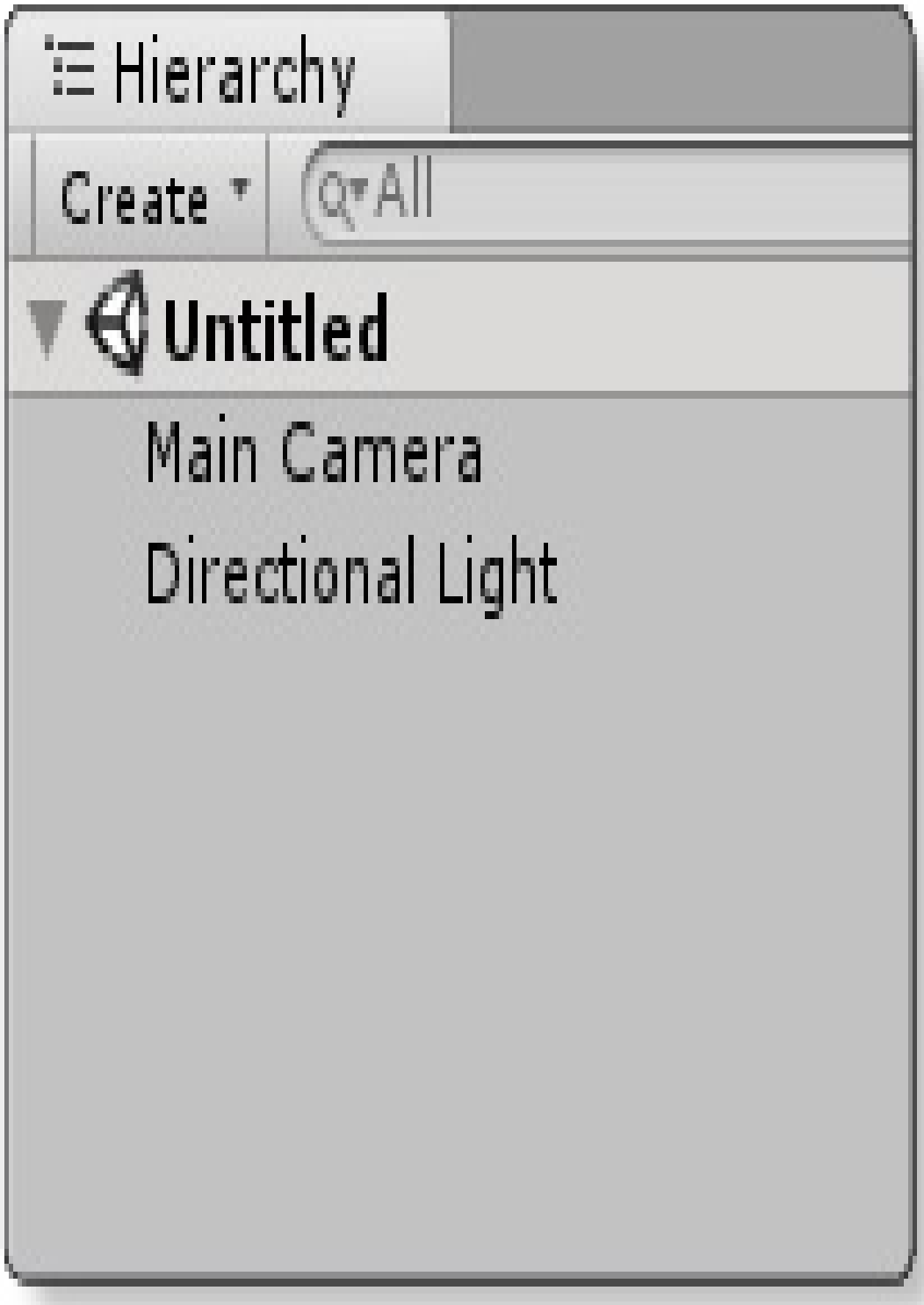
Ventana “Hierarchy”

La ventana “Hierarchy” es donde se encontrarán los elementos existentes dentro de nuestro videojuego.

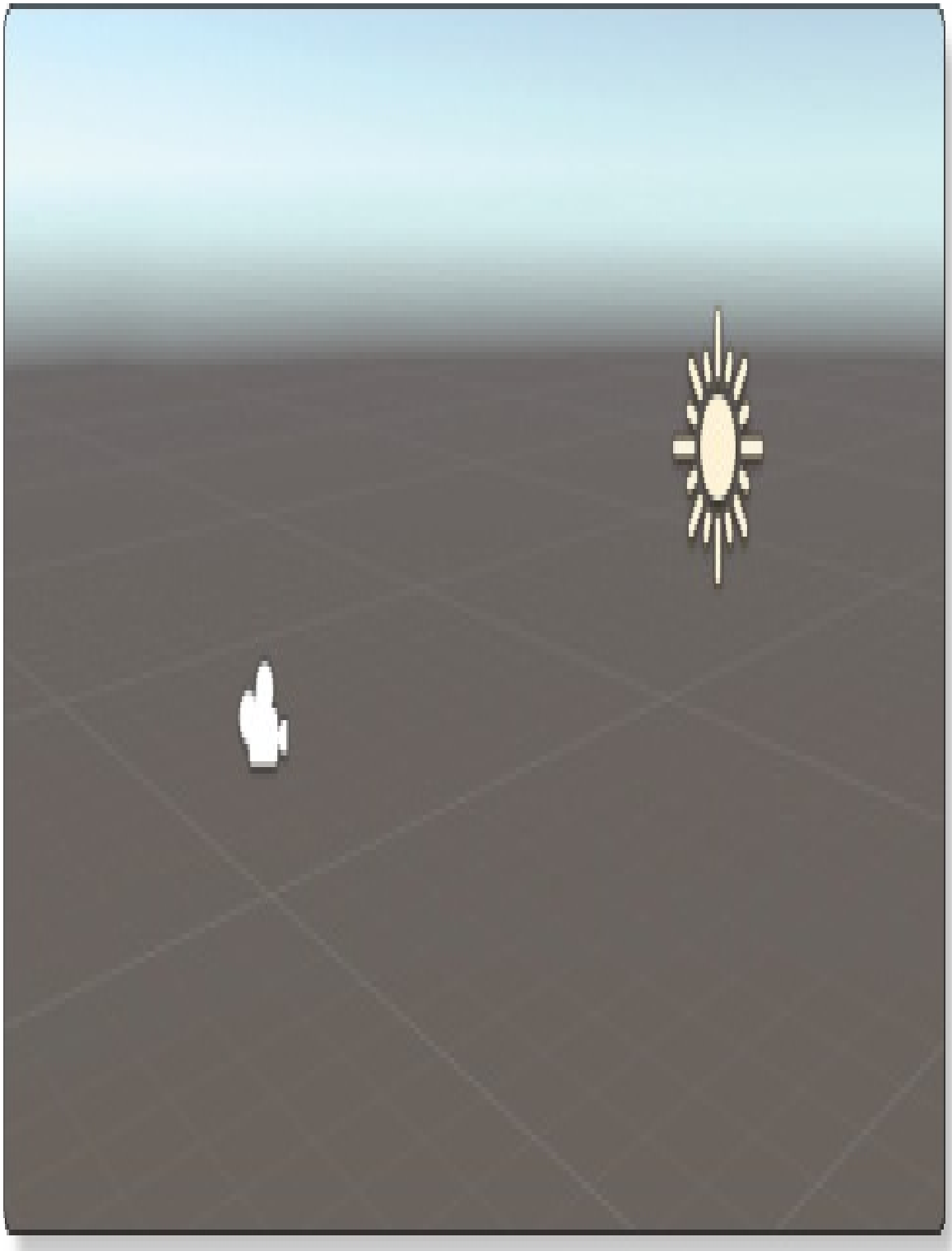
Por defecto, al iniciar una nueva escena aparecerán dos elementos:

Main Camera: Es la cámara que nos permite visualizar el juego en la pestaña “Game”, sin ella no tenemos visualización alguna.

Directional Light: Es la luz que nos permite iluminar los elementos dentro de nuestro juego, sin ella todo estaría en oscuridad.


Hierarchy
Create
All
Untitled
Main Camera
Directional Light

Ventana Hierarchy



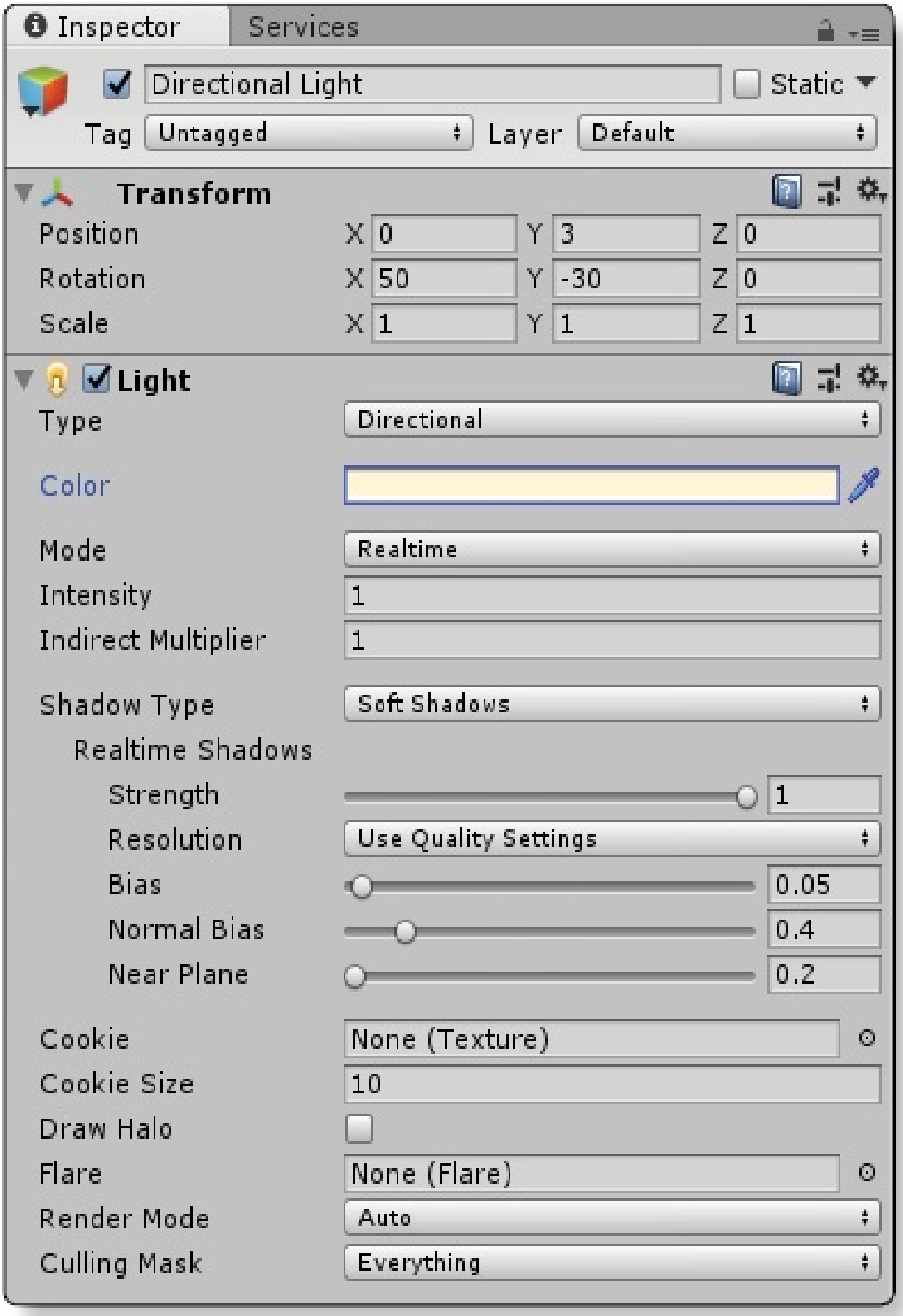
Elementos por defecto

Ventana “Inspector”

En la ventana “Inspector” podremos visualizar y editar todos los “settings” de los elementos que tengamos seleccionados.

En la ventana de “Hierarchy” seleccione el objeto “Main Camera”

Vaya a la ventana “Inspector” para visualizar los “settings” de la cámara


Inspector
Services
Directional Light
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 3
Z 0
Rotation
X 50
Y -30
Z 0
Scale
X 1
Y 1
Z 1
Light
Type
Directional
Color
Mode
Realtime
Intensity
1
Indirect Multiplier
1
Shadow Type
Soft Shadows
Realtime Shadows
Strength
1
Resolution
Use Quality Settings
Bias
0.05
Normal Bias
0.4
Near Plane
0.2
Cookie
None (Texture)
Cookie Size
10
Draw Halo
Flare
None (Flare)
Render Mode
Auto
Culling Mask
Everything

Ventana Inspector

EDITORES DE SCRIPT

Ventana “Console”

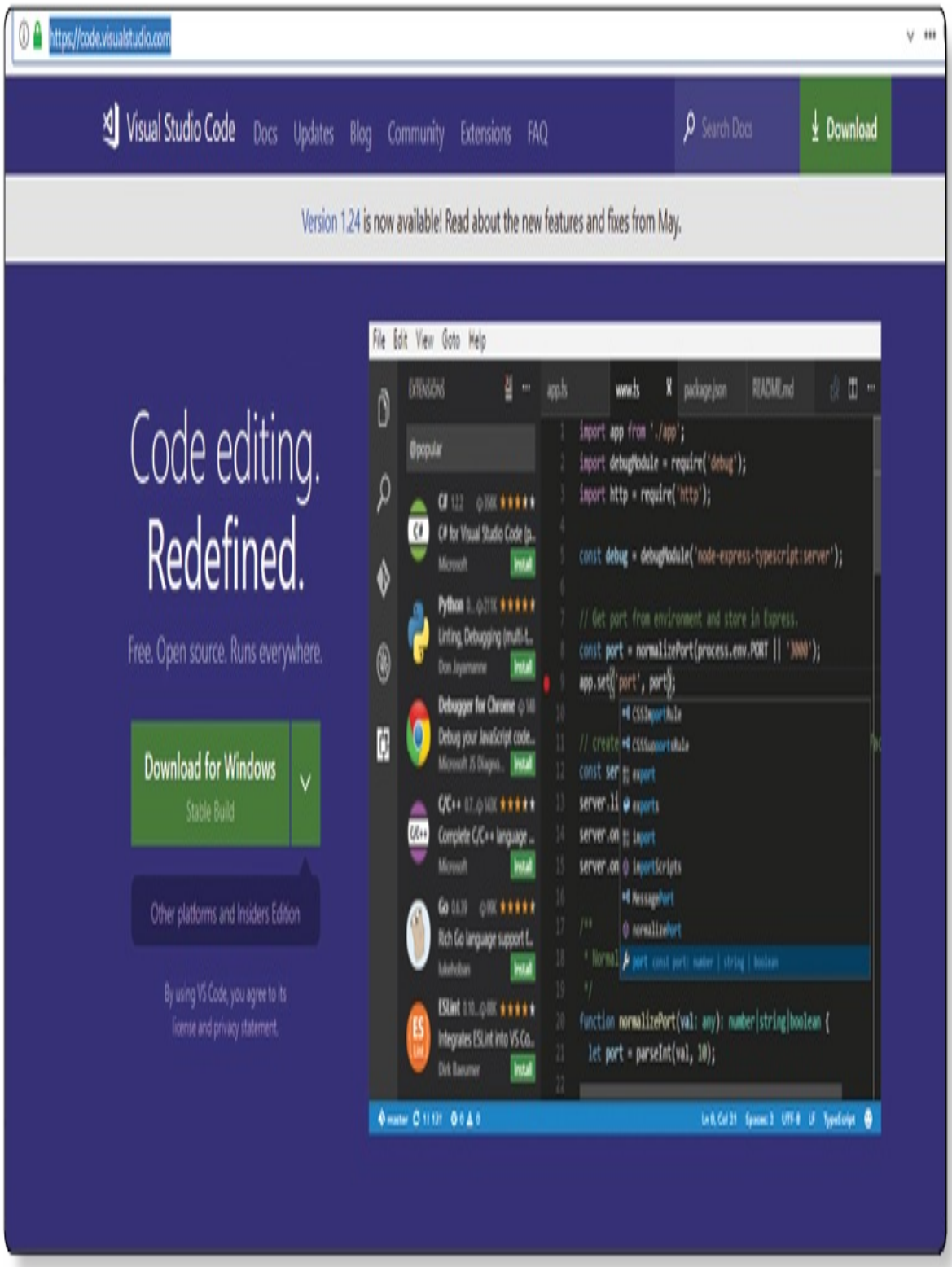
En la ventana “Console” podremos visualizar las alertas de errores en el código de programación, así como las advertencias sobre nomenclaturas obsoletas e impresiones con el commando “print” dentro del Código.

Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor

Para editar el código de nuestros scripts, Unity cuenta con el editor Visual Studio.

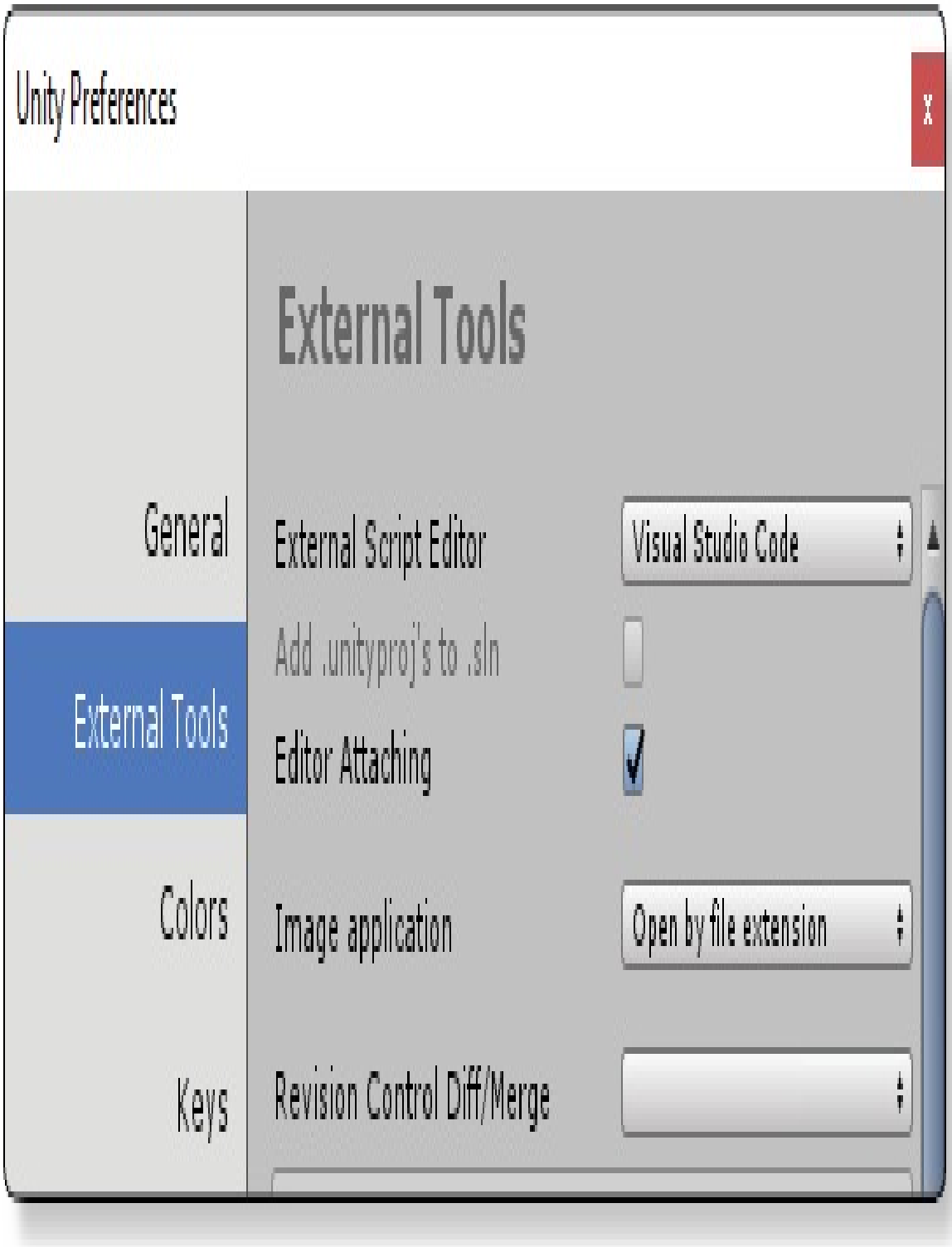
Los ejercicios de este libro están editados con Visual Studio Code que podrás obtener de manera gratuita en el siguiente enlace.

https://code.visualstudio.com/


https://code.visualstudio.com
Visual Studio Code
Docs
Updates
Blog
Community
Extensions
FAQ
Search Docs
Download
Version 1.24 is now available! Read about the new features and fixes from May.
Code editing.
Redefined.
Free. Open source. Runs everywhere.
Download for Windows
Stable Build
Other platforms and Insiders Edition
By using VS Code, you agree to its
license and privacy statement.
File Edit View Goto Help
EXTENSIONS
app.ts
www.ts
package.json
README.md
@popular
C# for Visual Studio Code (p...
Microsoft
Python
Linting, Debugging (multi-t...
Don Jayamanne
Debugger for Chrome
Debug your JavaScript code...
Microsoft JS Diagno...
C/C++
Complete C/C++ language ...
Microsoft
Go
Rich Go language support f...
lukehoban
ESLint
Integrates ESLint into VS Co...
Dirk Baeumer
import app from './app';
import debugModule = require('debug');
import http = require('http');
const debug = debugModule('node-express-typescript:server');
// Get port from environment and store in Express.
const port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
function normalizePort(val: any): number|string|boolean {
let port = parseInt(val, 10);

Página Web de Visual Studio Code

Para asignar un editor de código, en el menú general de Unity asigne “Edit / Preferences” y en la ventana de preferencias seleccione la pestaña “External Tools” y ahí podrá asignar el editor de su preferencia.


Unity Preferences
External Tools
General
External Tools
Colors
Keys
External Script Editor
Visual Studio Code
Add .unityproj's to .sln
Editor Attaching
Image application
Open by file extension
Revision Control Diff/Merge

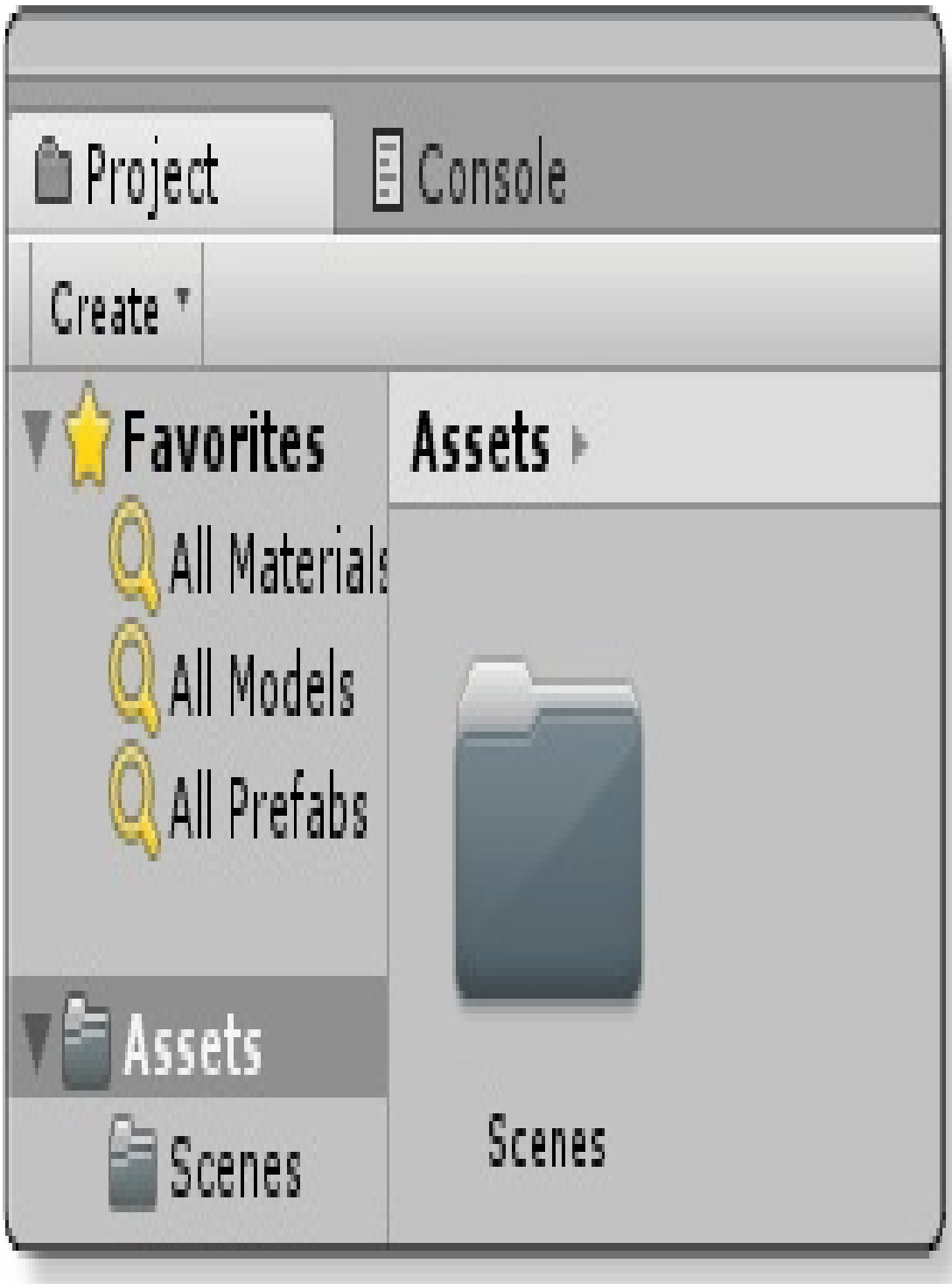
Preferencias de Unity

CONSTRUCCIÓN DE UNA PLATAFORMA BÁSICA

En este apartado crearemos una plataforma básica con todos los elementos físicos y visuales necesarios para que el personaje se pueda desplazar en ella, importando los elementos básicos desde nuestra carpeta de “resources”.

Para iniciar con nuestro proyecto debemos de crear un nuevo folder (clic derecho) de escenas en la ventana de “Proyect”, dentro del folder “Assets”.

Haga un folder de “Scenes”.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Scenes
Assets
Scenes

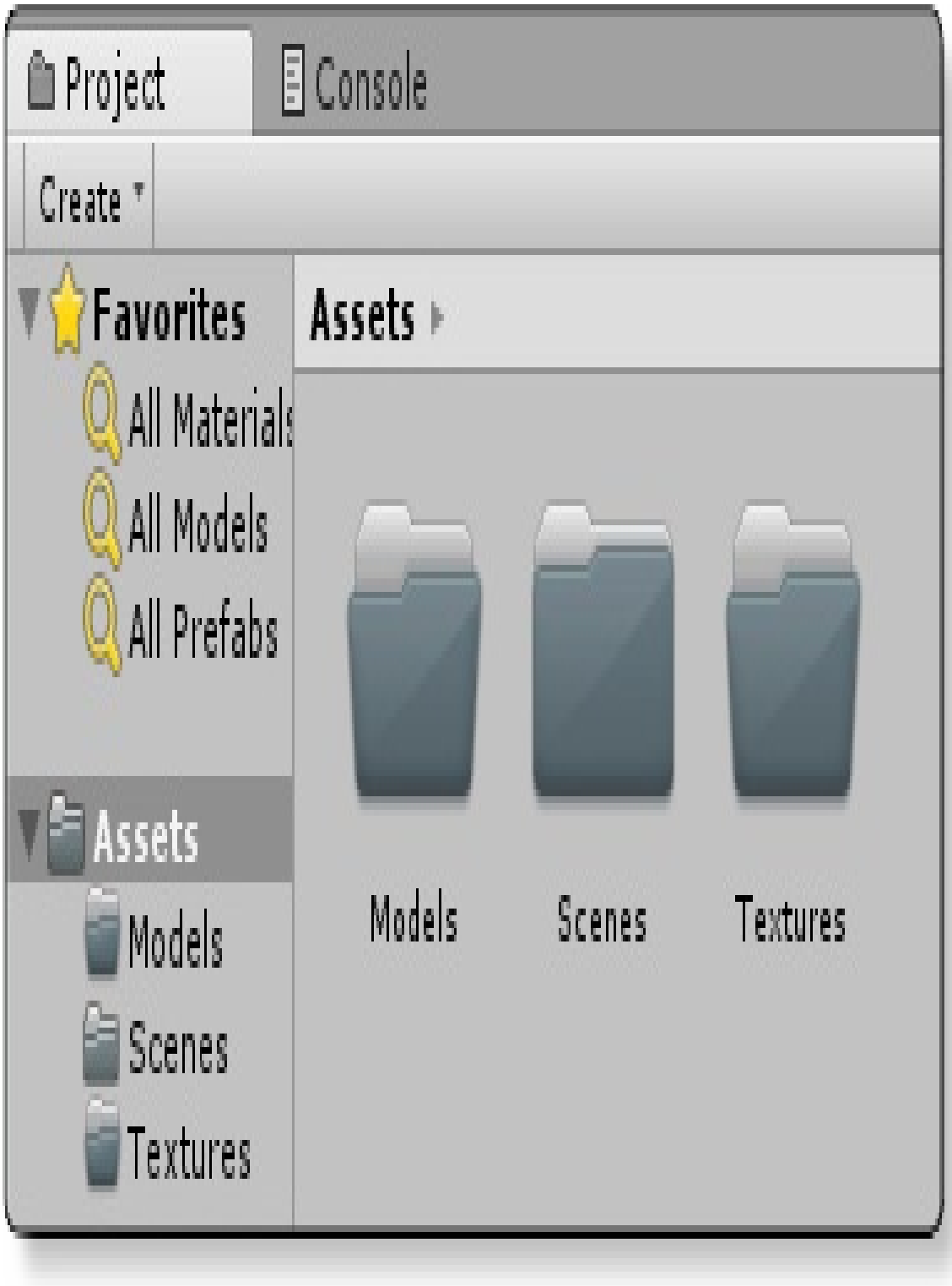
Carpeta de escenas

Salve la escena como “GhoulieMan01 (File/ Save Scene as…).

NOTA:

Es recomendable que salve el proyecto con diferentes nombres a medida que se va avanzando en el proyecto por cuestiones de seguridad. En dado caso que se llegará a corromper un archivo, se tiene un respaldo de pasos anteriores.

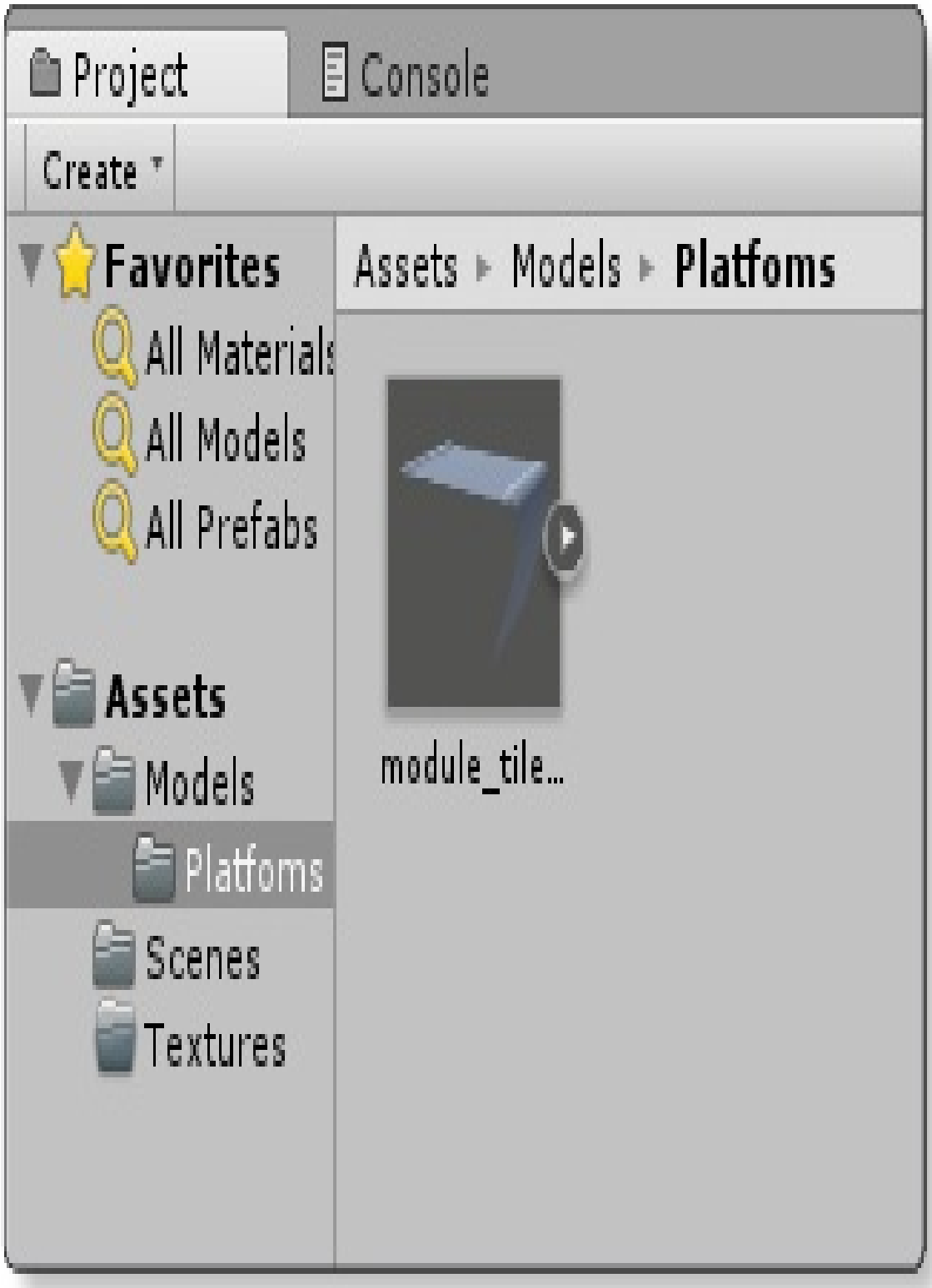
Haga dos folders y nómbrelo “Models” y “Textures”.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Models
Scenes
Textures
Assets
Models
Scenes
Textures

Creación de carpetas

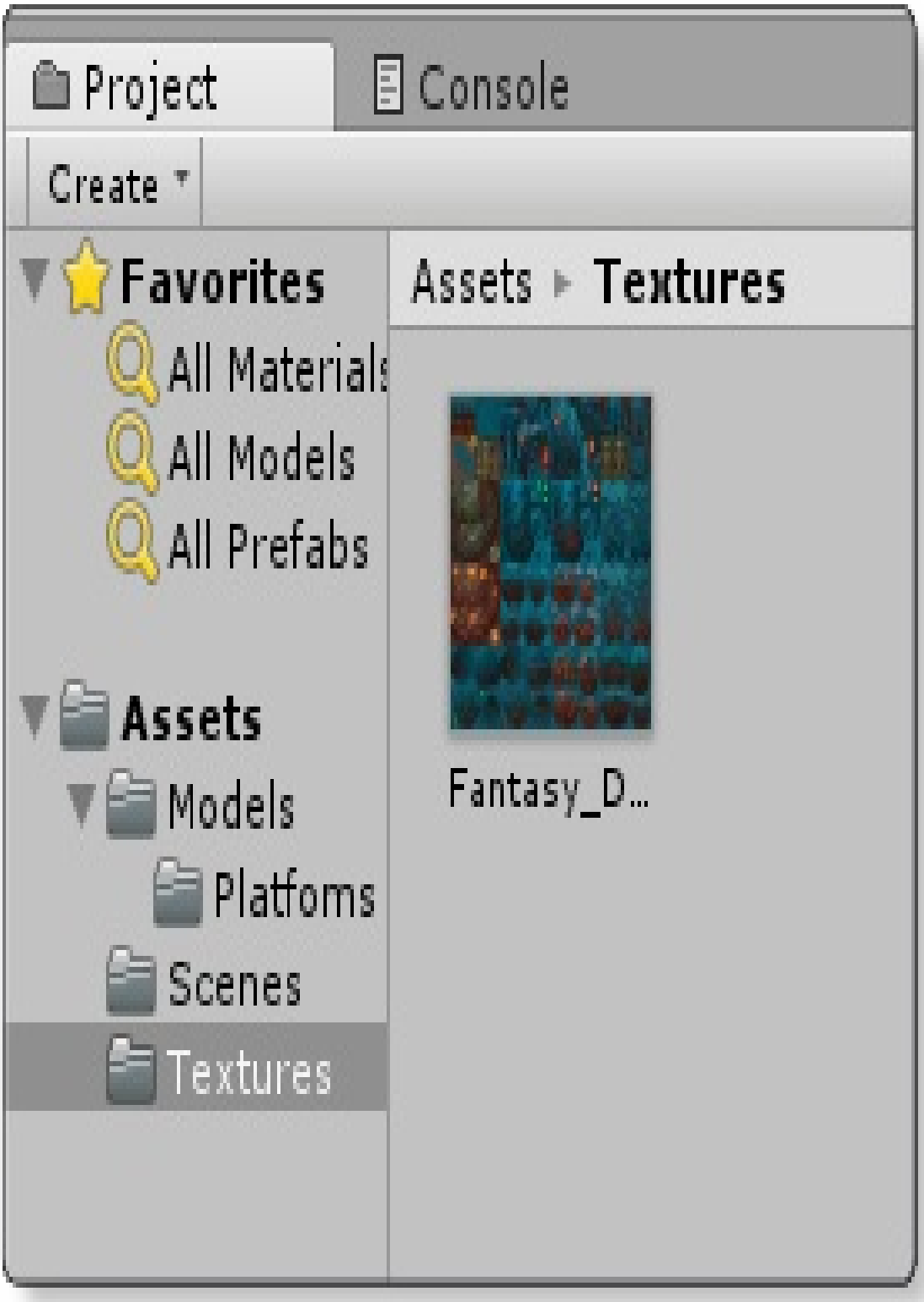
Dentro del folder “Models” haga otra carpeta y nómbrela “Platforms”

Importe del folder de resources el objeto “module_tile_02_bd” al folder de “Platforms”.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Models
Platfoms
Scenes
Textures
Assets ▸ Models ▸ Platfoms
module_tile...

Importar 3D Assets

Importe al folder de “Textures” la textura “Fantasy_Demon_Cave_Atlas.tga”.

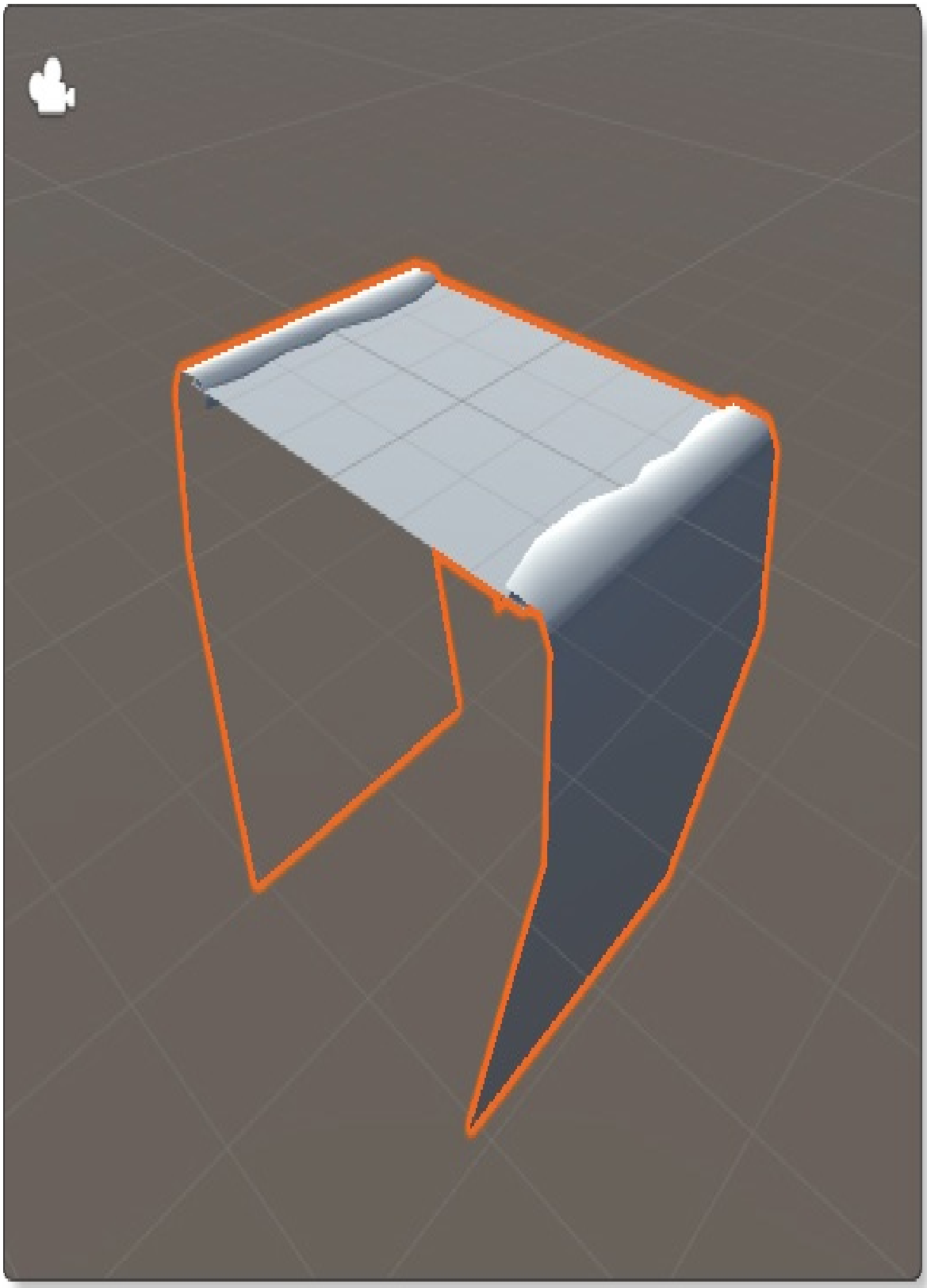

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Models
Platforms
Scenes
Textures
Assets ▸ Textures
Fantasy_D...

Importar texturas

Arrastre el objeto “module_tile” a la ventana “Hierarchy”.

Hierarchy
Create
Q▾All
Untitled*
Main Camera
Directional Light
module_tile_02_bd

Ventana Hierarchy

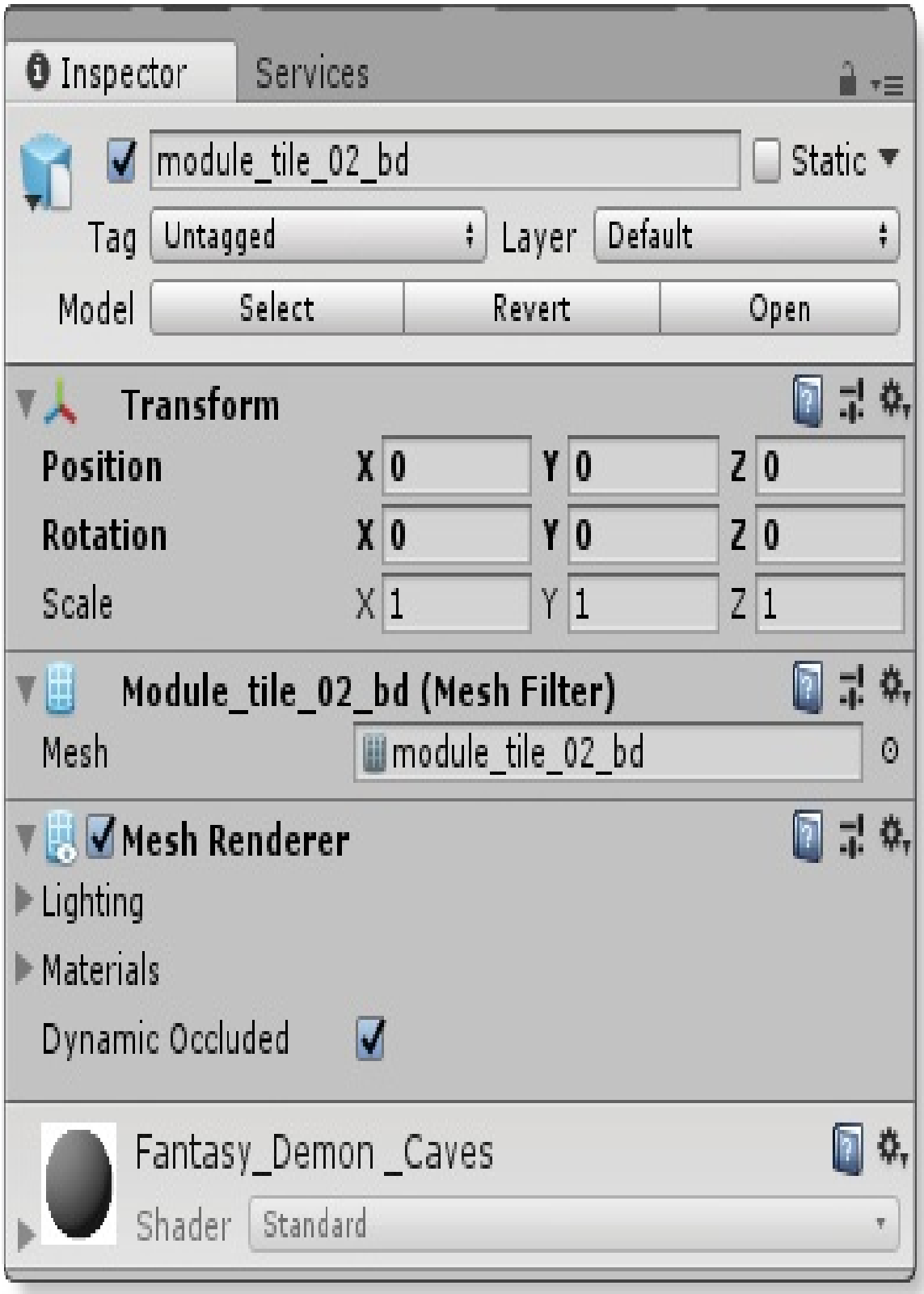


Vista “Scene”

Teniendo seleccionado el objeto observe sus atributos en la ventana “Inspector”.

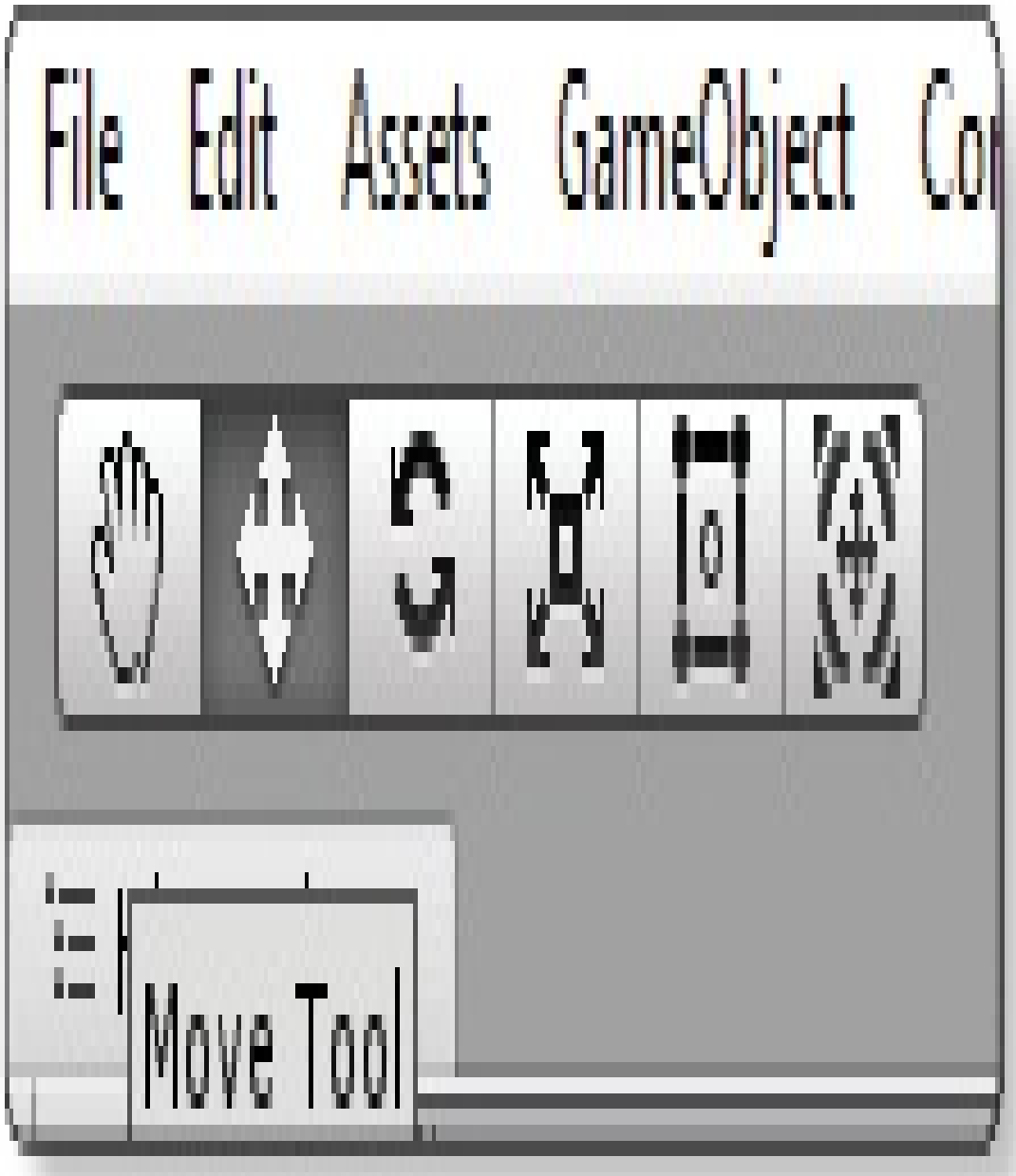
NOTA:

Cuando añadimos un objeto colocándolo directamente en la ventana Hierachy sin tener nada seleccionado, este objeto se coloca en su posición 0,0,0.


Inspector
Services
module_tile_02_bd
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Module_tile_02_bd (Mesh Filter)
Mesh
module_tile_02_bd
Mesh Renderer
Lighting
Materials
Dynamic Occluded
Fantasy_Demon _Caves
Shader
Standard

Propiedades de la plataforma 3D

Cuando colocamos estos elementos podemos moverlos de lugar con la herramienta “Move Tool” (tecla “W” en el teclado).


File Edit Assets GameObject Co
Move Tool

MoveTool

NOTA:

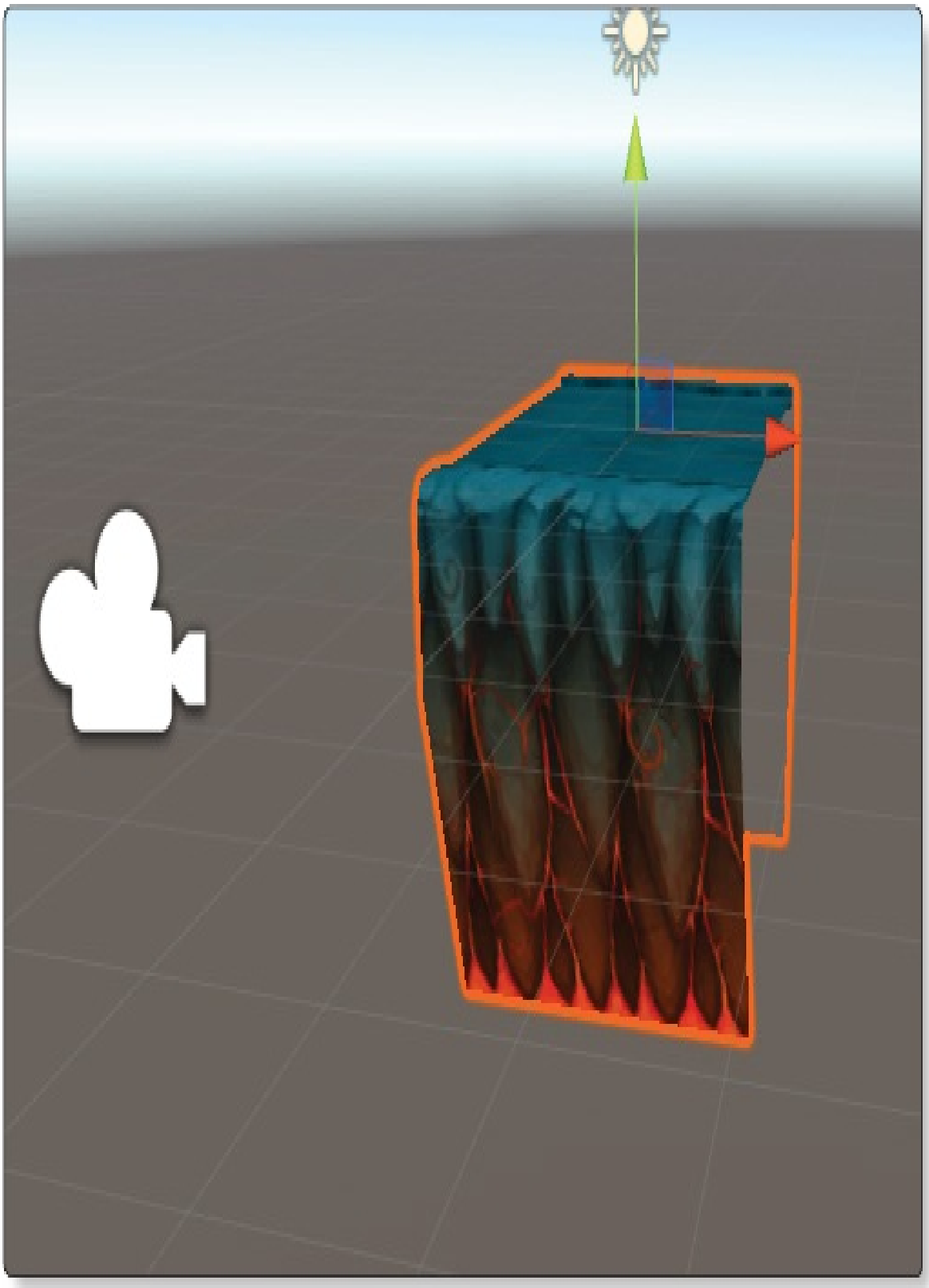
No es recomendable rotarla ya que nos puede generar problemas cuando asignemos algunos componentes.

Con la tecla “F” del teclado se realiza un “Frame” que significa que centra en la pantalla el objeto que tenemos seleccionado.

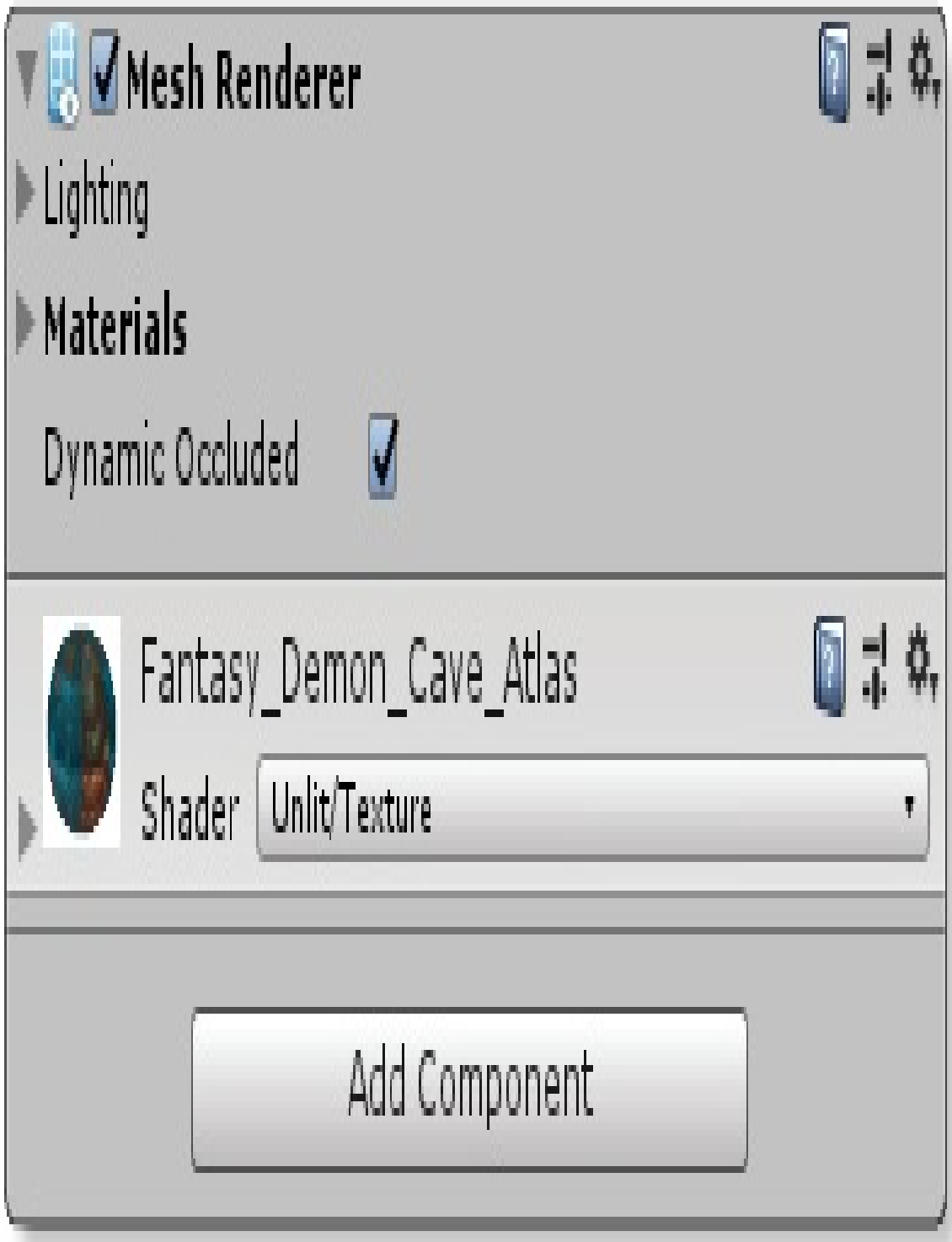
Rote la vista de la cámara de tal manera que la cámara quede frente al objeto

De la ventana Proyect, arrastre la textura “Fantasy_Demon_Cave_Atlas.tga” al objeto

En inspector cambie el tipo de material a Unlit/Texture.



Texturización de objeto

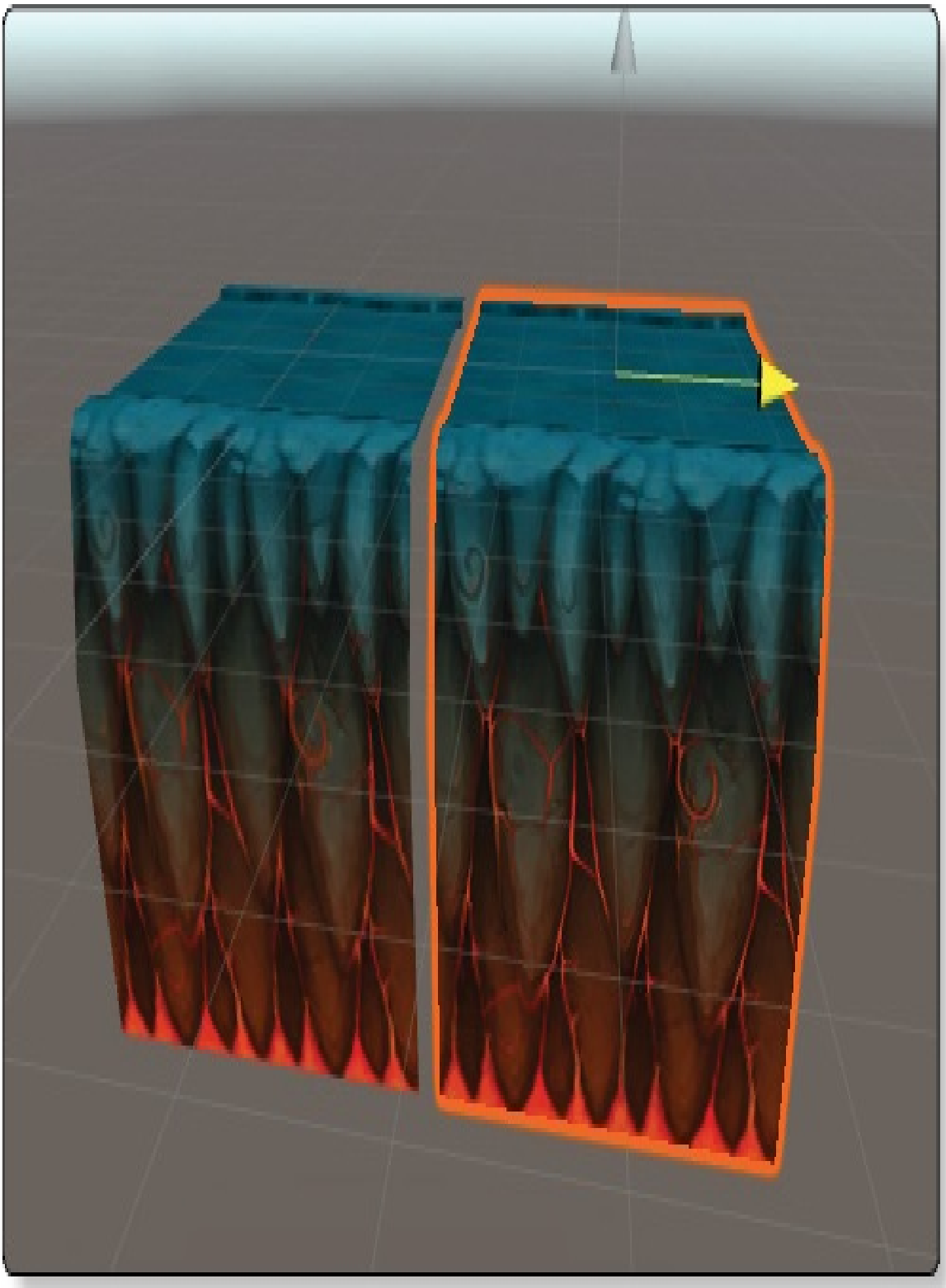

Mesh Renderer
Lighting
Materials
Dynamic Occluded
Fantasy_Demon_Cave_Atlas
Shader
Unlit/Texture
Add Component

Material Unlit/Texture

El tipo de material o “Shader” tiene diversas propiedades en cuanto a la reflexión de luz y a los tipos de textuarizaciones se refiere. En cuanto al tipo de material que estamos utilizando “Unlit/ Texture” es un material con iluminación propia y que no depende de la iluminación de la escena.

Para duplicar un elemento basta con seleccionarlo y asignar “Control + D”.

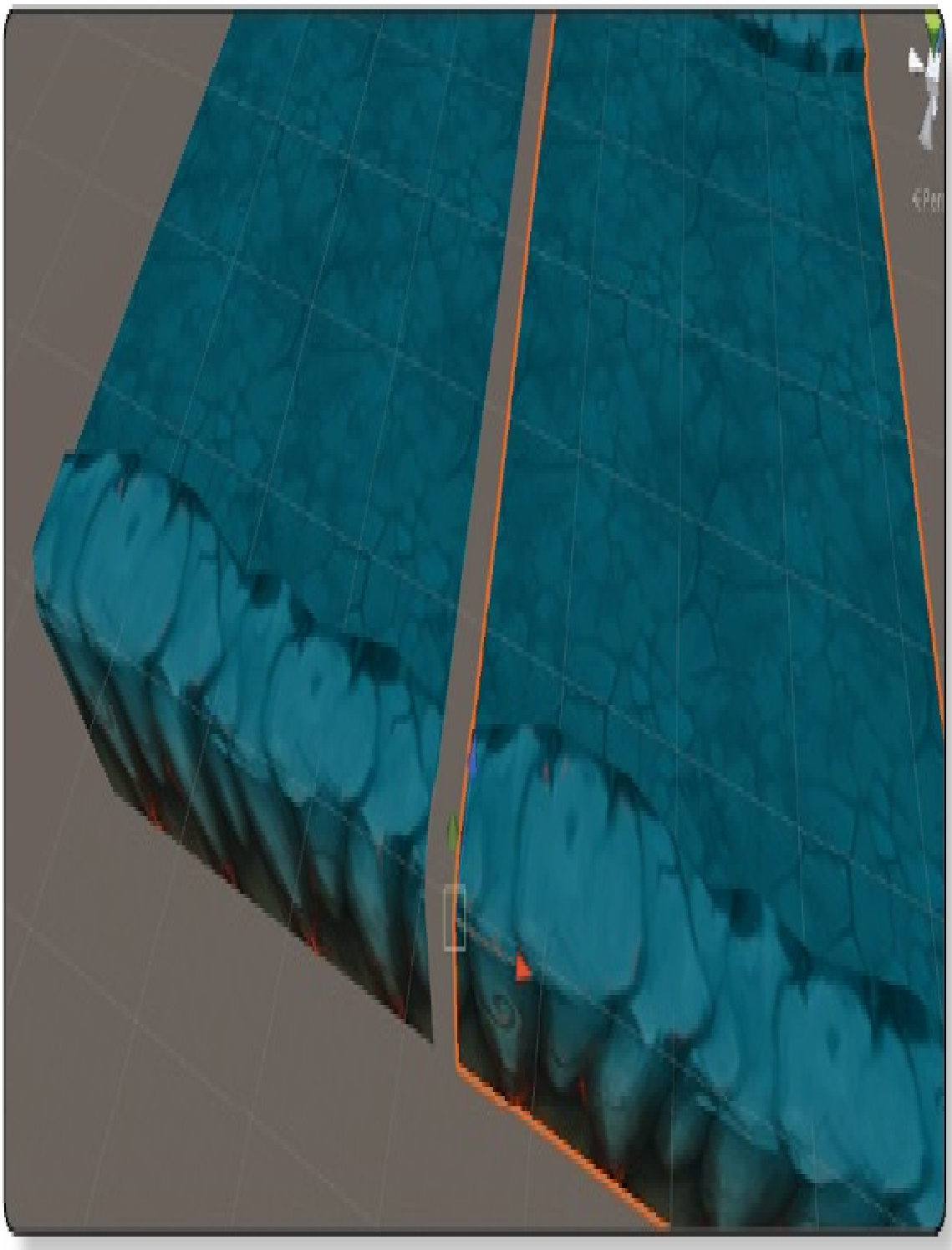
Una vez duplicado el elemento deberá de tomar su herramienta “Move Tool” y lo más recomendable es ir lo moviendo por eje, es decir tomar la flecha relativa al eje X y moverlo con ella para lograr el resultado que se muestra.



Duplicado de objetos

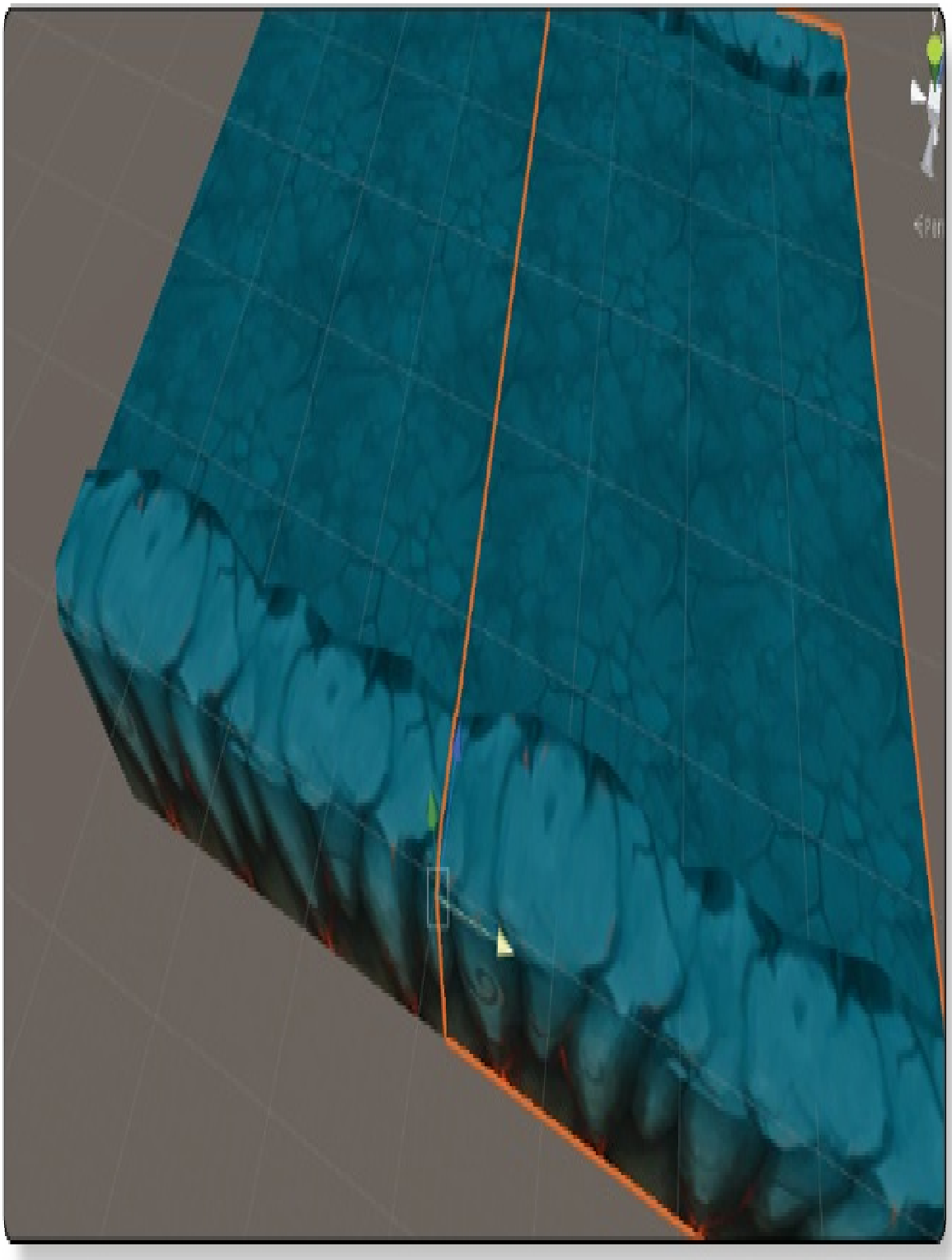
Para hacer “Snap” entre dos objetos:

Con la tecla “V” del teclado seleccionar una de las esquinas del objeto como se muestra.



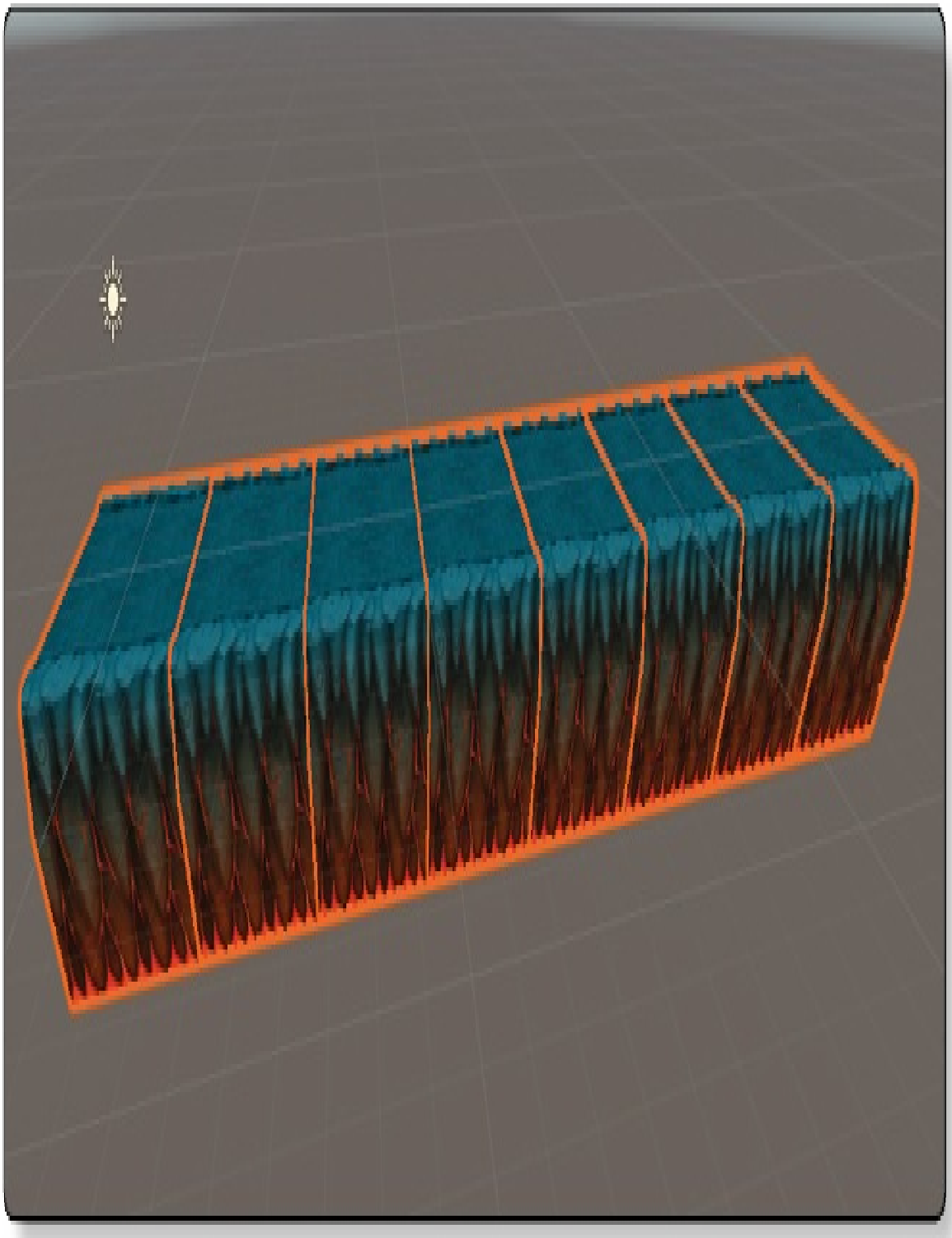
Ejemplo de Snap

Junte los objetos como se muestra sin soltar la tecla “V”.



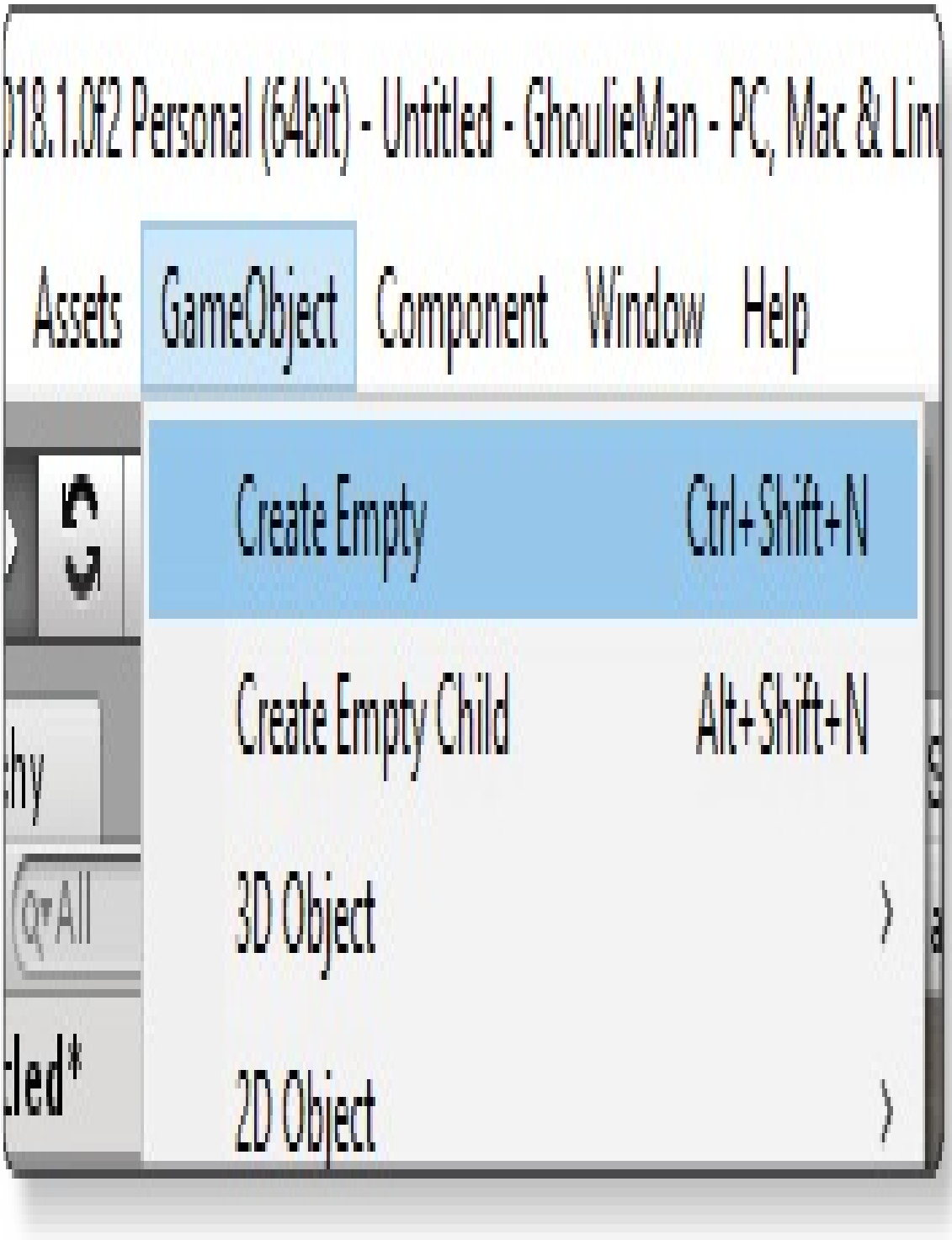
Objetos con Snap

Haga 7 duplicados y colóquelos con snap.



Duplicado de plataformas

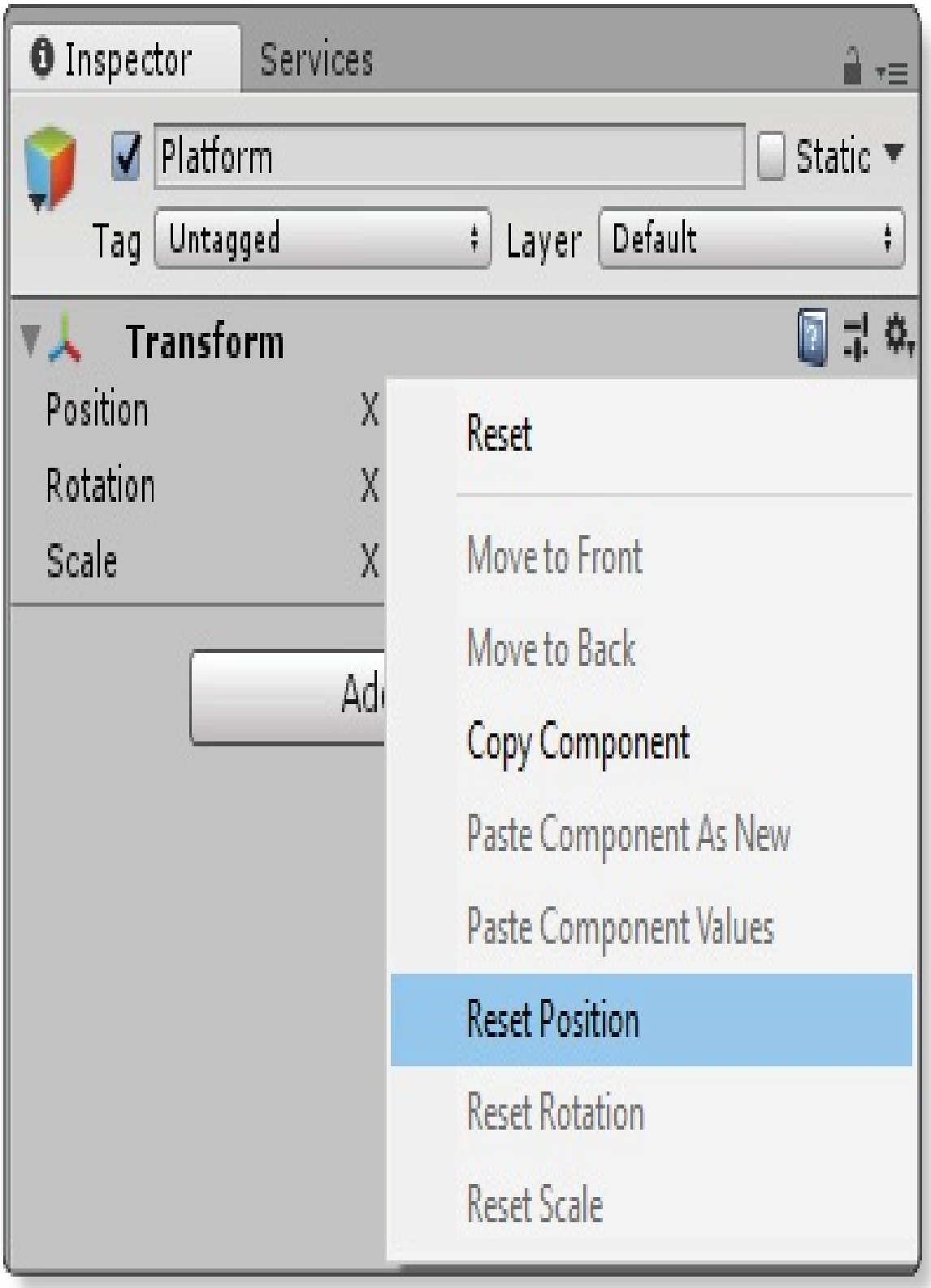
Para hacer un nuevo objeto nuevo (“New Empty Object”), vaya al menú general y asigne “GameObject / Create Empty”.


018.1.0f2 Personal (64bit) - Untitled - GhoulieMan - PC, Mac & Linu
Assets
GameObject
Component
Window
Help
Create Empty
Ctrl+Shift+N
Create Empty Child
Alt+Shift+N
3D Object
2D Object
Qr:All
cled*

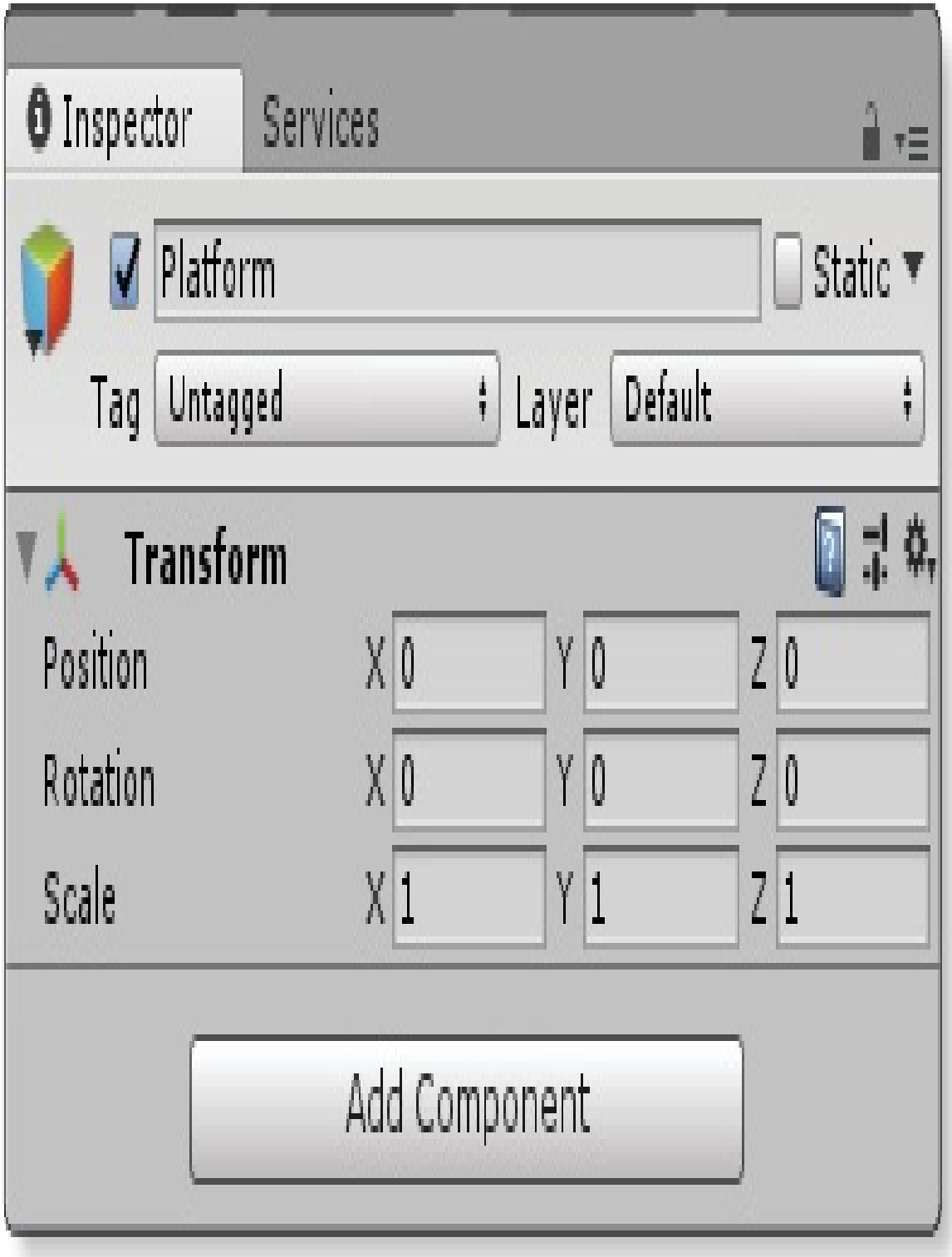
Creación de un “Empty Object”

Haga un nuevo “Empty Object”, renómbrelo “Platform”

Reset sus transformaciones de posición a 0,0,0 haciendo clic en el engrane superior derecho de la ventana “Transform” de “Inspector”.


Inspector
Services
Platform
Static
Tag
Untagged
Layer
Default
Transform
Position
Rotation
Scale
Reset
Move to Front
Move to Back
Copy Component
Paste Component As New
Paste Component Values
Reset Position
Reset Rotation
Reset Scale

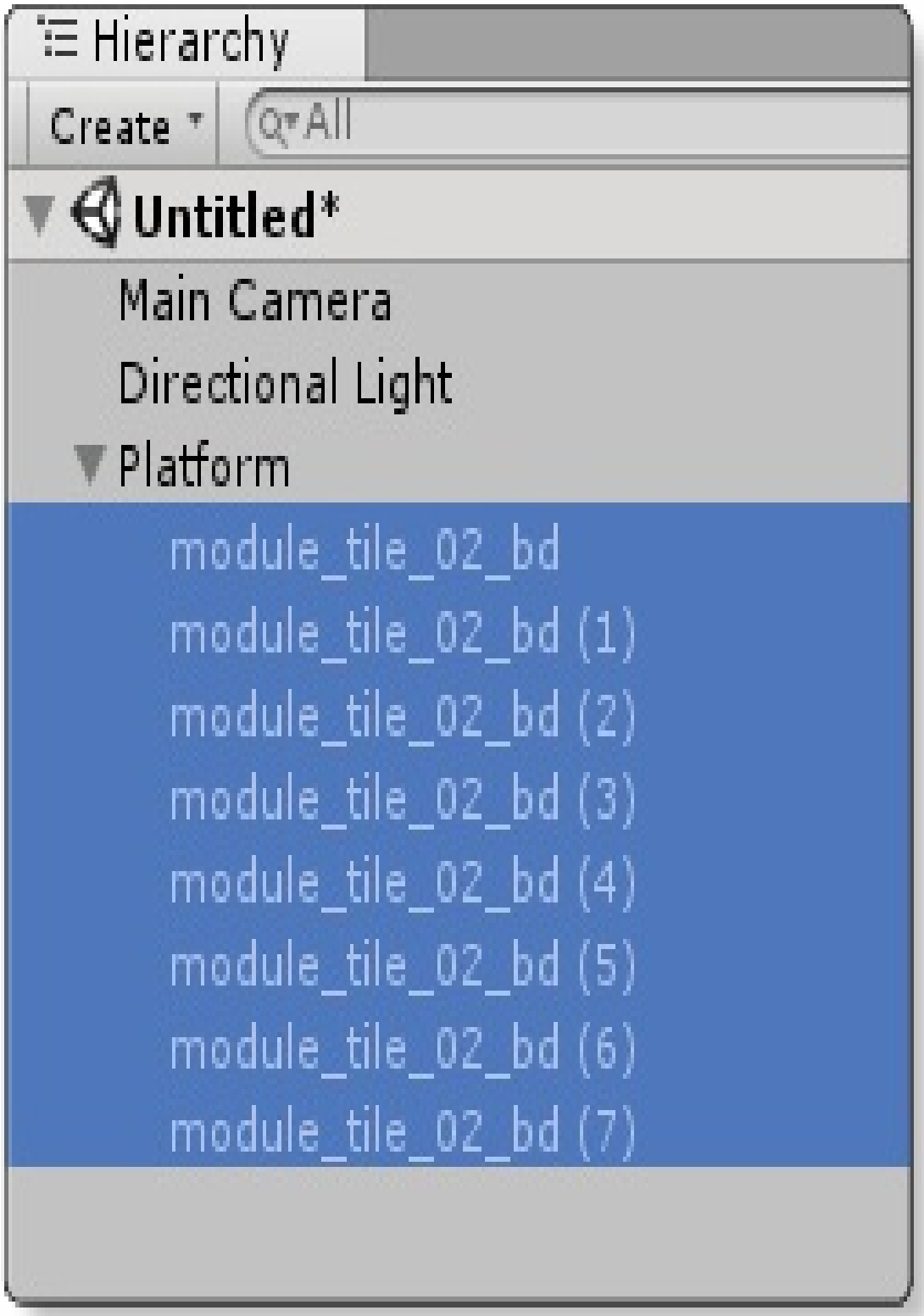
”Reset” de transformaciones.


Inspector
Services
Platform
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Add Component

Transformaciones “reseteadas”

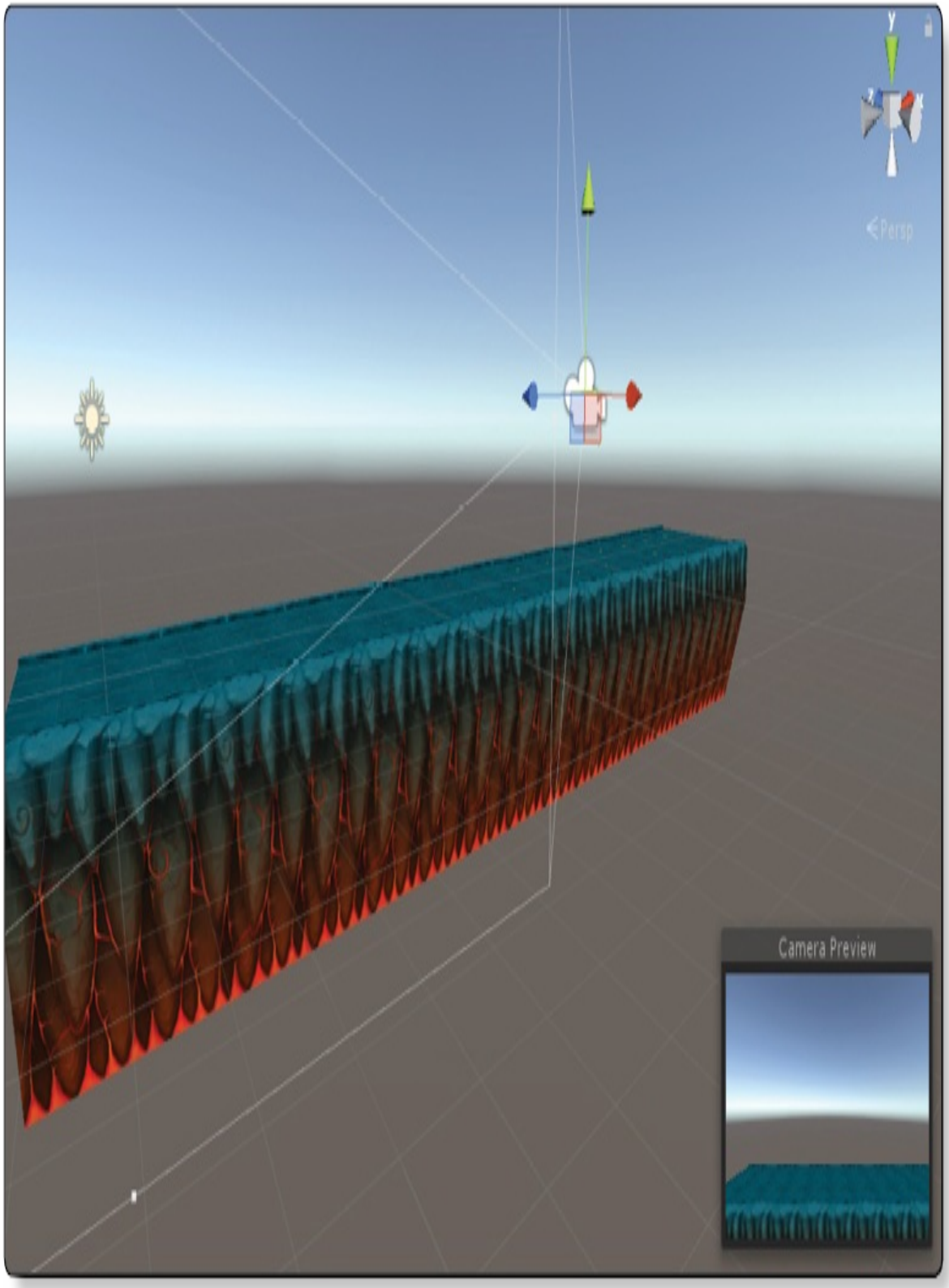
Para asignar un objeto como “hijo” de otro; en la ventana “Hierarchy” desplace el elemento que queremos como hijo sobre del que queremos que sea el padre.

Asigne los objetos plataformas como hijos del objeto “Platform”.


Hierarchy
Create
All
Untitled*
Main Camera
Directional Light
Platform
module_tile_02_bd
module_tile_02_bd (1)
module_tile_02_bd (2)
module_tile_02_bd (3)
module_tile_02_bd (4)
module_tile_02_bd (5)
module_tile_02_bd (6)
module_tile_02_bd (7)

Emparentamiento de objetos

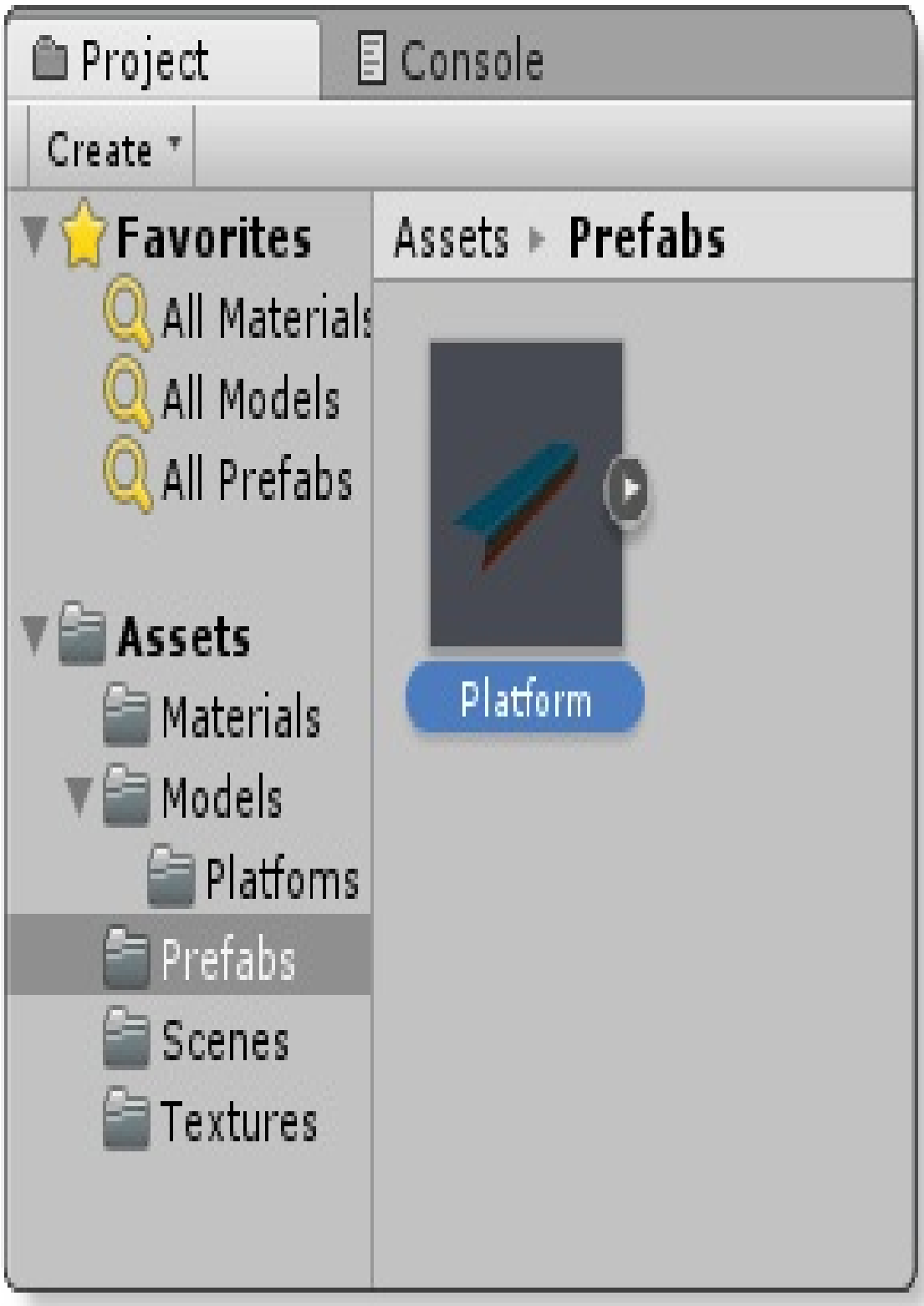
Coloque la cámara en la vista del juego.


y
Persp
Camera Preview

Reposición de la cámara

Los PREFABS son de suma importancia, ya que a través de ellos se nos permite crear una instancia de cualquier elemento y poderlo reutilizar cuantas veces necesitemos.

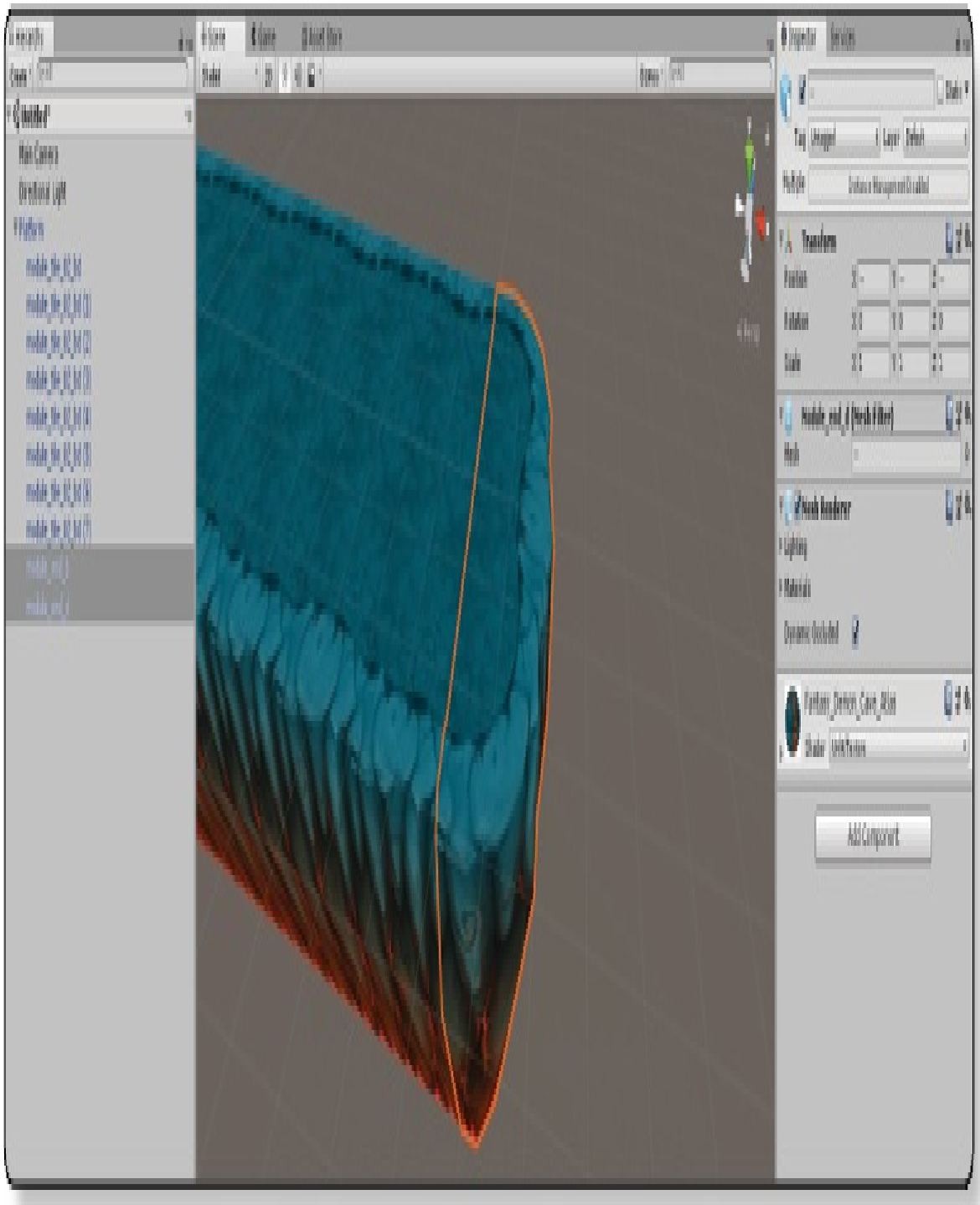
Haga un nuevo folder, nómbrelo “Prefabs” y arrastre a él el objeto “Platform”.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Platforms
Prefabs
Scenes
Textures
Assets ▸ Prefabs
Platform

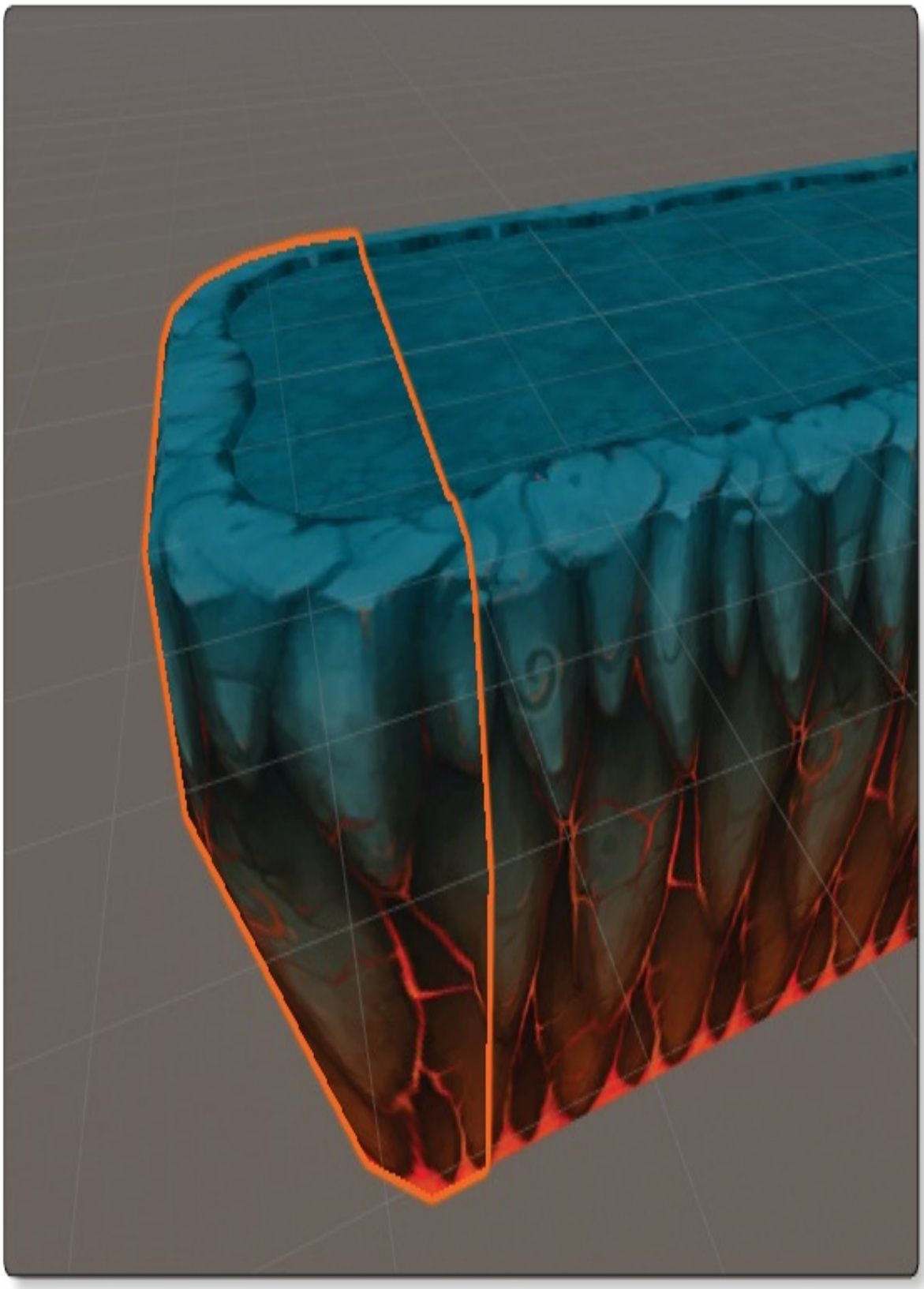
Creación de Prefabs

RETO AL LECTOR

Con los conocimientos adquiridos importe, texturice, acomode y asigne como hijos de Platform los siguientes elementos.



Plataforma completa derecha



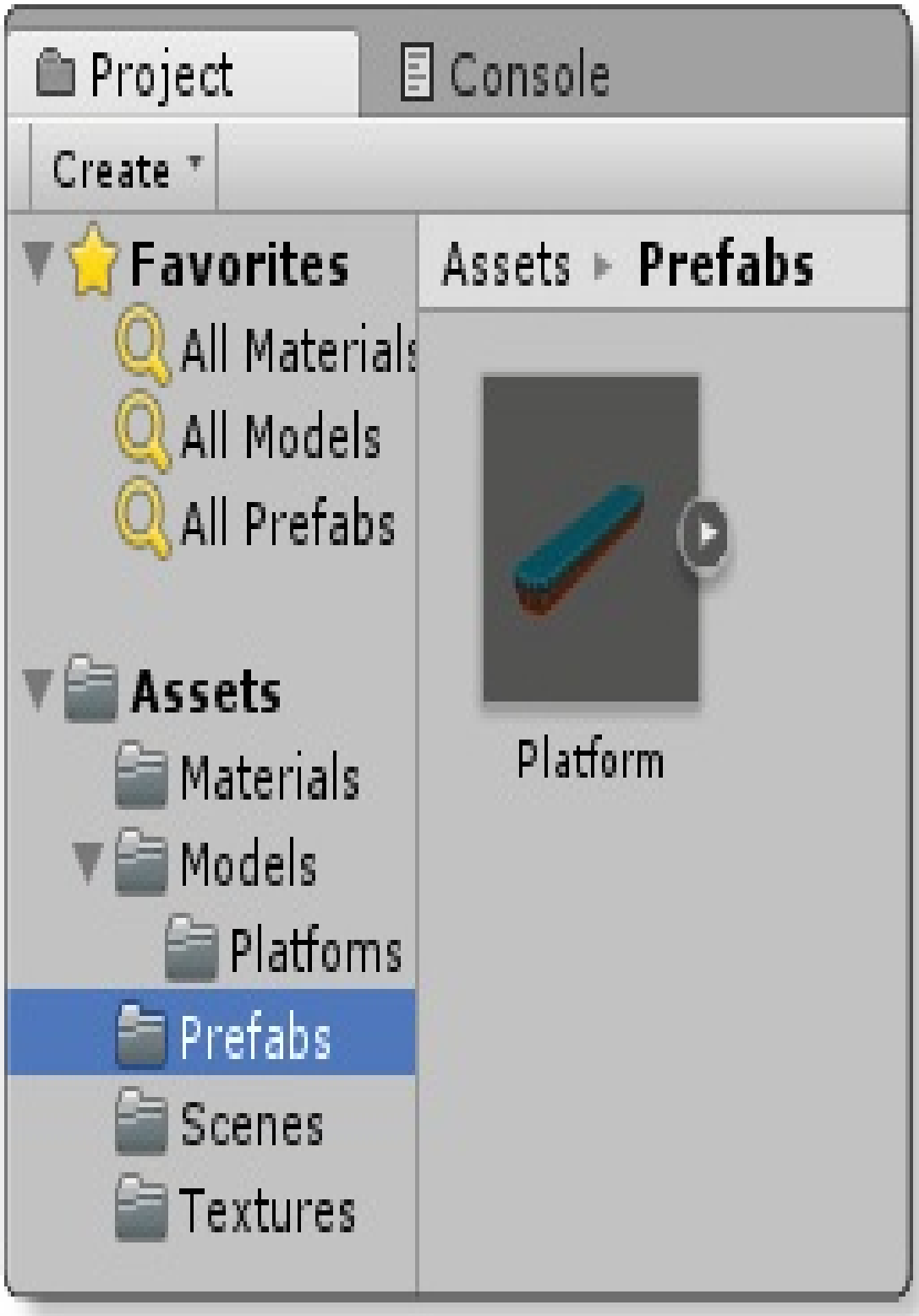
Plataforma completa izquierda

NOTA:

Cuando modificamos un “Prefab” en la ventana de “Scene”, será necesario seleccionar desde “Hierarchy” el elemento “Prefab” (en este caso “Platform”) y asignar el botón de “Apply” en “Inspector para que los nuevos cambios se realicen en el prefab original y en todas sus instancias.

Inspector
Services
Platform
Static
Tag
Untagged
Layer
Default
Prefab
Select
Revert
Apply

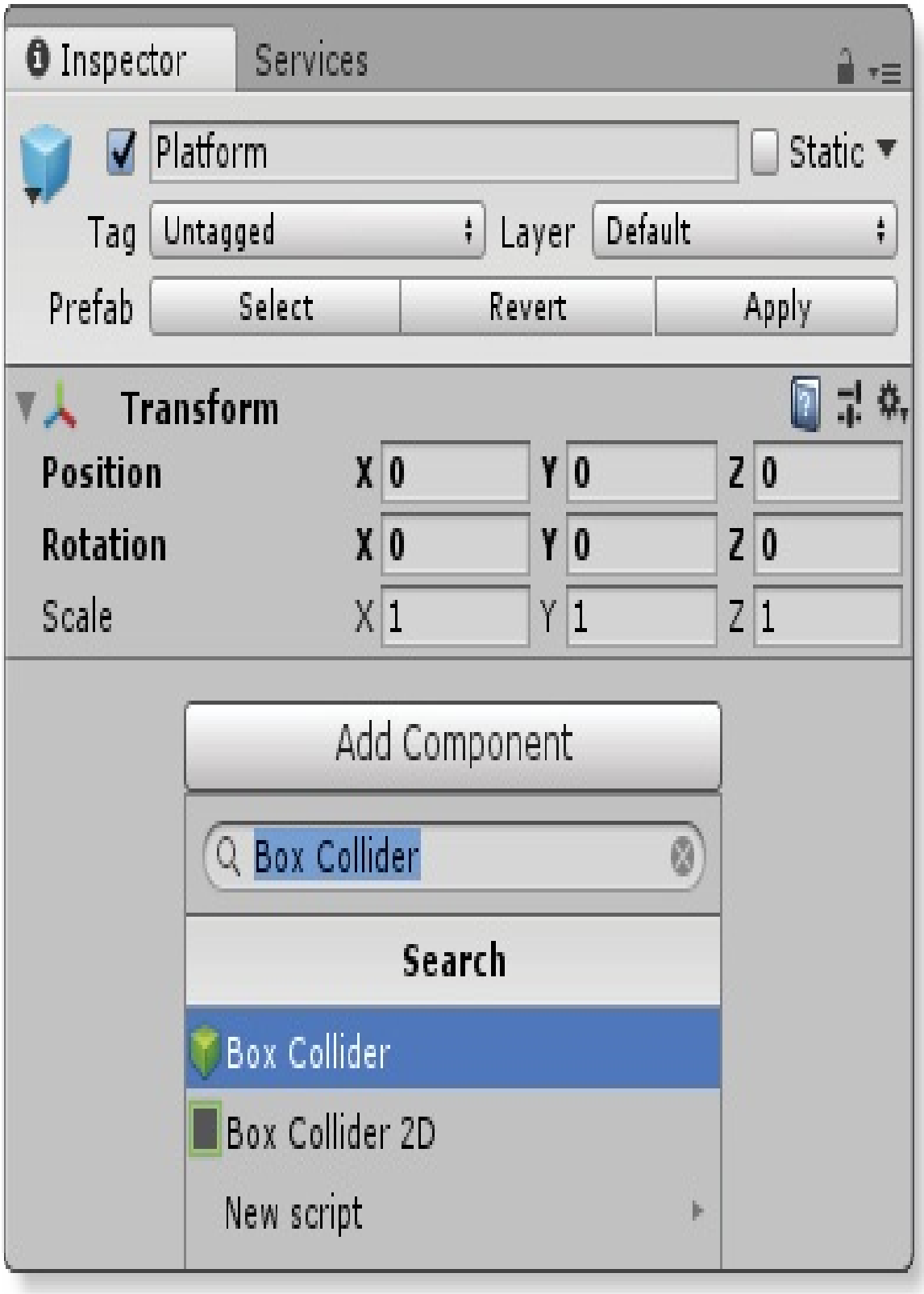
Aplicar cambios para el Prefab


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Platforms
Prefabs
Scenes
Textures
Assets ▸ Prefabs
Platform

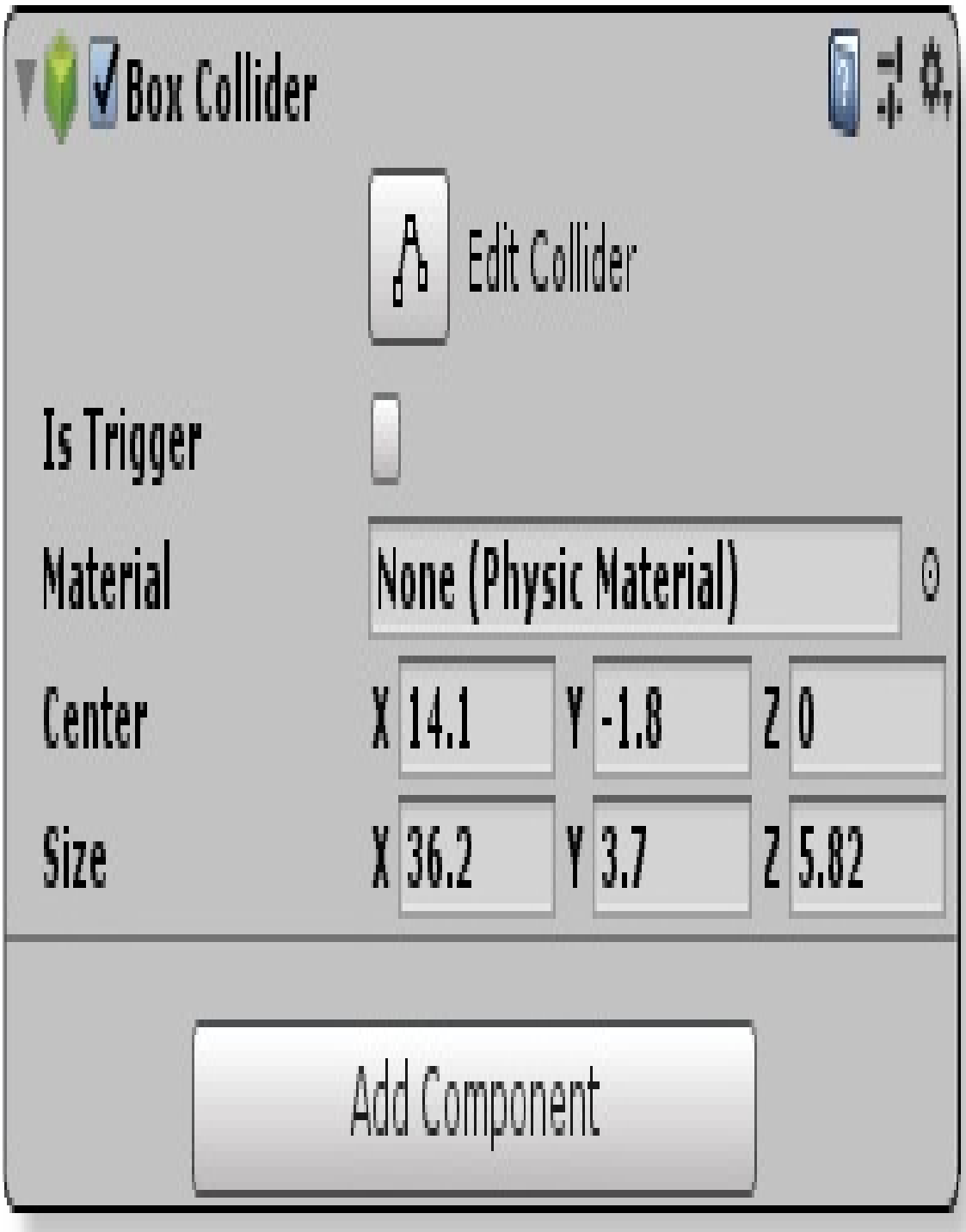
Cambios aplicados al Prefab

Añadir un componente tipo “Collider” nos permite asignar propiedades colindantes a los objetos en relación a otros objetos que también tengan sus respectivos “Colliders”. Existen tipo caja, esfera, capsula o “mesh”, en este caso utilizaremos el básico que es tipo “Box”.

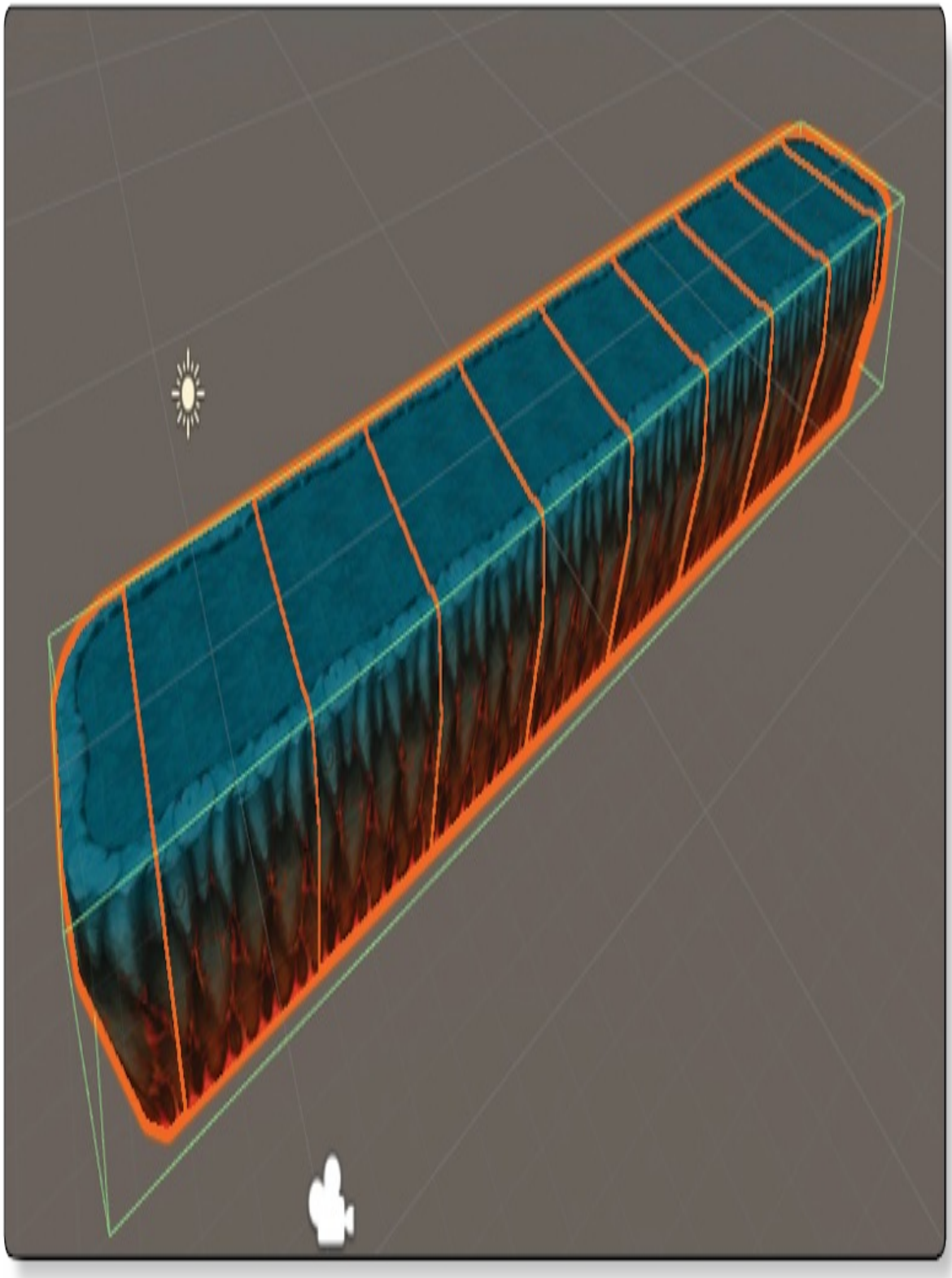
Añada un componente “BoxCollider” con los ajustes mostrados


Inspector
Services
Platform
Static
Tag
Untagged
Layer
Default
Prefab
Select
Revert
Apply
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Add Component
Box Collider
Search
Box Collider
Box Collider 2D
New script

Componente Box Collider


Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 14.1
Y -1.8
Z 0
Size
X 36.2
Y 3.7
Z 5.82
Add Component

Parámetros del Box Collider



Collider debidamente formateado

Dentro de los “Settings” de los colliders podemos encontrar:

Botón de Edit Collider: Para editar de forma gráfica la forma del “Collider” con puntos de edición

Is Trigger: Para asignarle una propiedad de detonante de una acción específica. Cuando el “Collider” es trigger, este pierde su propiedad física colindante y se convierte en un objeto “transparente” pero detonante

Center: Con estos parámetros podemos mover el “Collider” a nuestro gusto en vez de utilizar “Move Tool”

Size: Con estos parámetros podemos escalar el “Collider” a nuestro gusto en vez de utilizar “Scale Tool”

Clic en el botón de APPY para que los cambios se asignen en el Prefab.

TRABAJO CON PLAYER

INTRODUCCIÓN

En este capítulo trabajaremos con nuestro personaje principal que llamaremos “Player”.

Importaremos nuestro modelo externo desde nuestra carpeta de “resources sección 02”, haremos su material para texturizarlo, se explicará el modo de “Scripting” con el lenguaje C#, veremos el concepto y la funcionalidad de las variables y métodos.

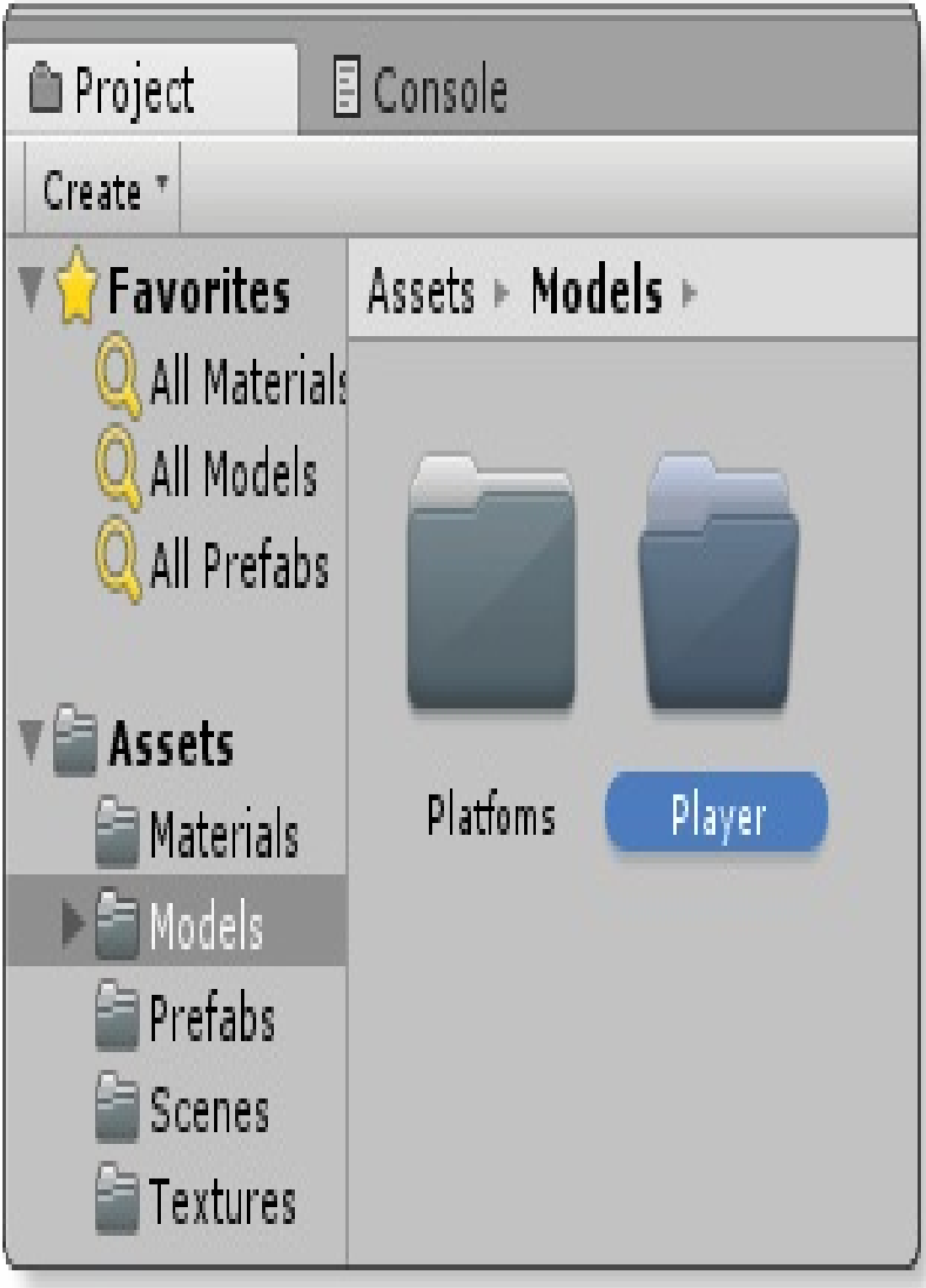
A su vez también trabajaremos con el movimiento de nuestro Player, su animación de corrido, su funcionalidad de salto y su animación relativa.

Programación y animación de su funcionalidad de ataque, así como de sus proyectiles y el arreglo de algunos “bugs” que su videojuego pudiera llegar a presentar.

PREPARACION DEL PLAYER

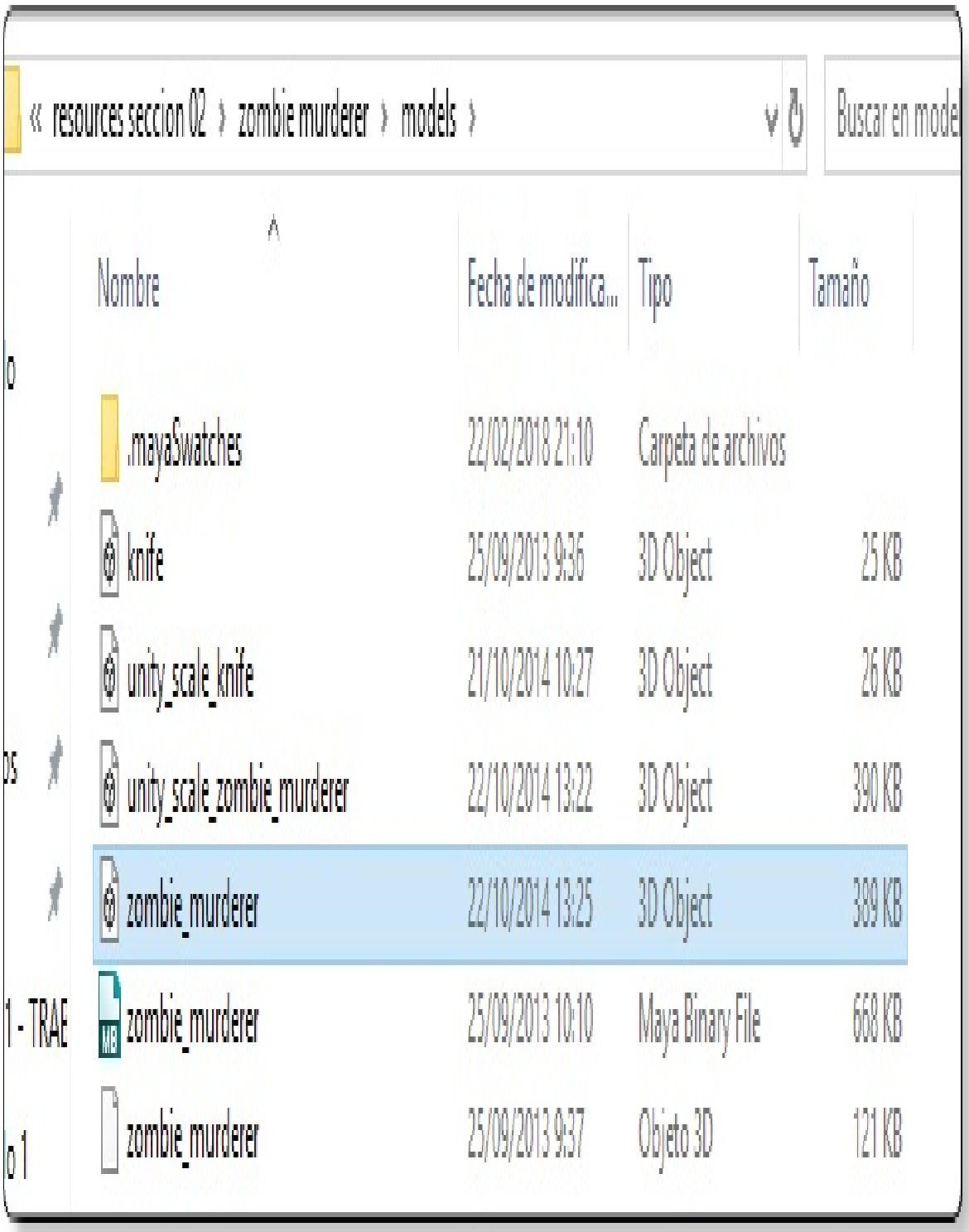
En este apartado importaremos el modelo de nuestro Player y prepararemos todos sus elementos necesarios para que sea funcional dentro de Unity y comenzar a programar sus funcionalidades.

Vaya al folder de models, haga un nuevo folder y nómbrelo “Player”.

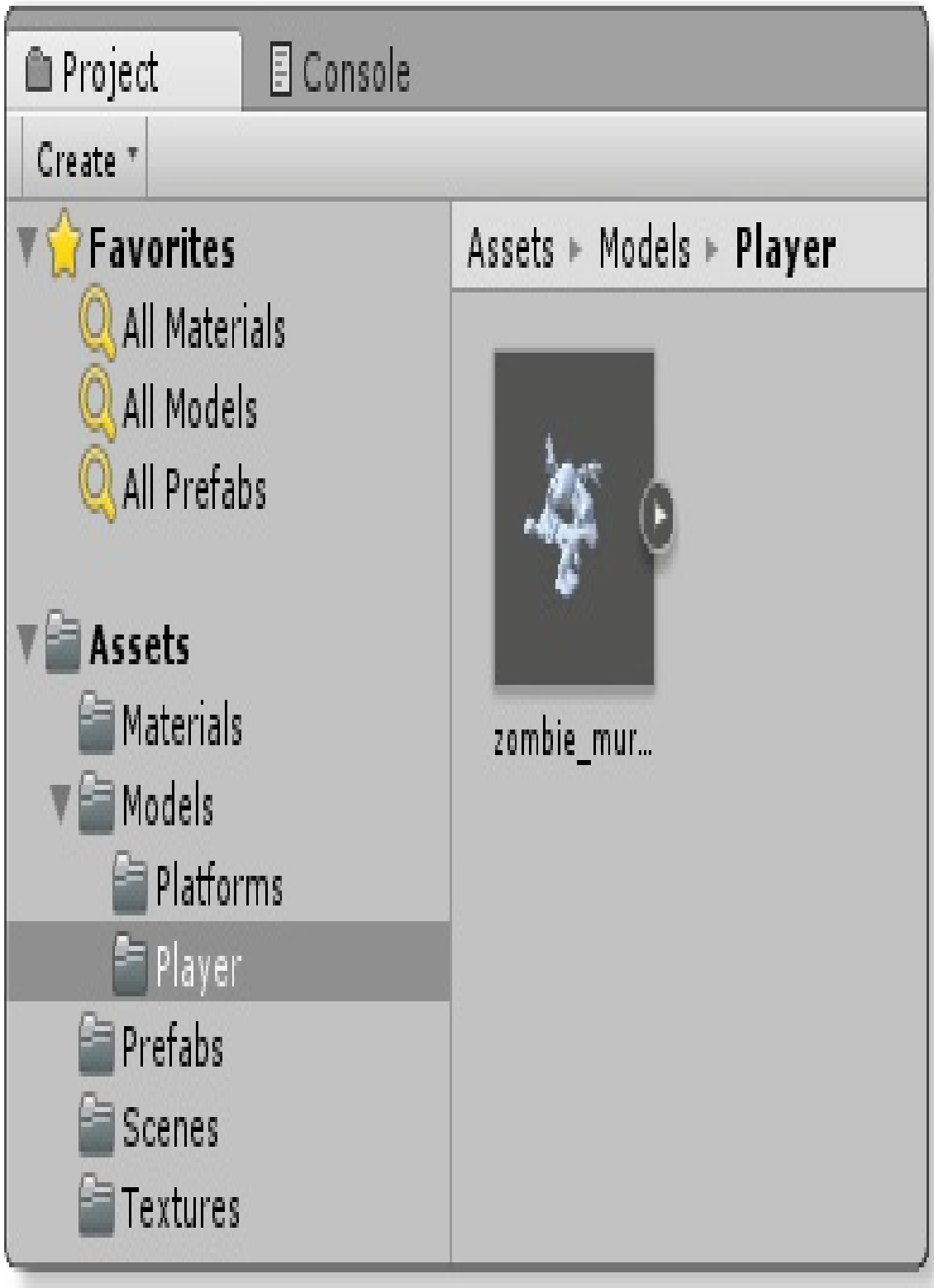

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Prefabs
Scenes
Textures
Assets ▸ Models ▸
Platforms
Player

Folder Player

Importe dentro del folder Player el objeto “zombie_murderer” de la carpeta de “resources seccion 02”.


« resources seccion 02 › zombie murderer › models ›
Buscar en model
Nombre
Fecha de modifica...
Tipo
Tamaño
.maya$watches
22/02/2018 21:10
Carpeta de archivos
knife
25/09/2013 9:36
3D Object
25 KB
unity_scale_knife
21/10/2014 10:27
3D Object
26 KB
unity_scale_zombie_murderer
22/10/2014 13:22
3D Object
390 KB
zombie_murderer
22/10/2014 13:25
3D Object
389 KB
zombie_murderer
25/09/2013 10:10
Maya Binary File
668 KB
zombie_murderer
25/09/2013 9:37
Objeto 3D
121 KB

Folder resources seccion 02

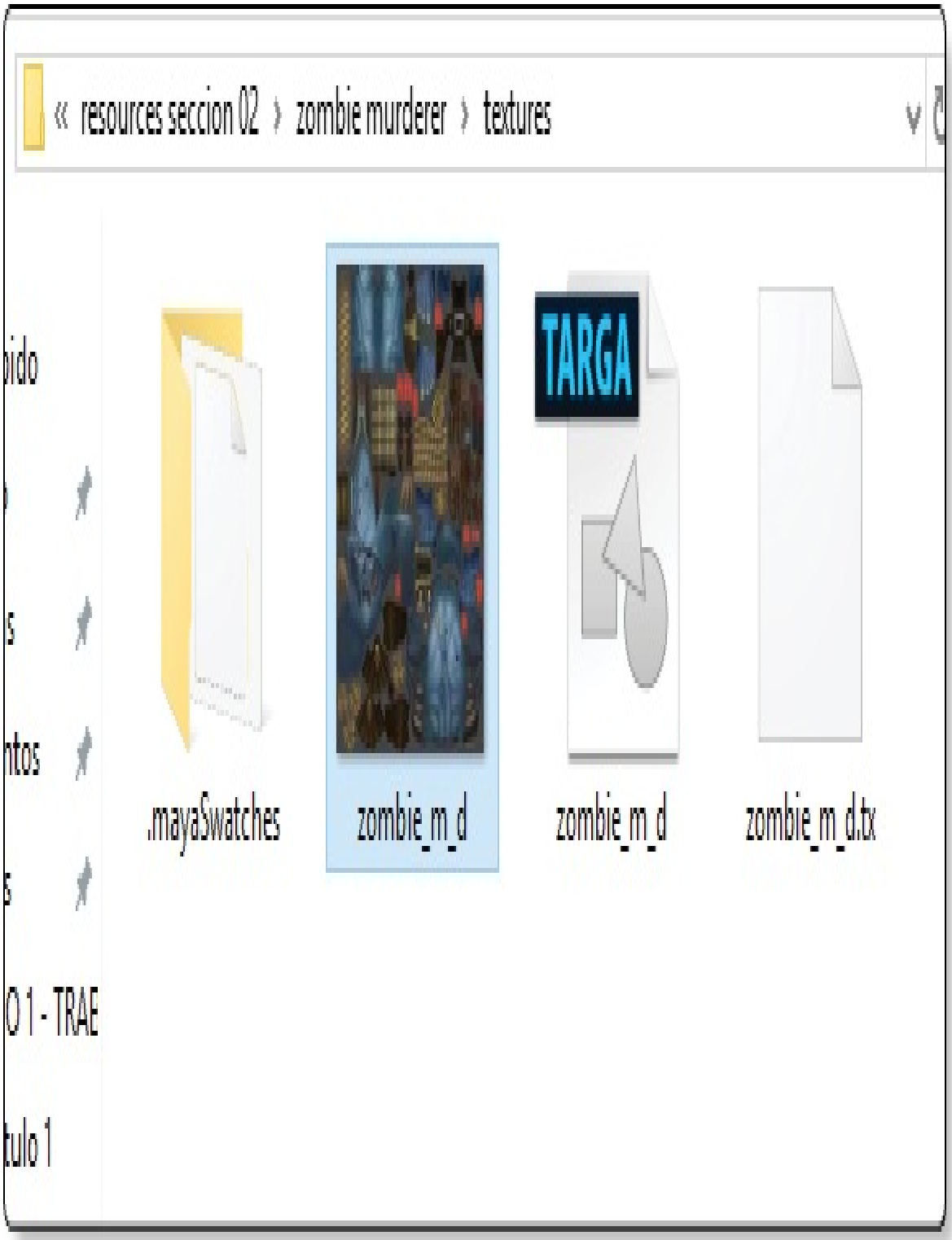

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Platforms
Player
Prefabs
Scenes
Textures
Assets ▸ Models ▸ Player
zombie_mur...

Importación del modelo

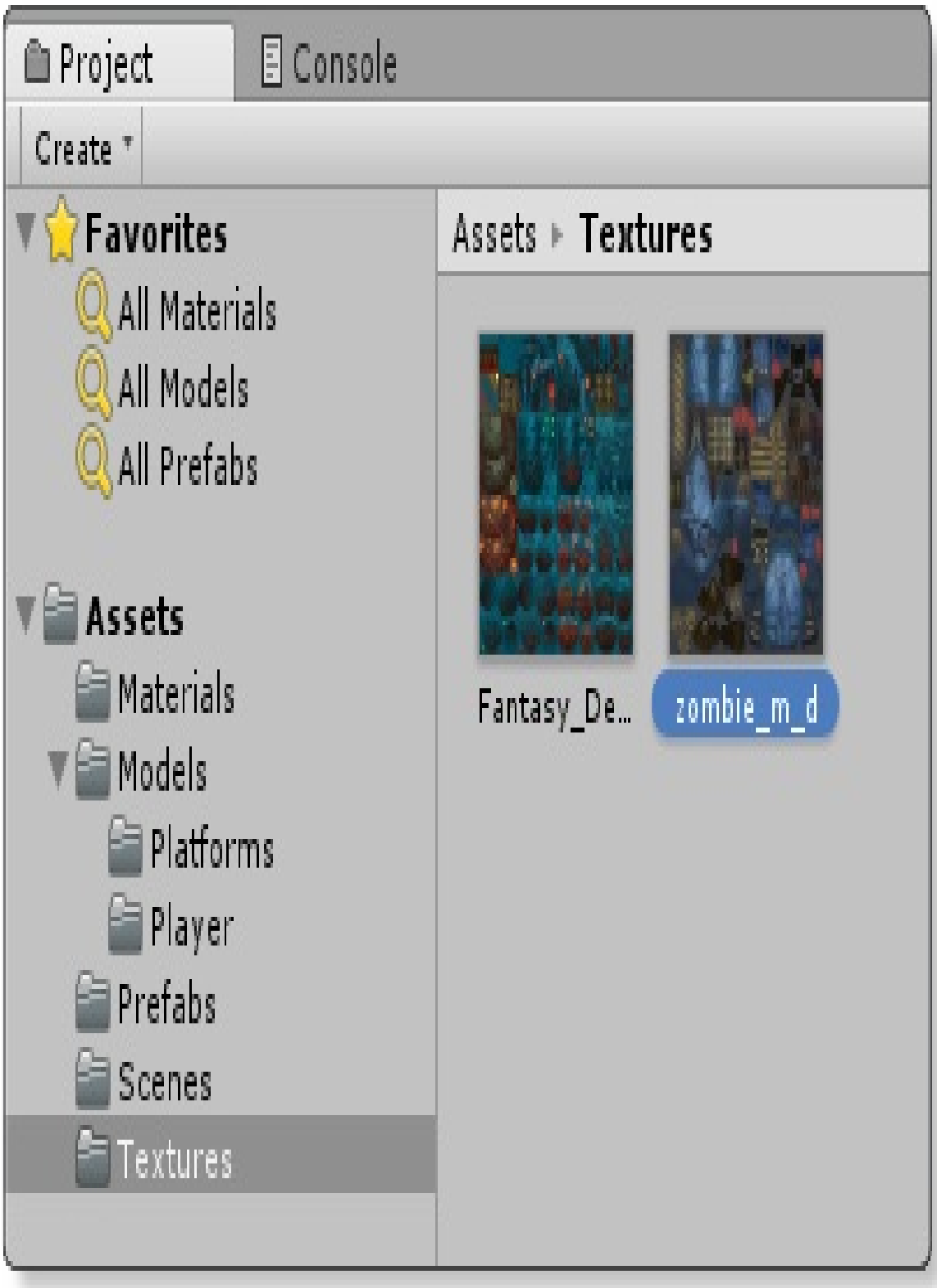
NOTA:

Puede importar arrastrando desde la ventana externa al folder de “Player” en Unity.

Vaya al folder “Textures” e importe la textura.

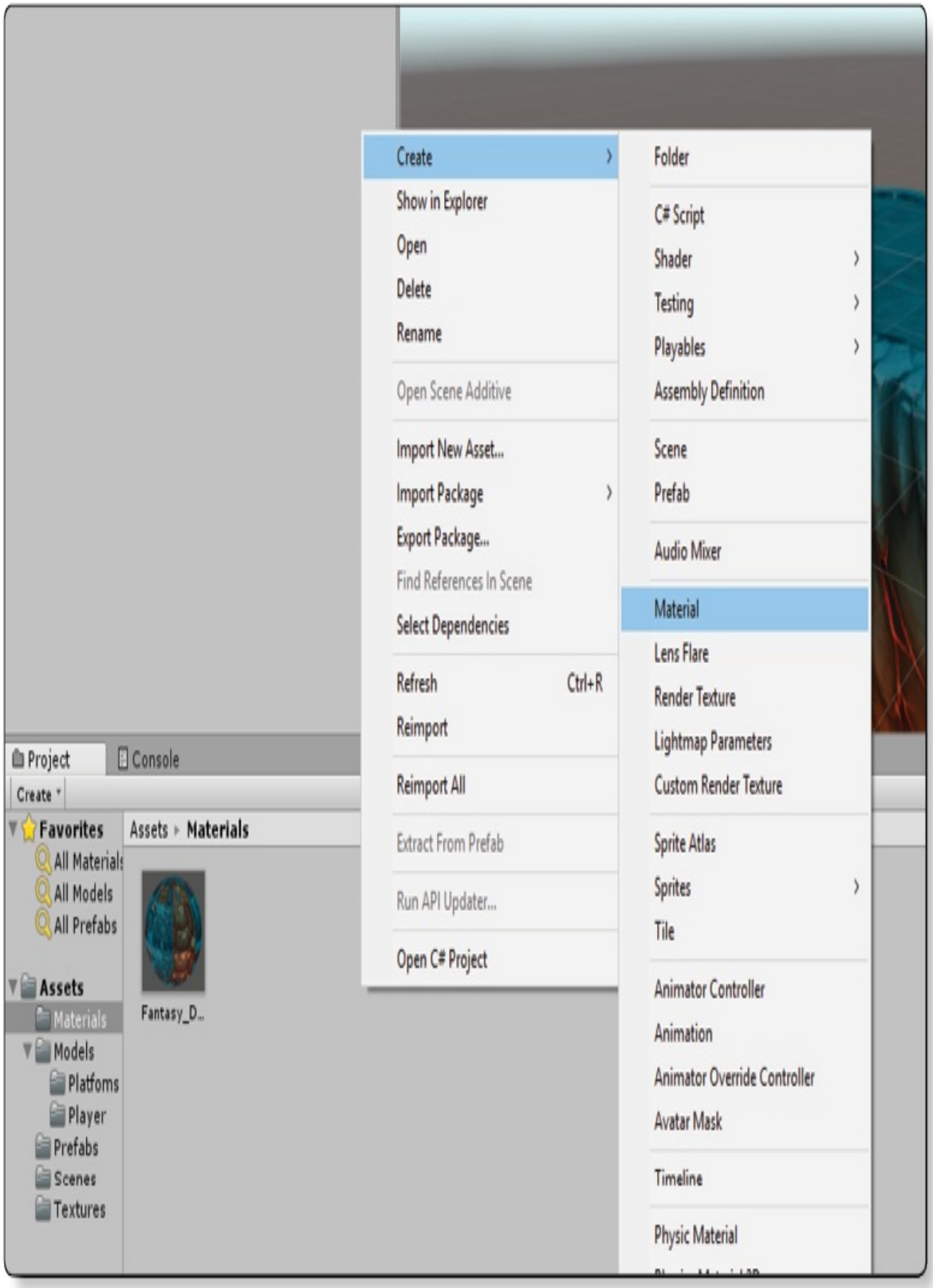

« resources seccion 02 › zombie murderer › textures
.mayaSwatches
zombie_m_d
TARGA
zombie_m_d
zombie_m_d.tx

Folder resources seccion 02

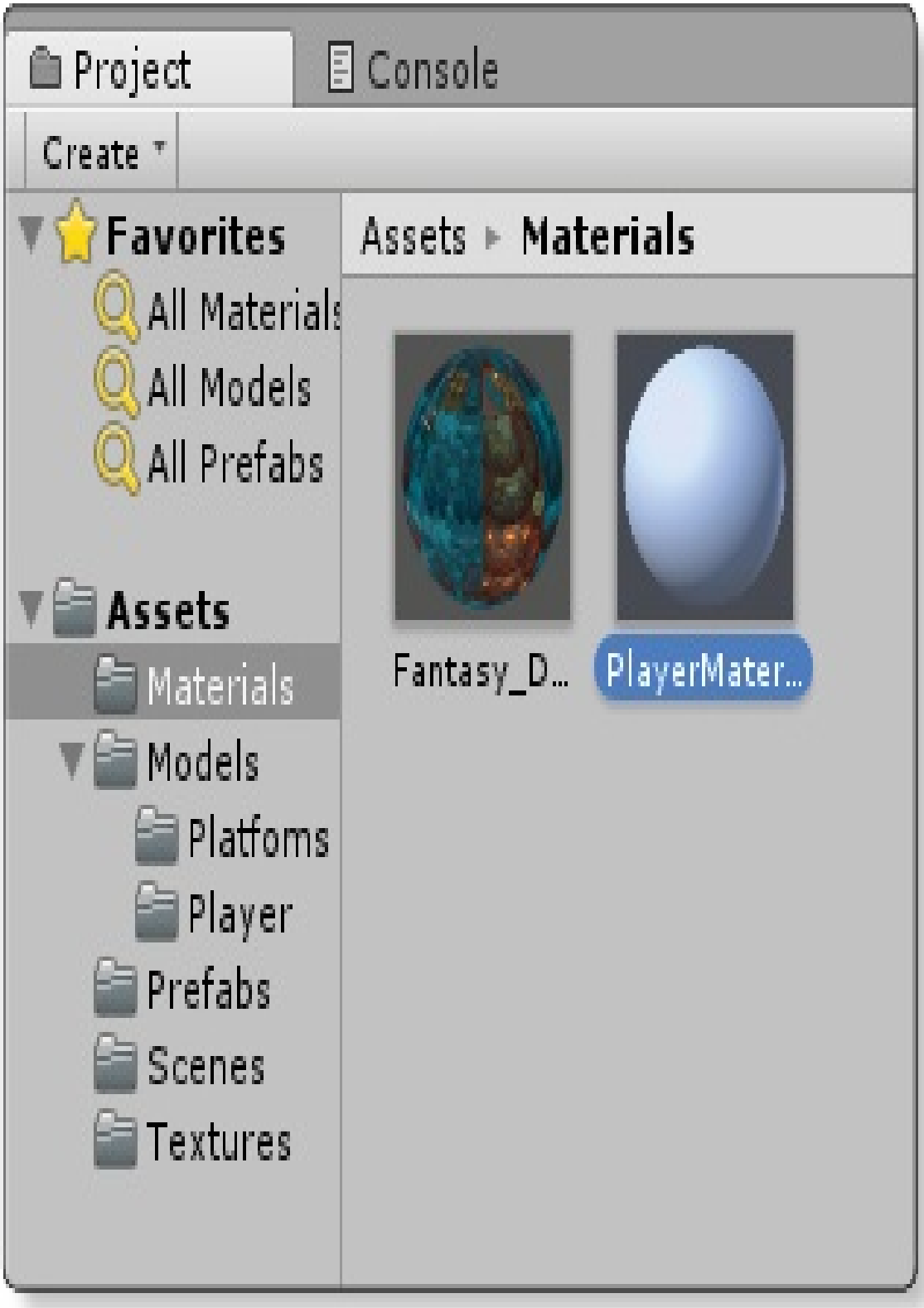

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Platforms
Player
Prefabs
Scenes
Textures
Assets ▸ Textures
Fantasy_De...
zombie_m_d

Importar la textura

Vaya al folder de “Materials”, haga un nuevo material (con clic derecho “new material” y renómbrelo como “PlayerMaterial”.

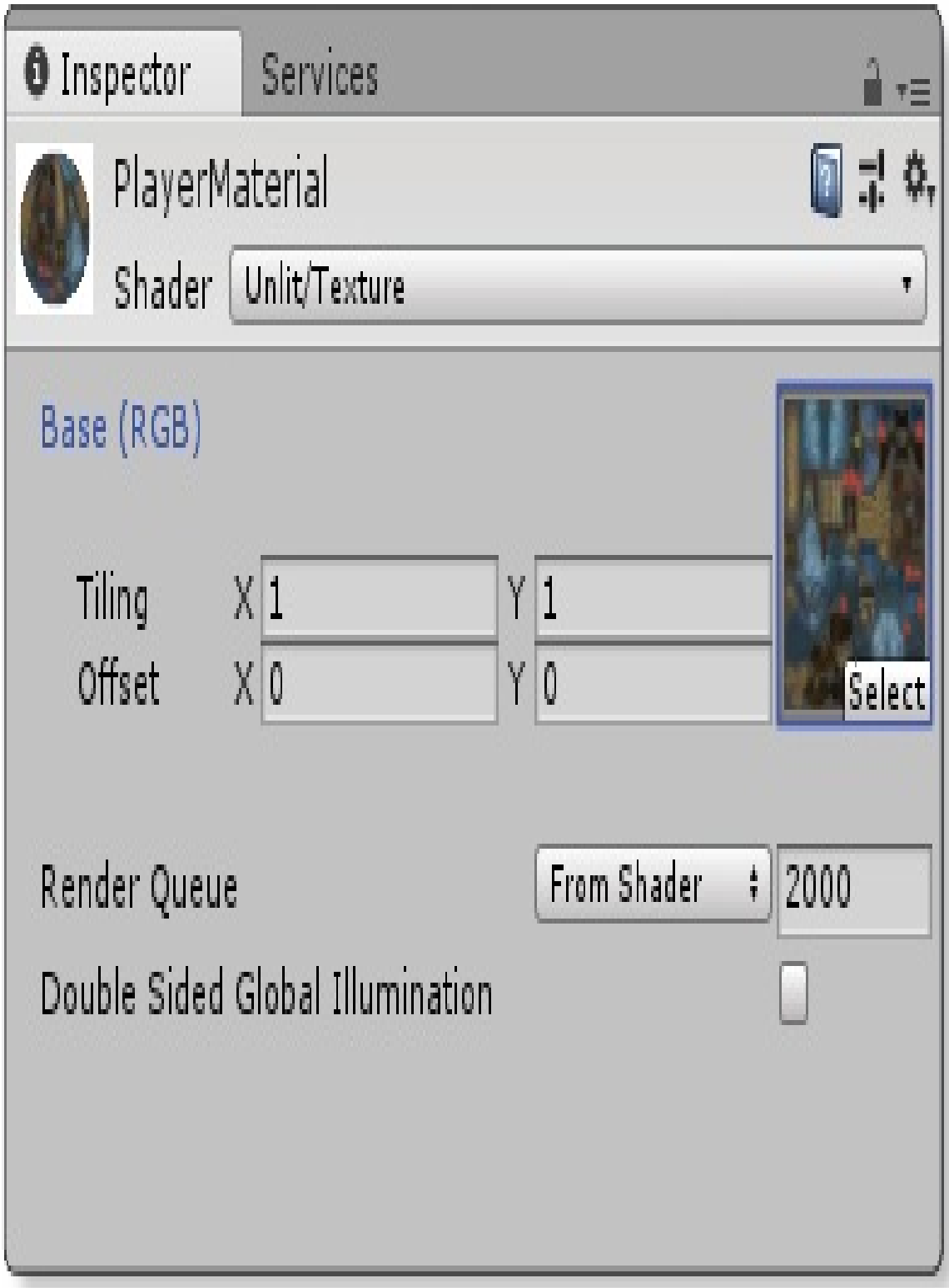

Create
Show in Explorer
Open
Delete
Rename
Open Scene Additive
Import New Asset...
Import Package
Export Package...
Find References In Scene
Select Dependencies
Refresh Ctrl+R
Reimport
Reimport All
Extract From Prefab
Run API Updater...
Open C# Project
Folder
C# Script
Shader
Testing
Playables
Assembly Definition
Scene
Prefab
Audio Mixer
Material
Lens Flare
Render Texture
Lightmap Parameters
Custom Render Texture
Sprite Atlas
Sprites
Tile
Animator Controller
Animation
Animator Override Controller
Avatar Mask
Timeline
Physic Material
Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Platforms
Player
Prefabs
Scenes
Textures
Assets ▸ Materials
Fantasy_D...

Creación del material


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Platfoms
Player
Prefabs
Scenes
Textures
Assets ▸ Materials
Fantasy_D...
PlayerMater...

Material PlayerMaterial

Seleccione el material “PlayerMaterial” y en la ventana de “Inspector” asigne los siguientes valores para aplicar el mapa de textura al material y que sea de tipo “Unlit”.

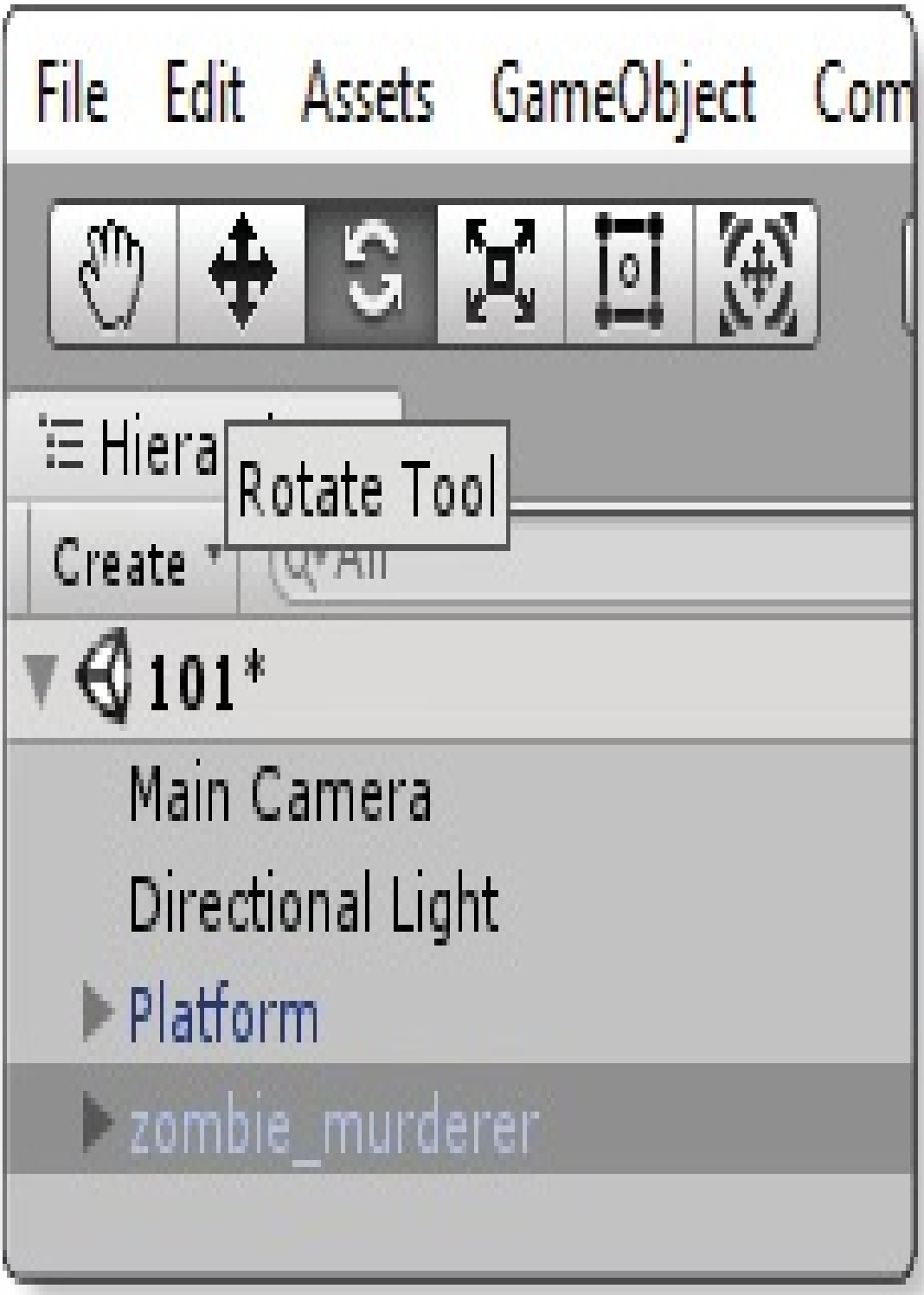

Inspector
Services
PlayerMaterial
Shader
Unlit/Texture
Base (RGB)
Tiling
X 1
Y 1
Offset
X 0
Y 0
Select
Render Queue
From Shader
2000
Double Sided Global Illumination

Valores para el material

Arrastre el modelo del Player a la escena y rótelo con la herramienta de rotación.

Scene
Game
Asset Store
Animator
Shaded
2D
Gizmos
All
y
x

Modelo importado sin material

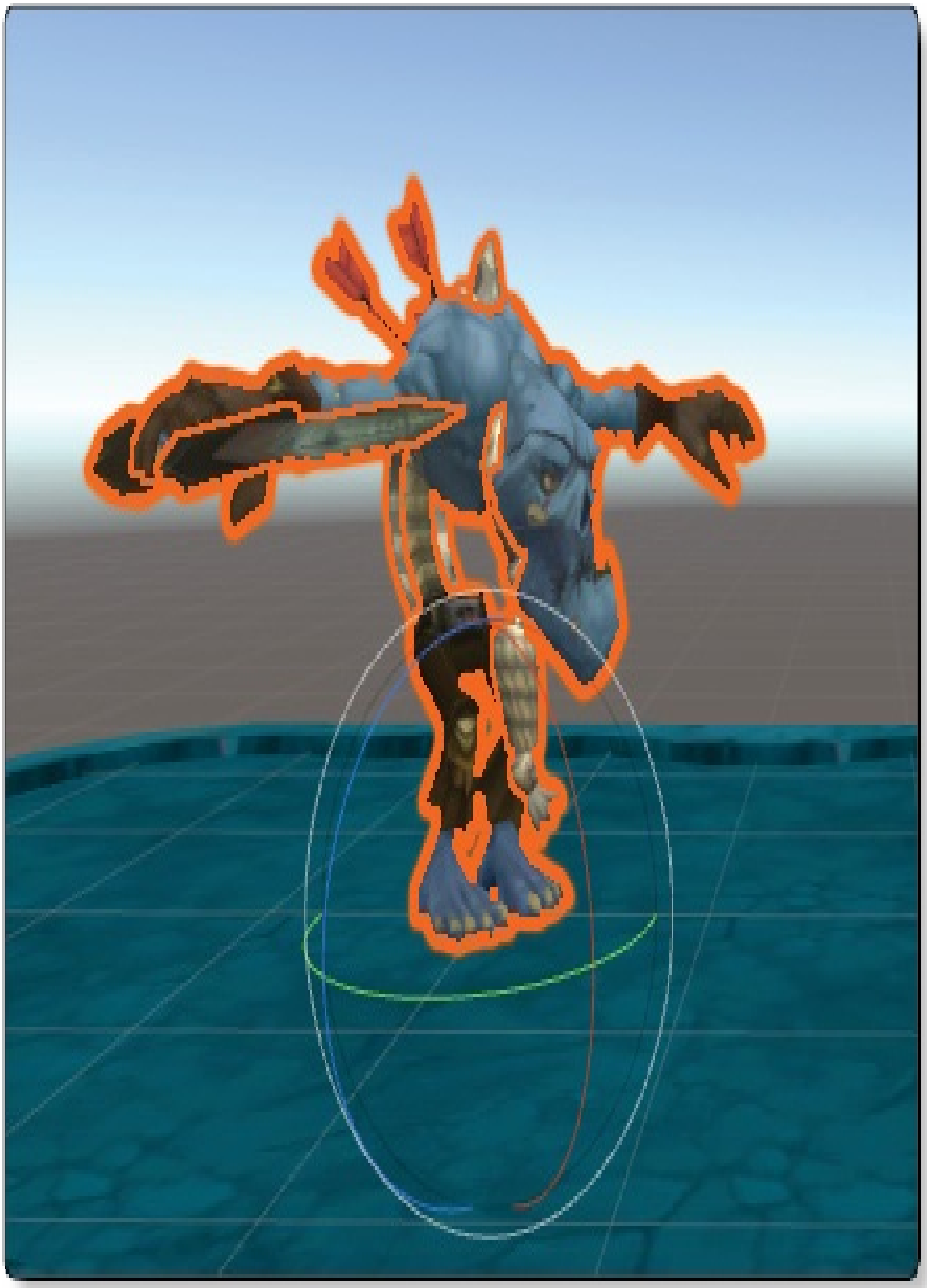

File Edit Assets GameObject Com
Hiera
Rotate Tool
Create
101*
Main Camera
Directional Light
Platform
zombie_murderer

Herramienta de rotación

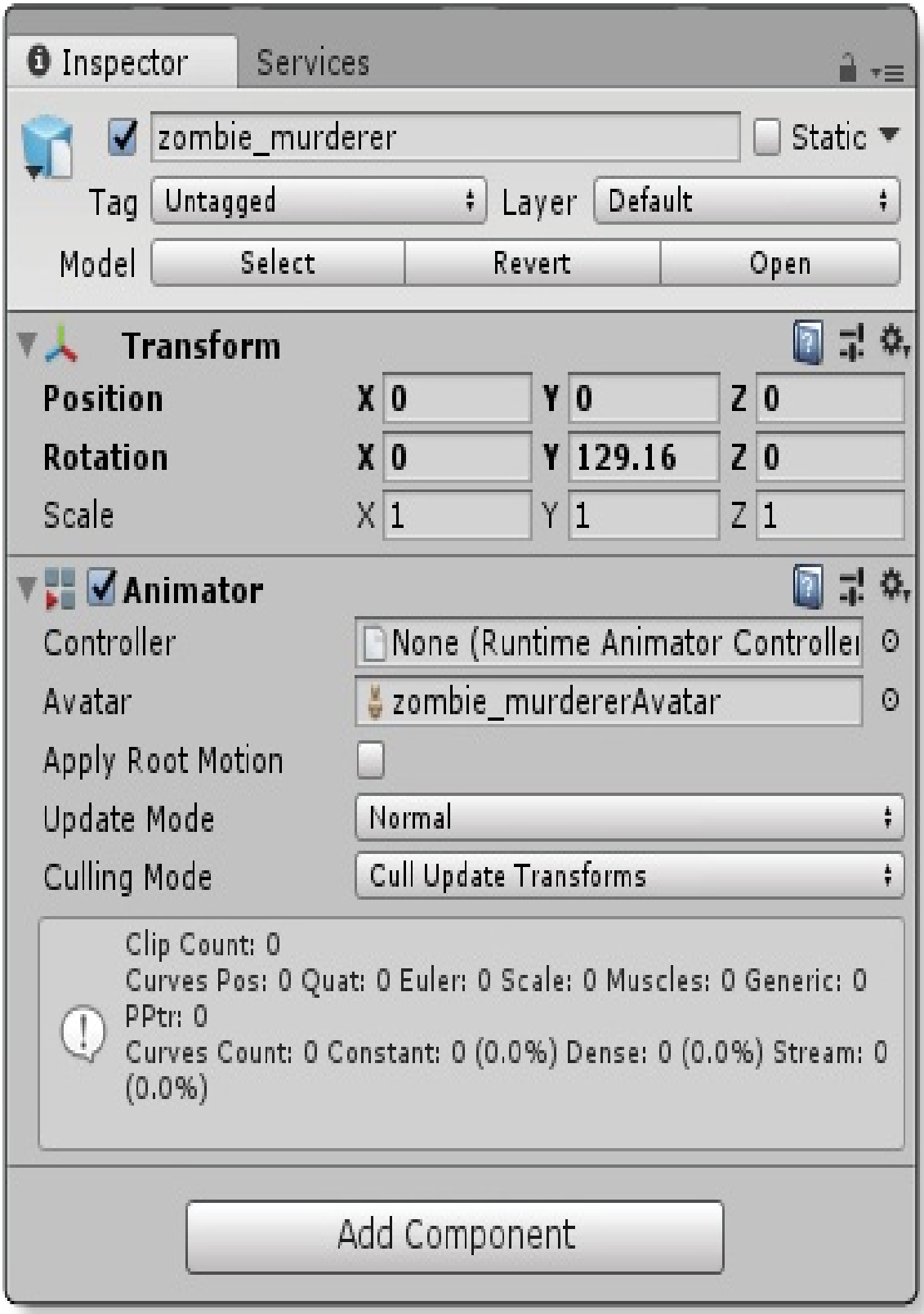
Asigne el nuevo material a todos los elementos del modelo.

Modelo texturizado

Rótelo en la escena con la herramienta de rotación.



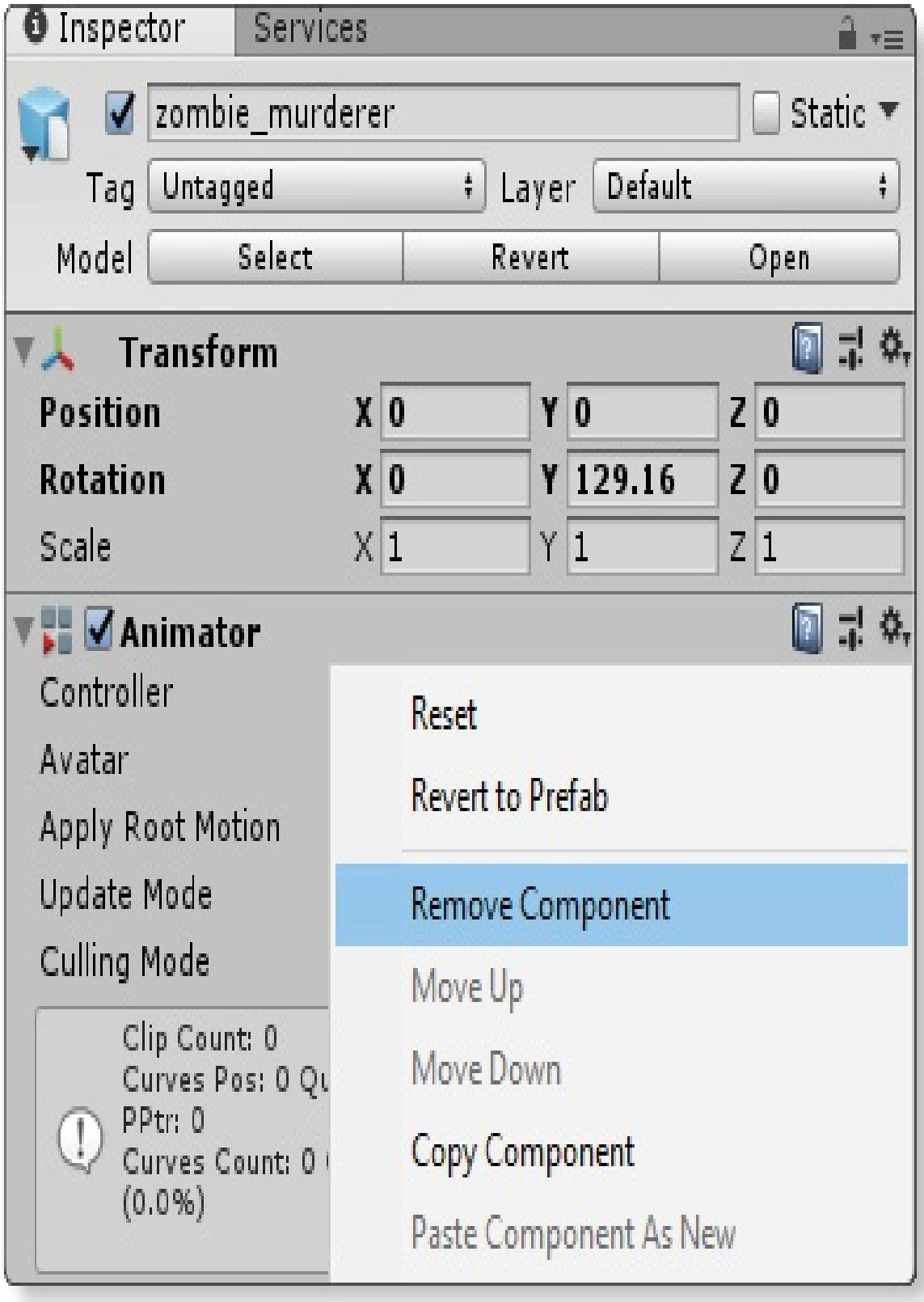
Rotación del modelo


Inspector
Services
zombie_murderer
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 129.16
Z 0
Scale
X 1
Y 1
Z 1
Animator
Controller
None (Runtime Animator Controller
Avatar
zombie_murdererAvatar
Apply Root Motion
Update Mode
Normal
Culling Mode
Cull Update Transforms
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0 (0.0%)
Add Component

Componentes del modelo

Es importante rotar en un solo eje para evitar problemas posteriores con las rotaciones, puede rotar con la herramienta de rotación o puede colocar el puntero en la letra “Y” de rotación en “Transform” y arrastrar para rotar únicamente en dicho eje.

Remueva el componente “Animator” de “Inspector”.


Inspector
Services
zombie_murderer
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 129.16
Z 0
Scale
X 1
Y 1
Z 1
Animator
Controller
Avatar
Apply Root Motion
Update Mode
Culling Mode
Clip Count: 0
Curves Pos: 0 Qu
PPtr: 0
Curves Count: 0
(0.0%)
Reset
Revert to Prefab
Remove Component
Move Up
Move Down
Copy Component
Paste Component As New

Remover componente Animator

NOTA:

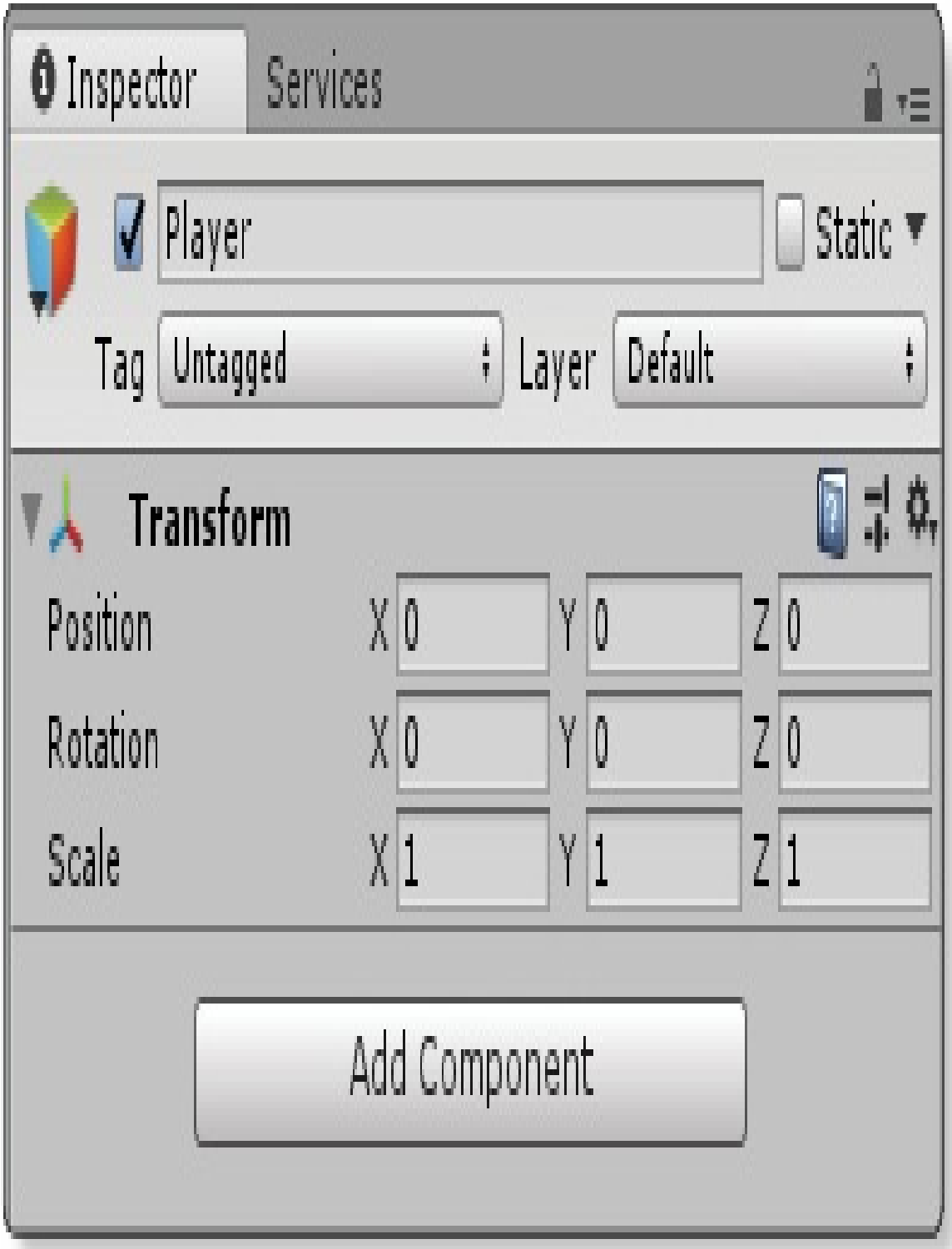
El componente “Animator” nos servirá para aplicar el controlador de Animación para nuestro modelo, pero vamos a crear un EmptyObject padre para el modelo y ahí es donde vamos a reasignar el componente “Animator”.

Haga un nuevo “EmptyObject”, nómbrelo “Player”, reset en sus transformaciones, coloque el objeto “Player” en la base del modelo “zombie_murderer” y hágalo padre del modelo.

Hierarchy
Create
All
202
Main Camera
Directional Light
Platform
Player
zombie_murderer

Emparentamiento

Posición correcta del objeto Player


Inspector
Services
Player
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Add Component

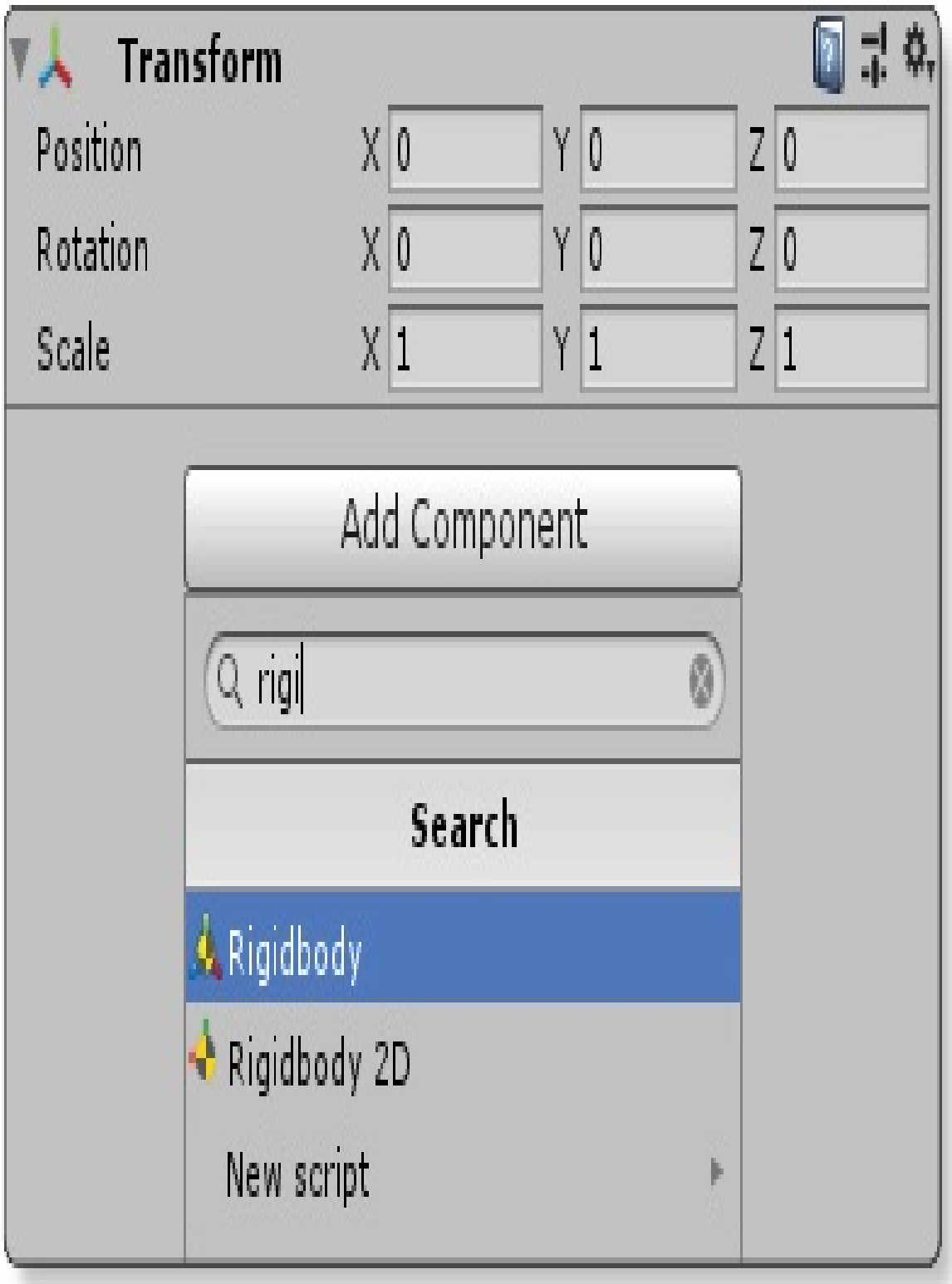
Transformaciones en cero

NOTA:

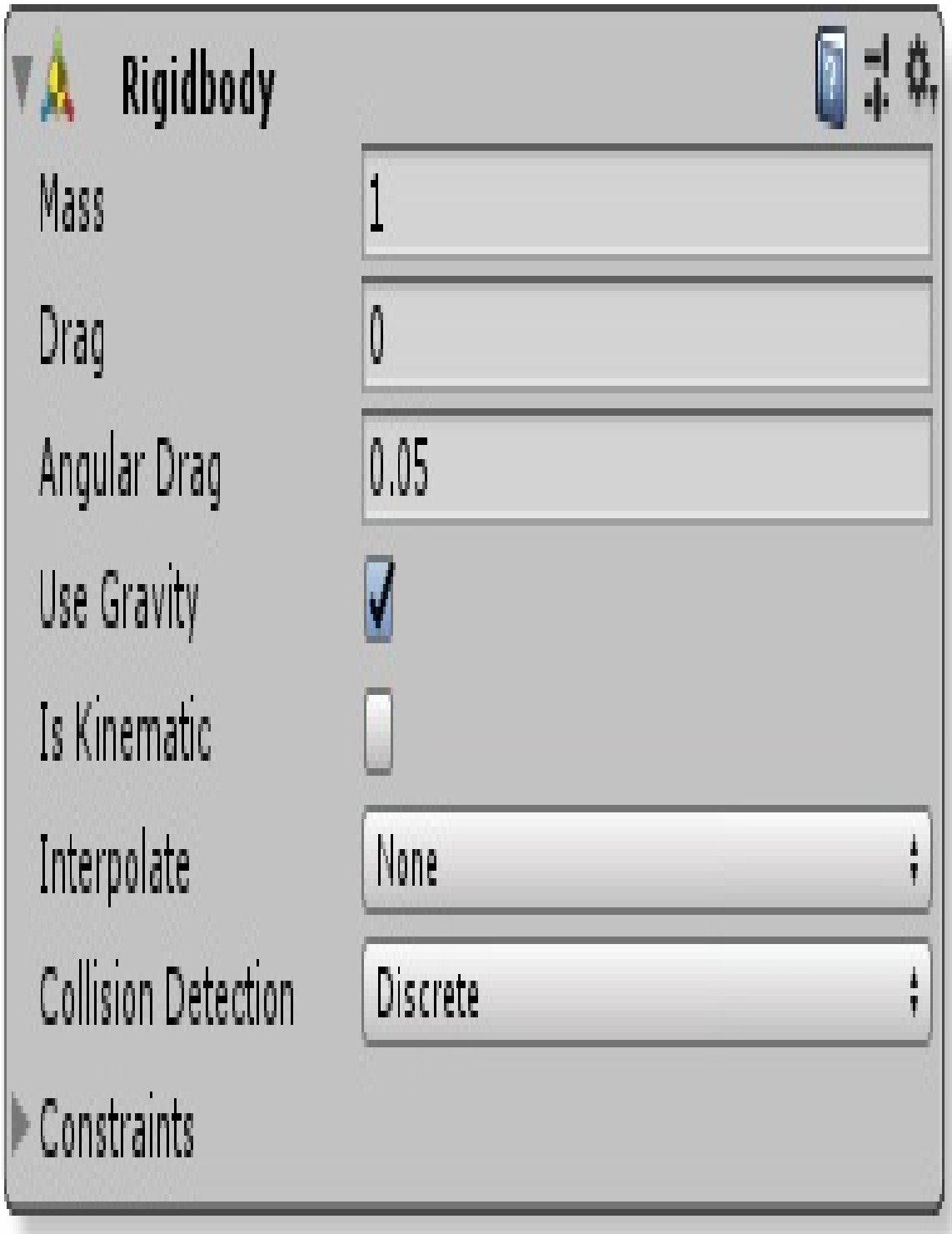
Es de suma importancia que las transformaciones de rotación del nuevo objeto “Player” se encuentren en 0,0,0 independientemente de las transformaciones de rotación “zombie_murderer”.

Recuerda que para emparentar un elemento a otro basta con arrastrarlo encima de él en la ventana de “Hierarchy”.

Asigne al objeto “Player” un componente “Rigid Body” con los valores mostrados.


Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Add Component
rigi
Search
Rigidbody
Rigidbody 2D
New script

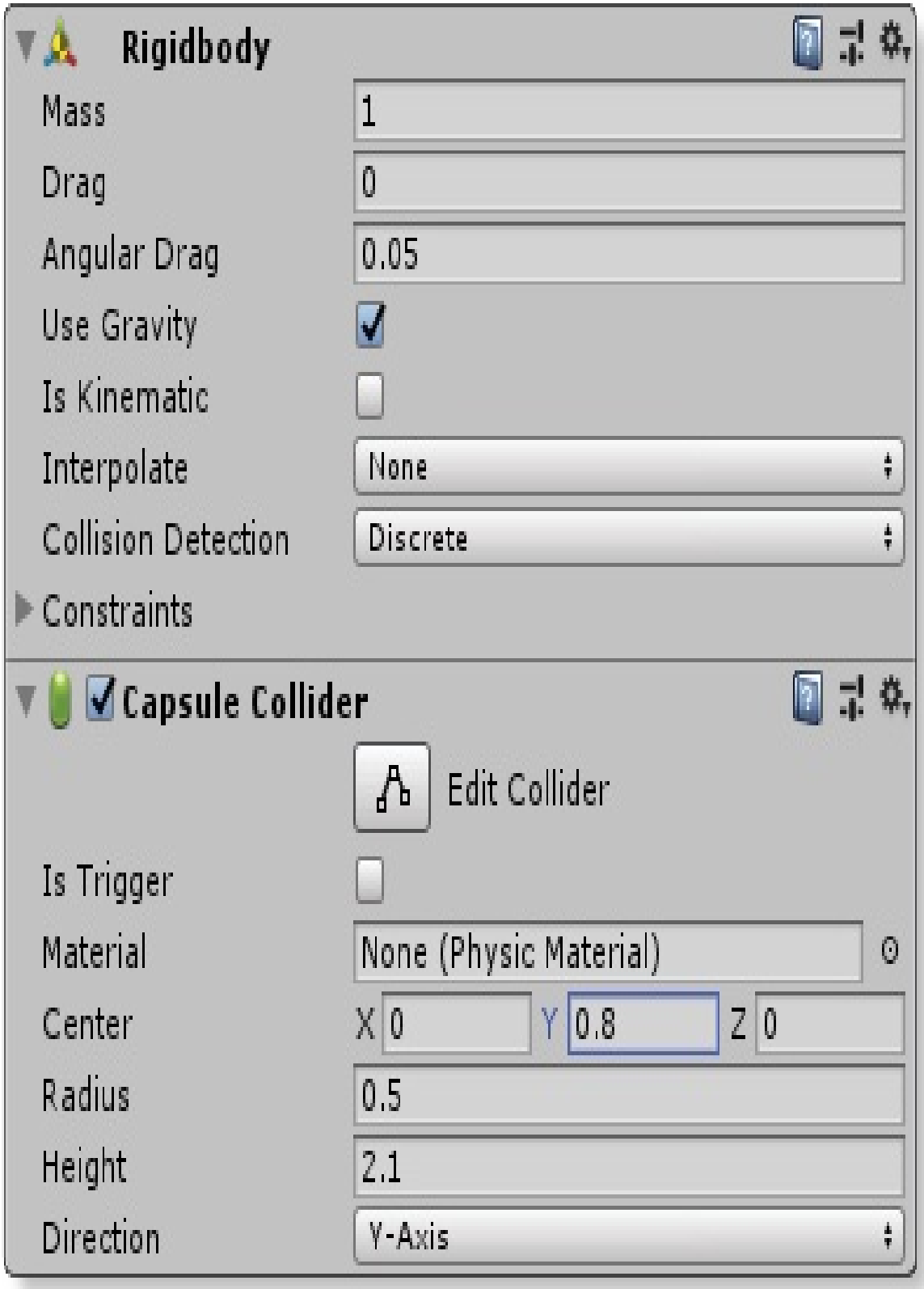
Componente Rigid Body


Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints

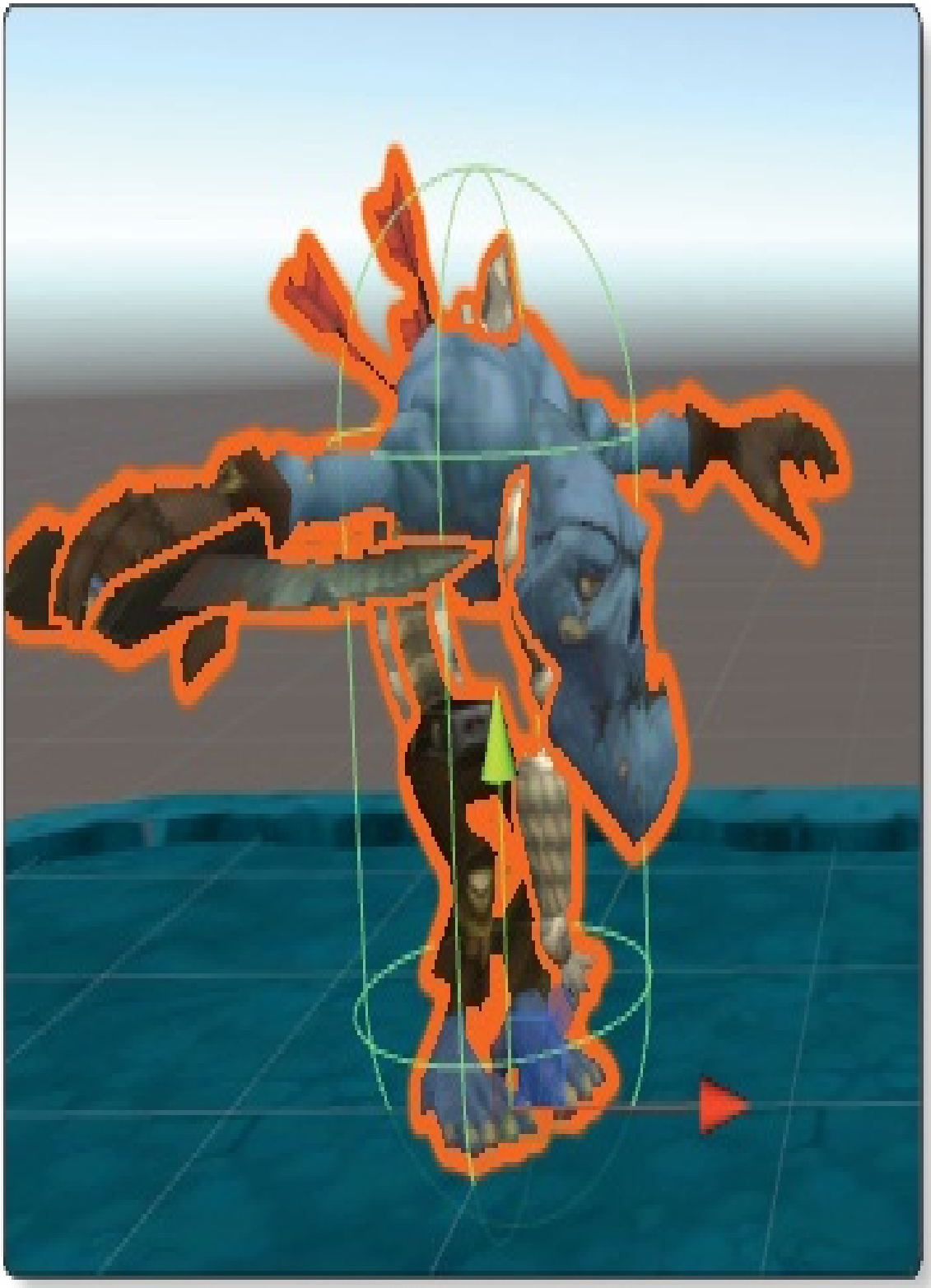
Valores del componente Rigid Body

El componente “Rigid Body” nos permite asignarle propiedades físicas al modelo, tales como el uso de la gravedad, detección de colisiones, masa y arrastre.

Asigne al objeto “Player” un componente “CapsuleCollider” con los valores mostrados.


Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Capsule Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 0.8
Z 0
Radius
0.5
Height
2.1
Direction
Y-Axis

Componente Capsule Collider



Player con Capsule Collider

Previsualice el juego para observar que el “Player” se mantiene en la plataforma

Para previsualizar el juego tiene que oprimir el botón “Play” en la parte superior de la ventana “Scene”.

Play

Controles de reproducción del juego

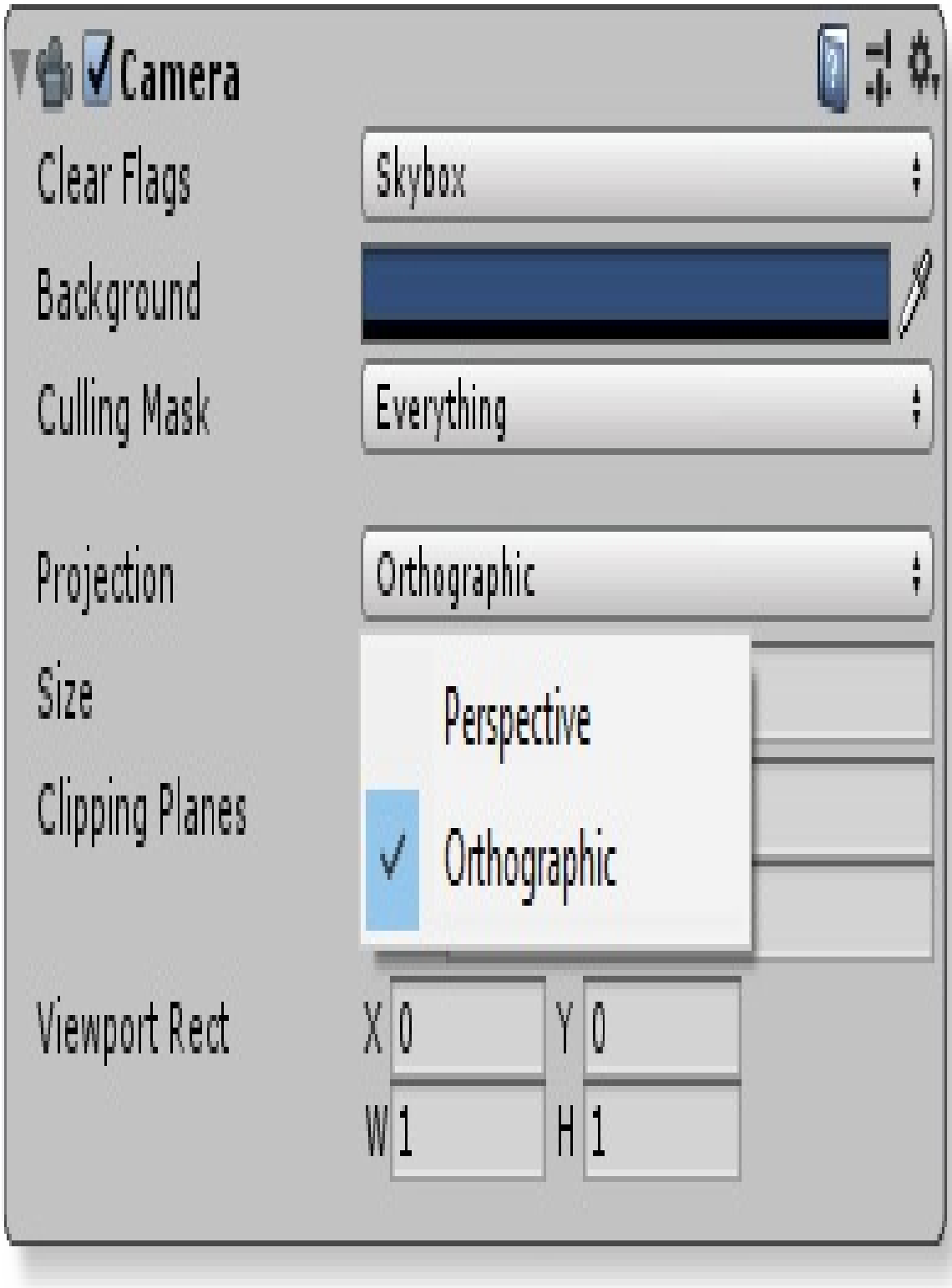
NOTA:

Al momento de previsualizar puede experimentar resultados impredecibles y en esta fase es donde podemos editar algunos valores para tener los resultados deseados.

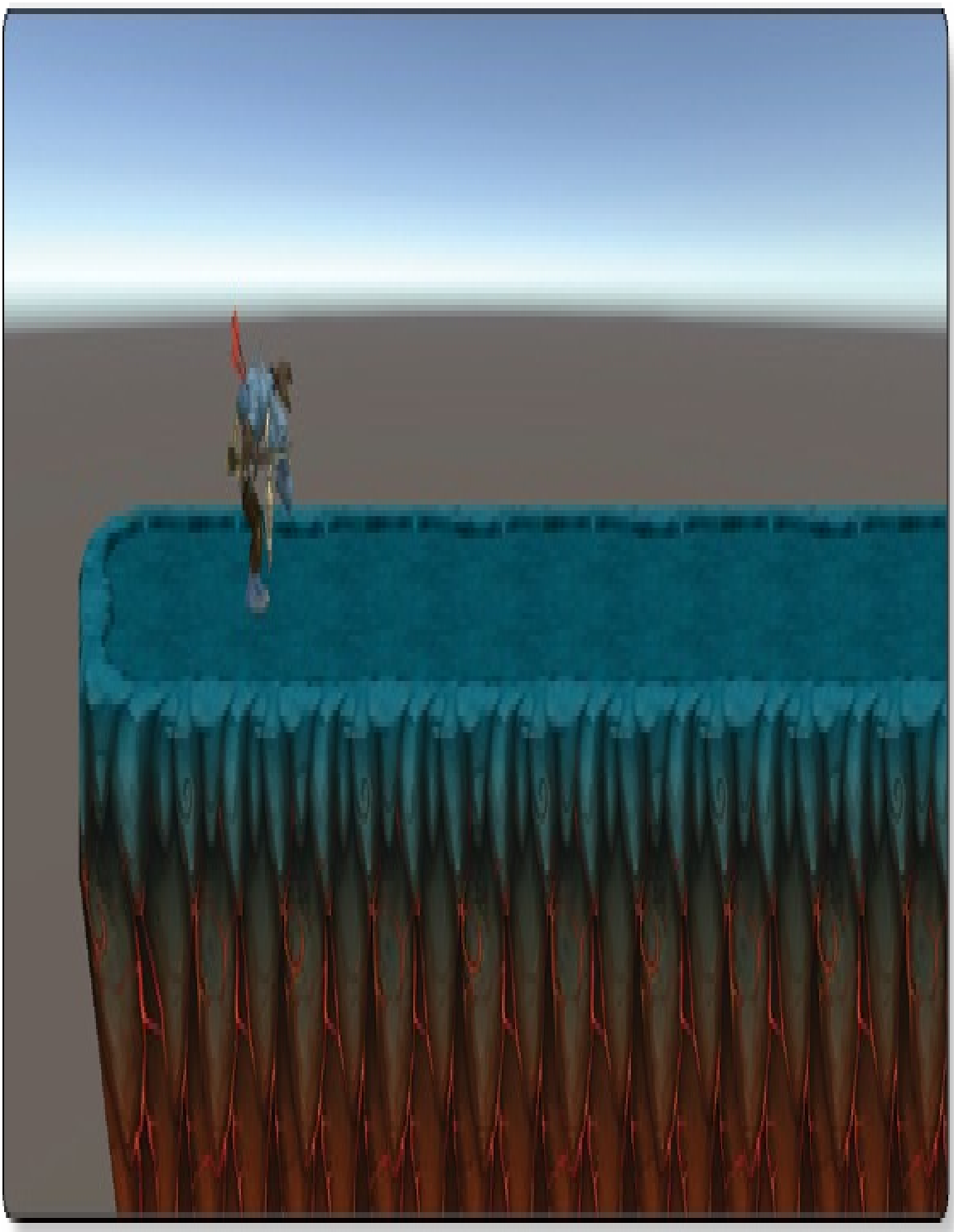
Si el personaje parece que se encuentra flotando, puede mover hacia arriba un poco el “CapsulleCollider” desde sus valores en Inspector.

Si el personaje se encuentra volteando más hacia la cámara en lugar de voltear hacia dónde va a caminar, puede cambiar sus valores de rotación en “Y” en las transformaciones del objeto “zombie_murderer” (OJO… No del objeto “Player”).

En función a este ejercicio voy a utilizar el tipo de cámara ortográfica misma que se podrá cambiar de tipo “Perspectiva” a “ortográfica” seleccionando desde la ventana de Hierarchy el objeto “Main Camera” y cambiando sus valores de “Projection” en la ventana de “Inspector”.


Camera
Clear Flags
Skybox
Background
Culling Mask
Everything
Projection
Orthographic
Size
Clipping Planes
Perspective
Orthographic
Viewport Rect
X 0
Y 0
W 1
H 1

Valores de Main Camera



Vista del juego en reproducción

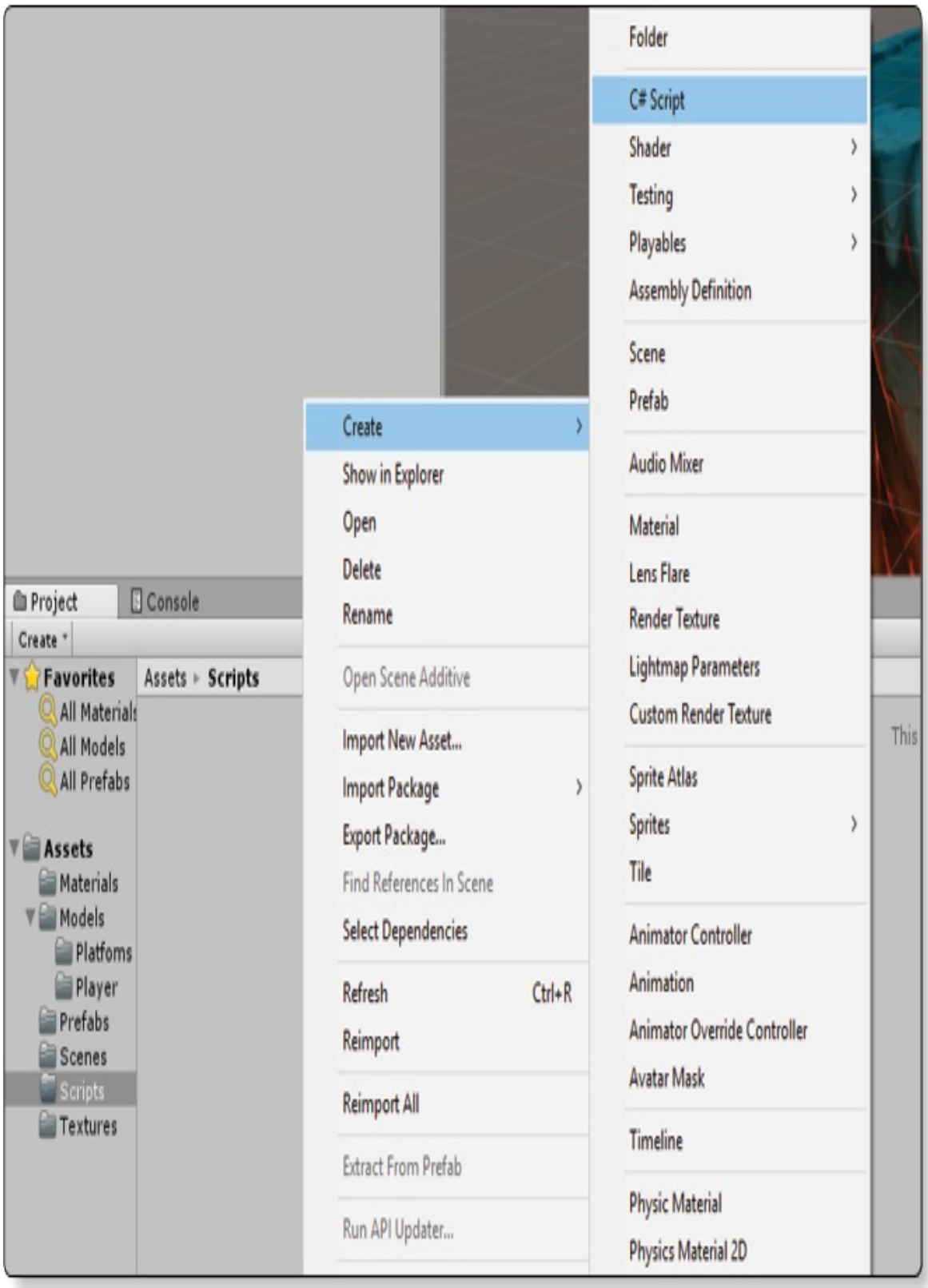
PROGRAMACIÓN DEL MOVIMIENTO HORIZONTAL

En este apartado nos introduciremos a elementos de la programación con “Scripting” en el lenguaje C#.

Hablaremos acerca de la estructura del script, trabajo con variables, el uso y creación de las funciones y escribiremos nuestro primer script para programar el movimiento horizontal de izquierda a derecha de nuestro Player.

En la ventana de “Project” haga un nuevo folder y nómbrelo “Scripts”.

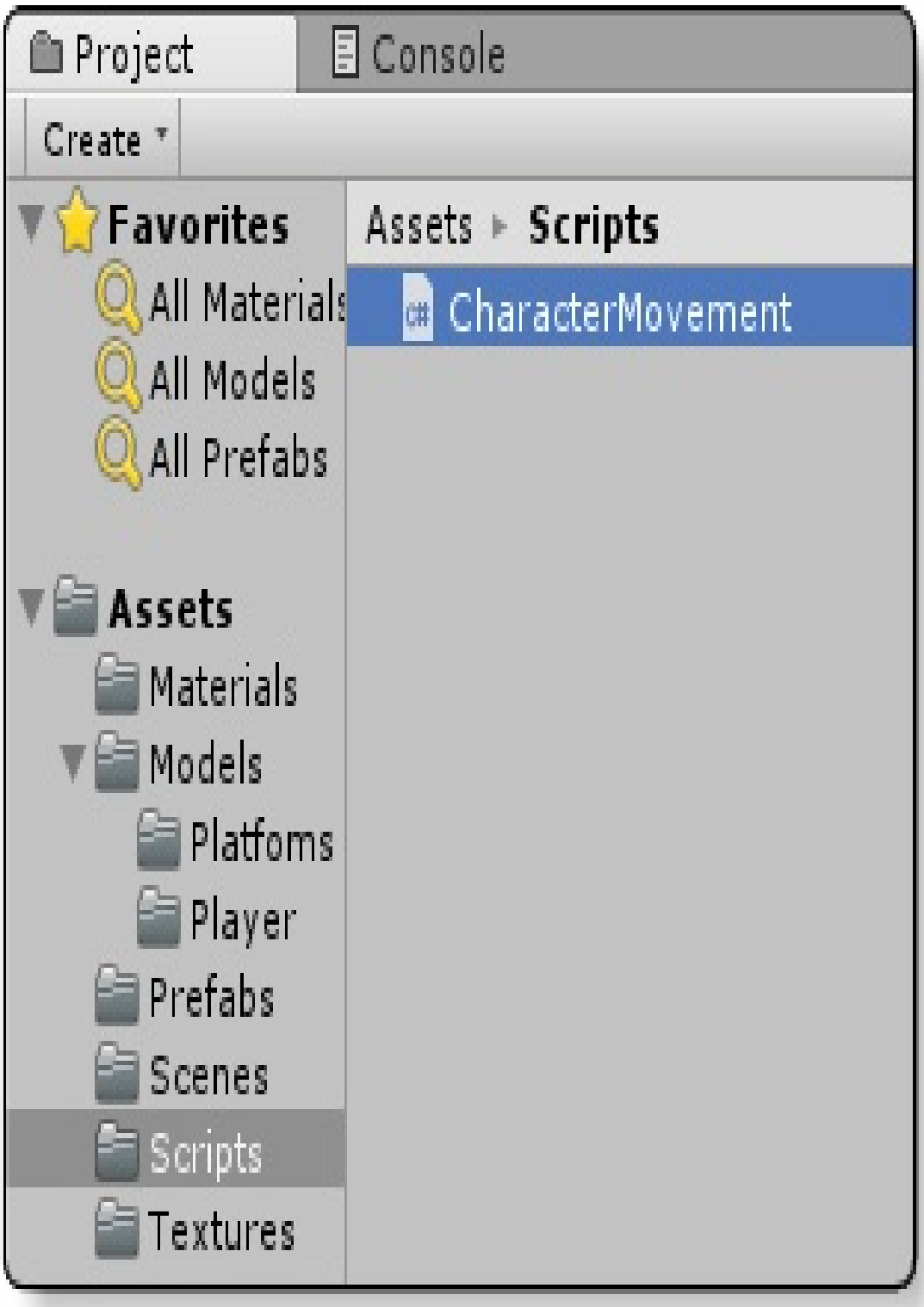
Clic derecho y asigne los valores mostrados para hacer un nuevo script y nómbrelo “CharacterMovement”.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Platforms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Scripts
Create
Show in Explorer
Open
Delete
Rename
Open Scene Additive
Import New Asset...
Import Package
Export Package...
Find References In Scene
Select Dependencies
Refresh Ctrl+R
Reimport
Reimport All
Extract From Prefab
Run API Updater...
Folder
C# Script
Shader
Testing
Playables
Assembly Definition
Scene
Prefab
Audio Mixer
Material
Lens Flare
Render Texture
Lightmap Parameters
Custom Render Texture
Sprite Atlas
Sprites
Tile
Animator Controller
Animation
Animator Override Controller
Avatar Mask
Timeline
Physic Material
Physics Material 2D
This

Creación de un Script

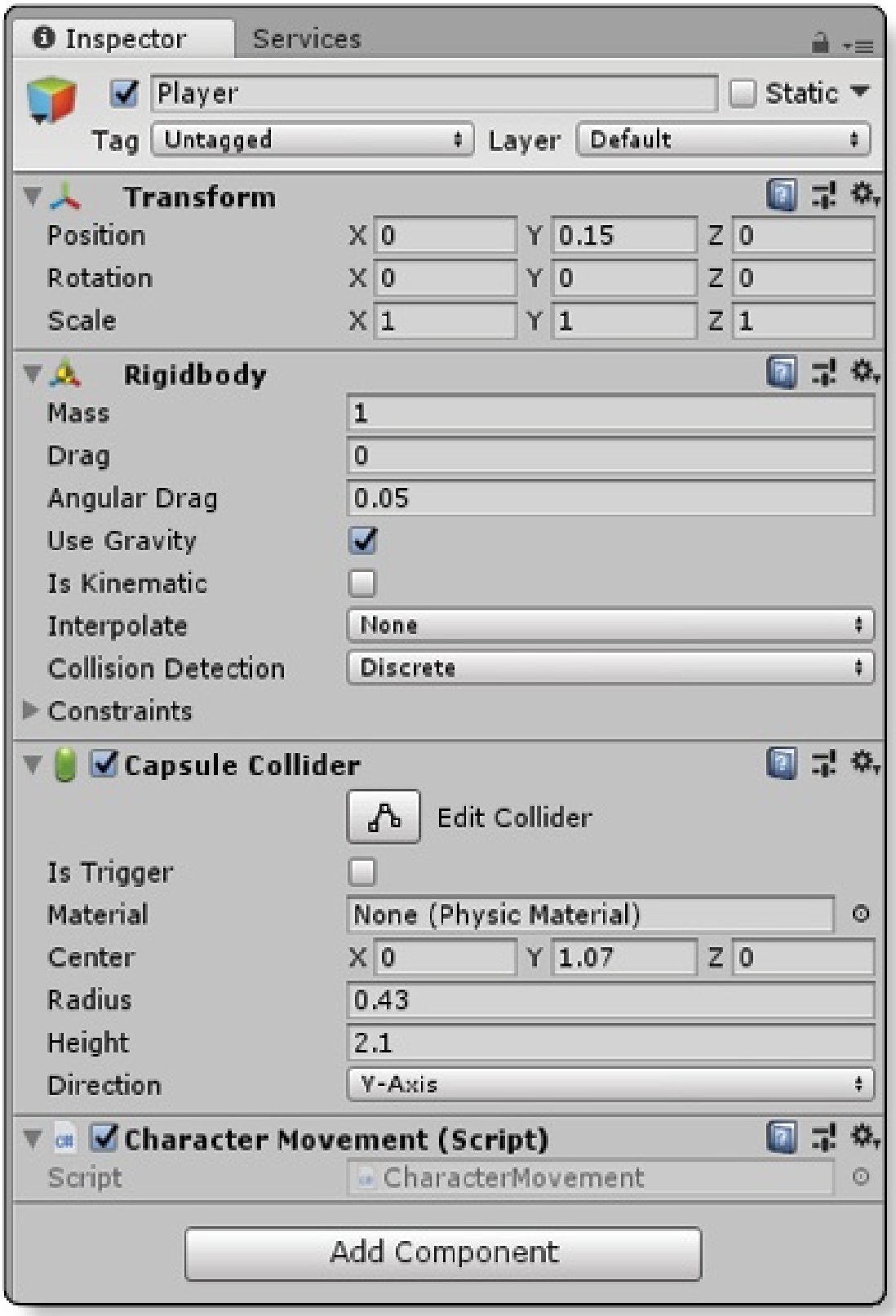
NOTA:

Es importante tomar en cuenta que cada vez que creamos un nuevo script, debemos de asignarle inmediatamente su nombre, ya que al crear el nombre del script también se crea su nombre de clase.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Platforms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Scripts
CharacterMovement

Script creado

Asigne el script creado al objeto “Player”.


Inspector
Services
Player
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 0.15
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Capsule Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 1.07
Z 0
Radius
0.43
Height
2.1
Direction
Y-Axis
Character Movement (Script)
Script
CharacterMovement
Add Component

Script asignado al Player

Puede asignar un script a un objeto arrastrando el script encima del objeto en la ventana Hierarchy.

Doble clic en el script creado para abrirlo en su editor de código.

Estructura básica de un C# Script

CharacterMovement.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class CharacterMovement : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9
10      }
11
12      // Update is called once per frame
13      void Update () {
14
15      }
16  }
17
```

Librerías de Unity

Al abrir por primera vez un script nuevo encontraremos la nomenclatura “using” y por defecto son las librerías de C# que requiere mandar llamar el script para que pueda ser ejecutado dentro de “Unity”.

CharacterMovement.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class CharacterMovement : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9
10      }
11
12      // Update is called once per frame
13      void Update () {
14
15      }
16  }
17
```

Clase del script

La clase debe de ser “public” (pública) para poder utilizar elementos de otras clases o bien que otra clase utilice los de la clase que estamos programando.

El nombre de la clase (en este caso “CharacterMovement”) debe de ser exactamente igual al nombre del script creado en Unity para que pueda funcionar correctamente, de lo contrario obtendremos un error de compilación.

MonoBehaviour es la clase base que debe ser utilizada dentro del lenguaje C# cuando se crea un script para Unity.

CharacterMovement.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class CharacterMovement : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9
10      }
11
12      // Update is called once per frame
13      void Update () {
14
15      }
16  }
17
```

Uso de // para comentar

Los textos que son precedidos por (//) son ignorados completamente por el compilador y contienen instrucciones o notas para el programador.

CharacterMovement.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CharacterMovement : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}

```

Función Start

Los métodos o también llamados funciones son encapsulación de diversas instrucciones, a su vez también existen métodos con funciones preestablecidas dentro de las bibliotecas de C#.

Uno de estos métodos con funciones preestablecidas es la función “Start” y todo el código que vayamos a escribir dentro de esta función va a correr inmediatamente cuando previsualicemos el juego con “Play”.

CharacterMovement.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class CharacterMovement : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9
10      }
11
12      // Update is called once per frame
13      void Update () {
14
15      }
16  }
17
```

Función Update

La función “Update” nos sirve para que el código escrito dentro de ella corra a cada cuadro de reproducción de nuestro juego y su velocidad depende del número de cuadros por segundo a los que corra el juego.

Explicación de variables

Una variable es un actor de nuestro script, es decir para que pueda existir un elemento dentro de nuestro script lo debemos de almacenar en una variable.

Los elementos que integran una variable varían por su privacidad, tipo, nombre y valor

Por su privacidad las variables pueden ser:

Públicas (public): Por lo cual pueden ser utilizadas por otro script y las podemos visualizar en Unity.

Privadas (private): Que pueden ser utilizadas únicamente en el script donde fueron creadas y no pueden ser visualizadas en Unity.

Por su tipo pueden ser:

Numéricas: Mismas que pueden ser clasificadas como:

Enteras (int): Con valores enteros, sin decimales

Flotantes (float): Con valores decimales

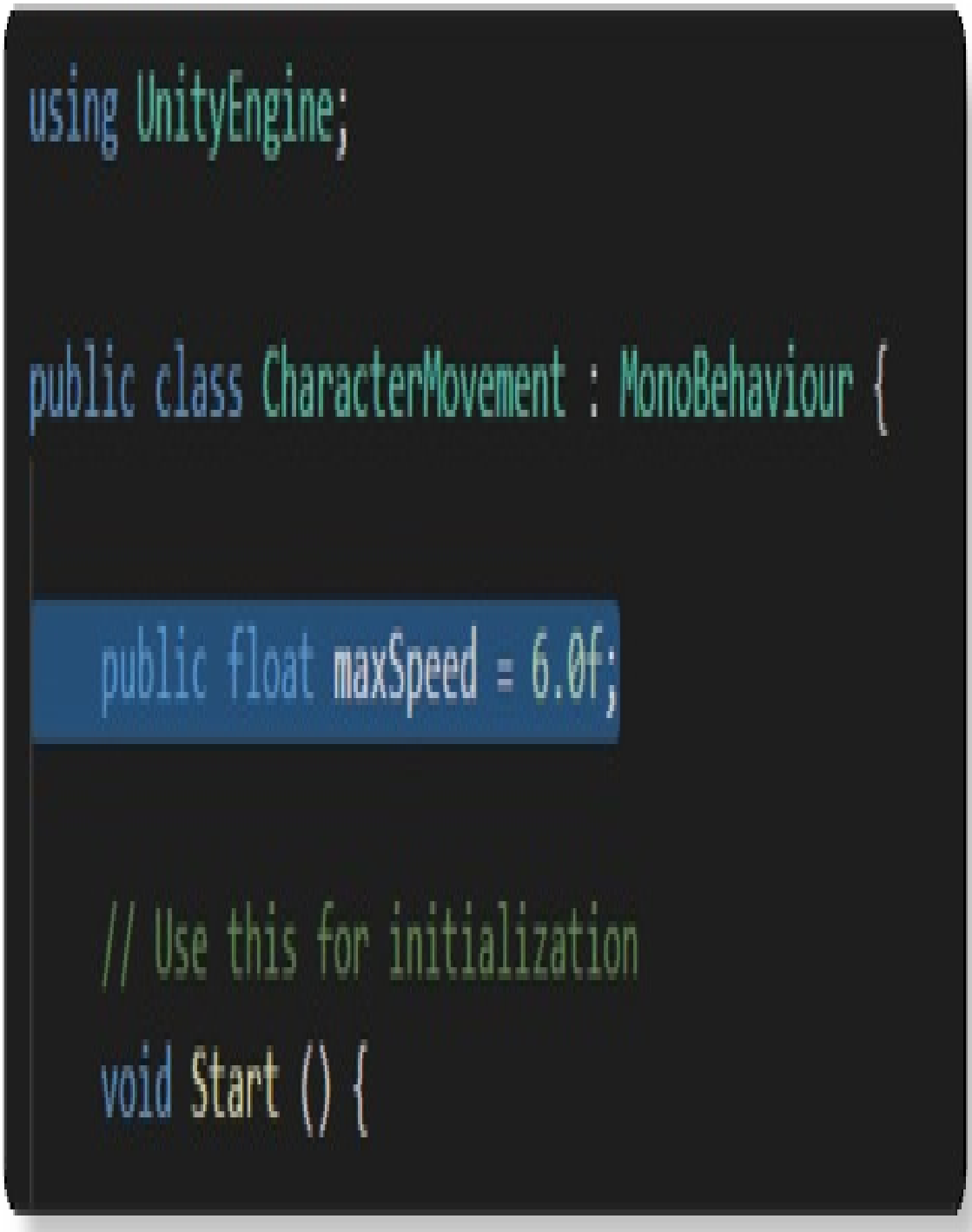
Lógicas (bool): Las cuales arrojan resultados verdaderos (true) o falsos (false)

De objeto:Las cuales se declaran con el nombre del tipo de objeto específico que se va a llamar

Por su nombre: Se le puede asignar el nombre que guste siempre y cuando tenga relación con los datos que se van a almacenar.

Por su valor: Al declarar o crear la variable se le puede, o no asignar un valor inicial.

Declare la siguiente variable en el script.


using UnityEngine;
public class CharacterMovement : MonoBehaviour {
public float maxSpeed = 6.0f;
// Use this for initialization
void Start () {

Variable tipo float

En este apartado creamos la variable de nombre “maxSpeed” con los siguientes valores:

La variable es publica, de tipo float llamada maxSpeed y con un valor inicial de 6.0.

NOTA:

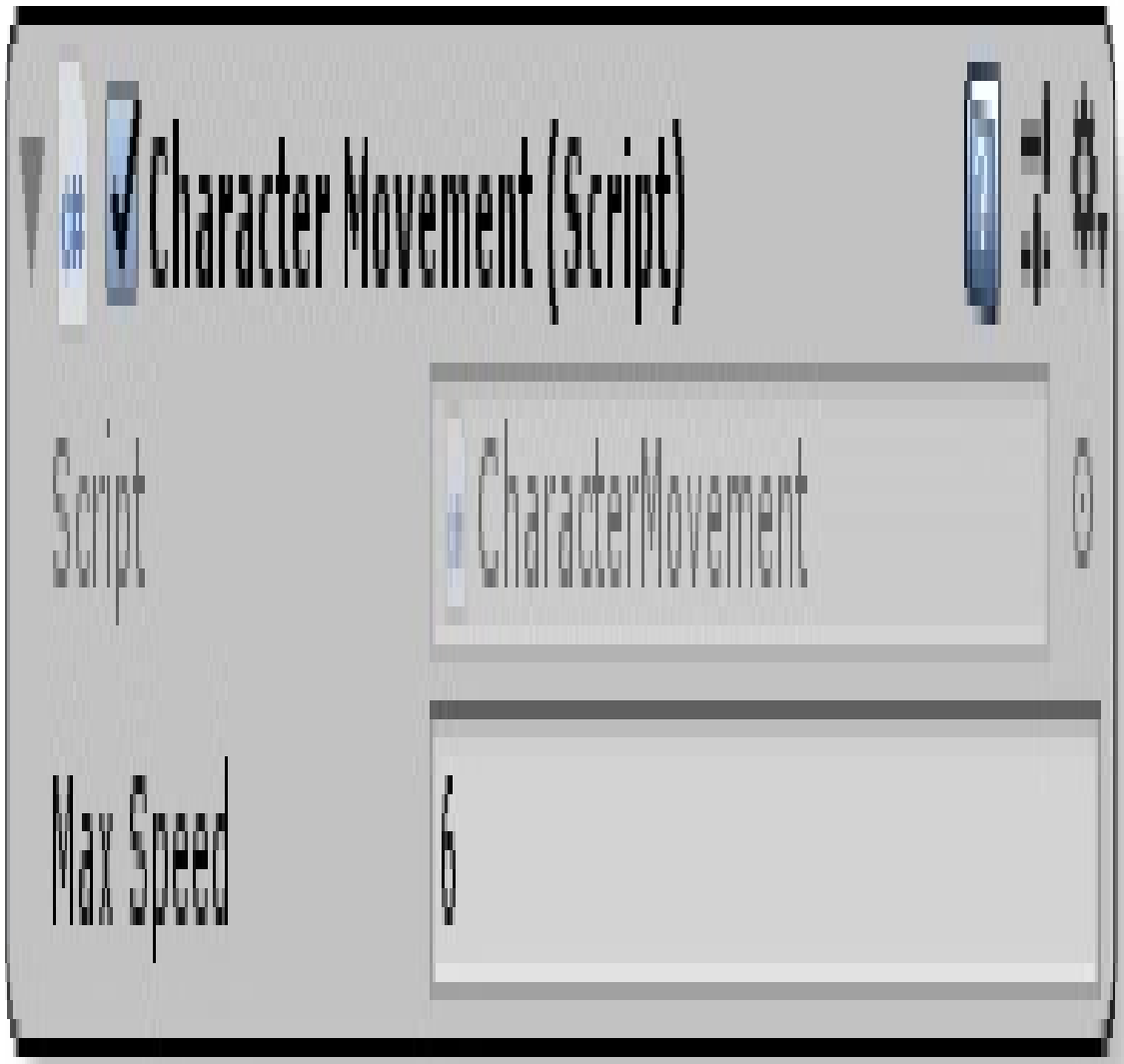
Cuando asignamos un valor a una variable tipo “float” debemos de asignar la “f” al final del valor.

NOTA:

Después de declarar una variable debemos de finalizar con (;)

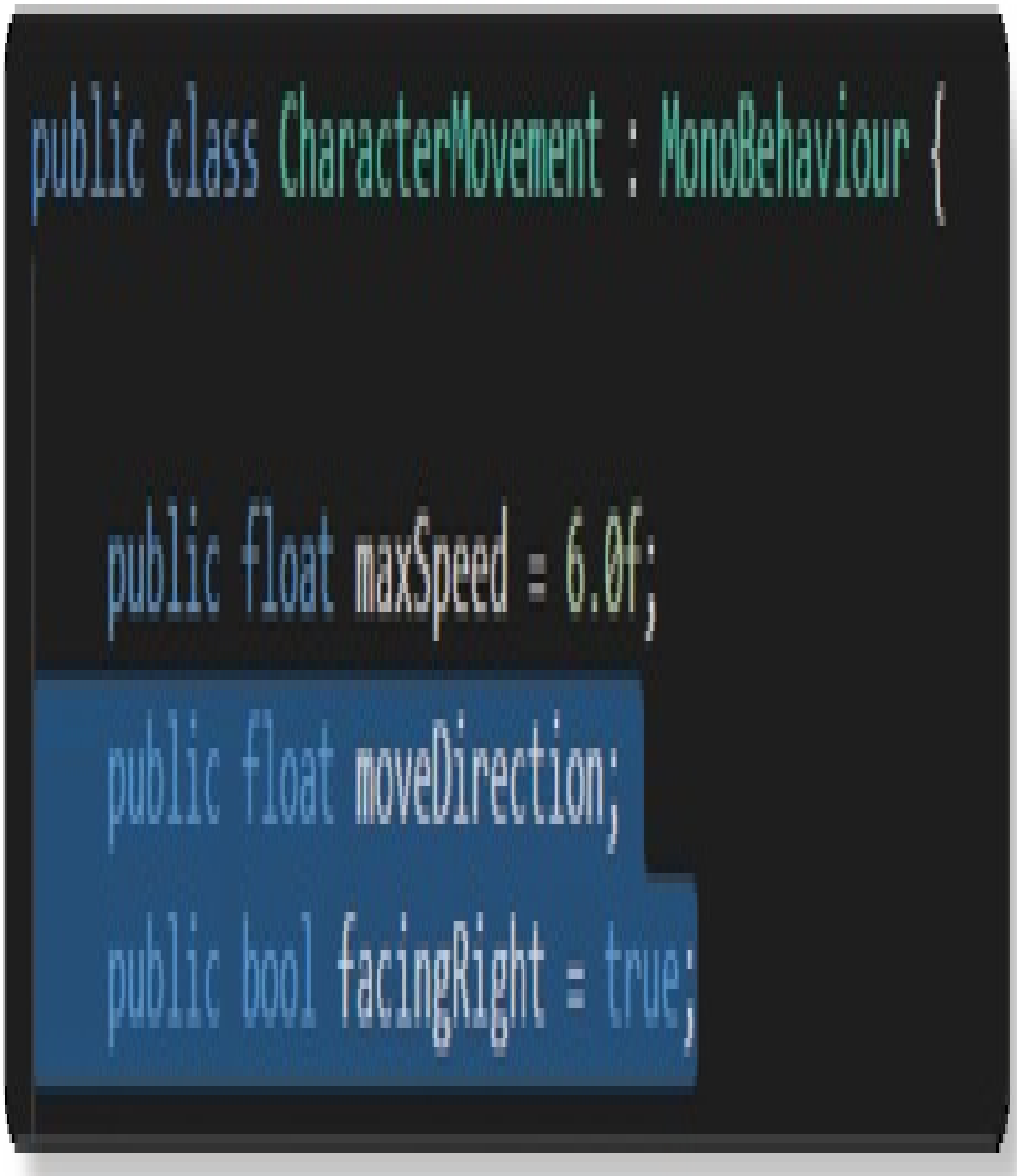
Salve el script (Control + s), vaya a Unity

Seleccione el objeto Player y observe en Inspector la creación de la variable publica


Character Movement (Script)
Script
CharacterMovement
Max Speed
6

Variable publica visualizada en Unity

Declare las siguientes variables en el script.


public class CharacterMovement : MonoBehaviour {
public float maxSpeed = 6.0f;
public float moveDirection;
public bool facingRight = true;

Variables float y bool

La variable tipo float “moveDirection” no tiene todavía valor, pero ya existe.

La variable tipo “bool” llamada “facingRight” le asignamos un valor verdadero.

Character Movement (Script)
Script
CharacterMovement
Max Speed
6
Move Direction
0
Facing Right

Declare la siguiente variable en el script.

public class CharacterMovement : MonoBehaviour {
public float maxSpeed = 6.0f;
public float moveDirection;
public bool facingRight = true;
private Rigidbody rigidbody;

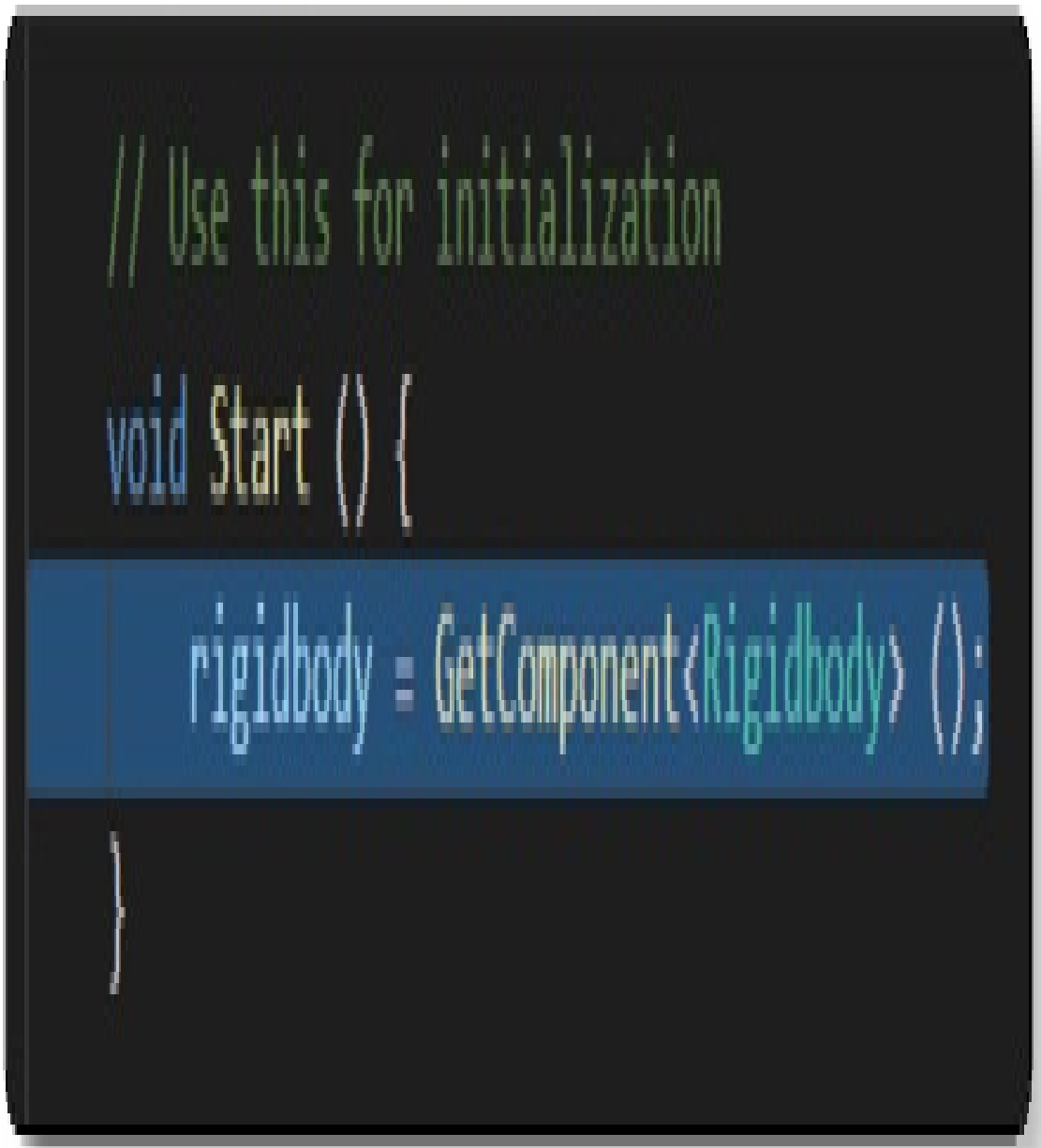
Variable de tipo objeto

Esta es una variable tipo objeto, en este caso específico necesitamos mandar llamar el componente “RigidBody” de nuestro Player y por convención, generalmente nombramos a la variable de la misma forma que su tipo, pero con la primera letra minúscula.

Inicialización de variables

Las variables de tipo objeto deben de ser inicializadas en la función “Start” para mandar llamar debidamente su componente.

Escriba el siguiente código en la función “Start” para inicializar la variable.


// Use this for initialization
void Start () {
rigidbody = GetComponent<Rigidbody> ();
}

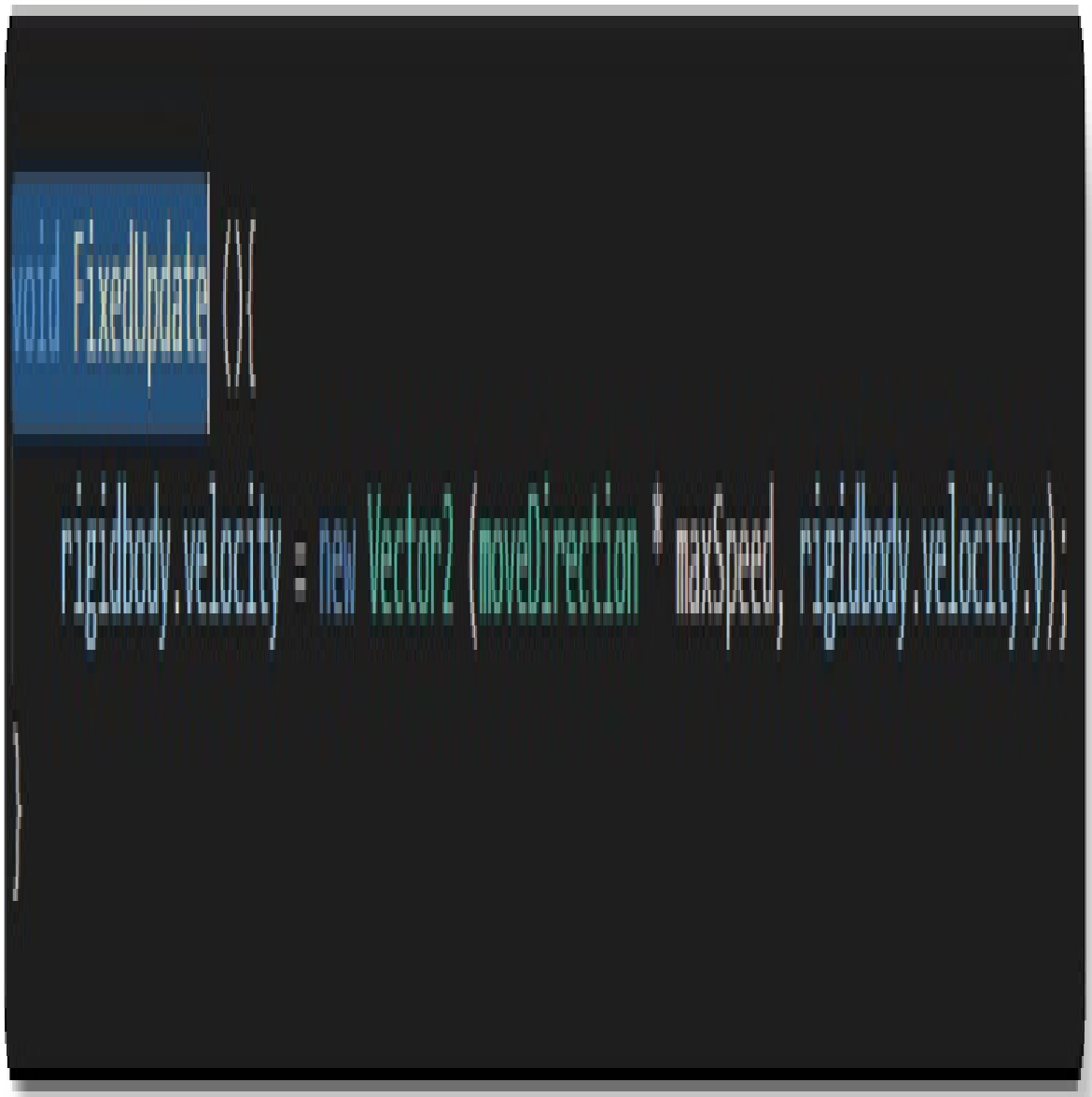
Inicialización de la variable tipo objeto

Escriba la siguiente función “FixedUpdate” para hacer que el personaje se mueva.

```
    // Use this for initialization
    void Start () {
        rigidbody = GetComponent<Rigidbody> ();
    }

    // Update is called once per frame
    void Update () {

    }

    void FixedUpdate (){
        rigidbody.velocity = new Vector2 (moveDirection * maxSpeed, rigidbody.velocity.y);
    }
}
```

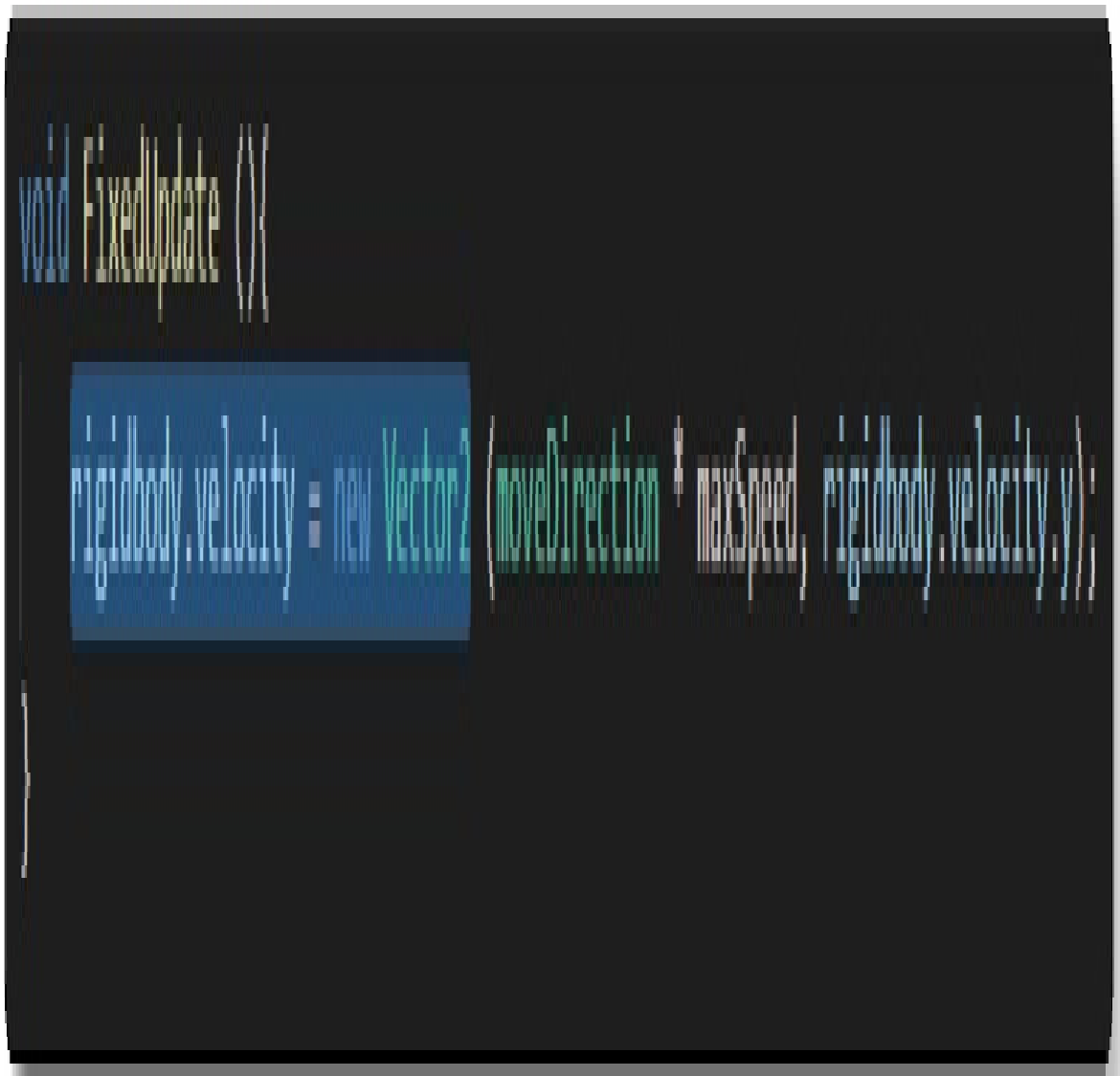
Función para mover el personaje

Se crea una nueva función llamada “FixedUpdate”, la cual es parte de las librerías preestablecidas de C# la cual, y al igual que la función “Update” se va repitiendo constantemente a cada cuadro.


void FixedUpdate()
rigidbody.velocity = new Vector2(moveDirection * maxSpeed, rigidbody.velocity.y);
}

Función FixedUpdate

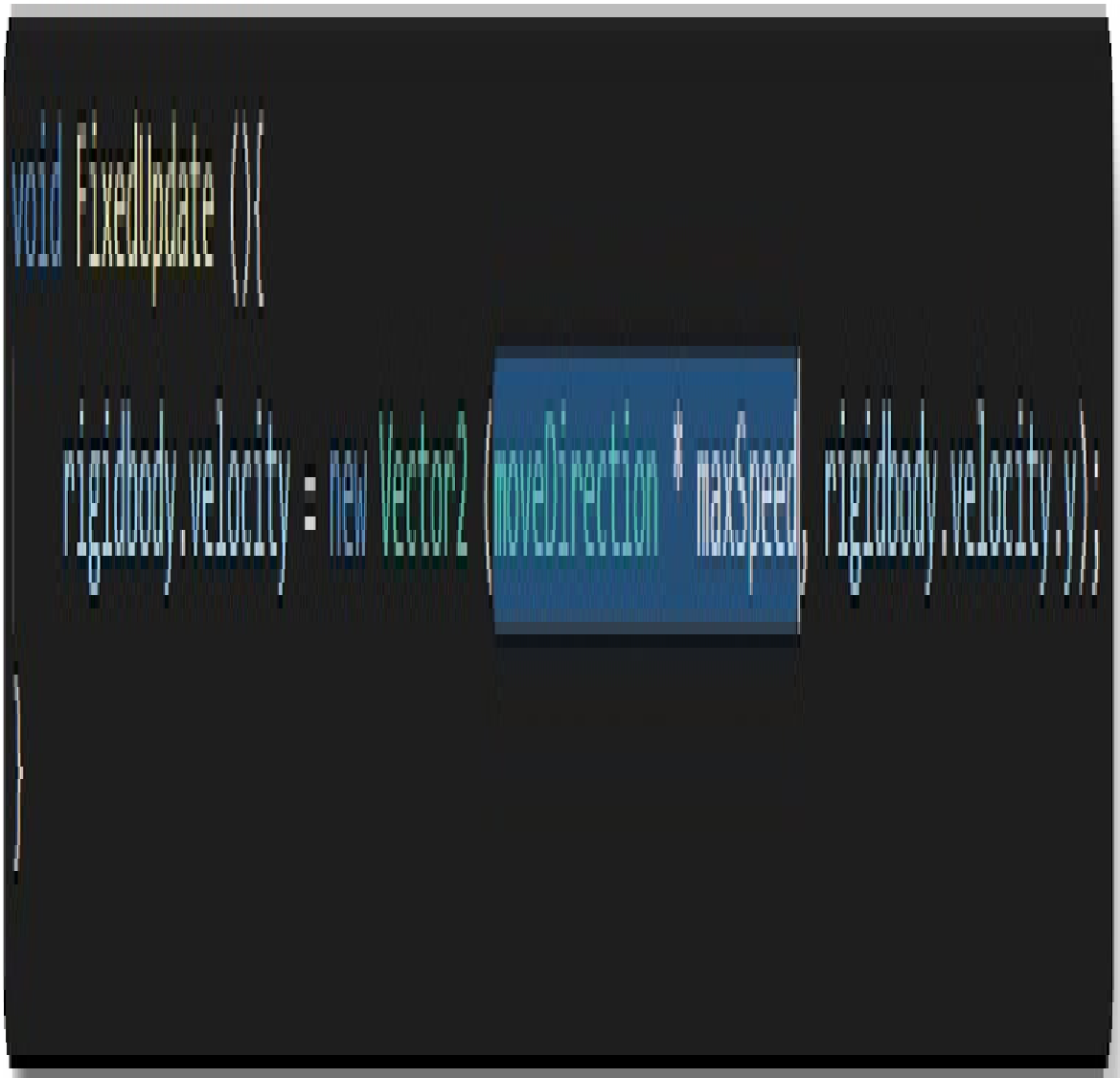
La función “FixedUpdate” en diferencia a la función “Udate” la utilizamos cuando está afectando a un objeto con componentes físicos, es decir, con “RigidBody”, ya que la manera que se tienen de procesar los fenómenos físicos en cuanto a cuadros se refiere, es diferente.


void FixedUpdate ()
{
rigidbody.velocity = new Vector2 (moveDirection * maxSpeed, rigidbody.velocity.y);
}

Asignación de nuevo Vector 2

Seguidamente mandamos llamar la velocidad de la variable “rigidbody” para asignarle un valor. Dicho valor lo establecemos como un nuevo vector.

Tomando en cuenta que vamos a trabajar en este proyecto solo con 2 vectores (x , y), por ello le asignamos el número 2 y los valores serán separados por una coma (,).


void FixedUpdate ()
{
rigidbody.velocity = new Vector2 (moveDirection * maxSpeed, rigidbody.velocity.y);
}

Valor del vector X

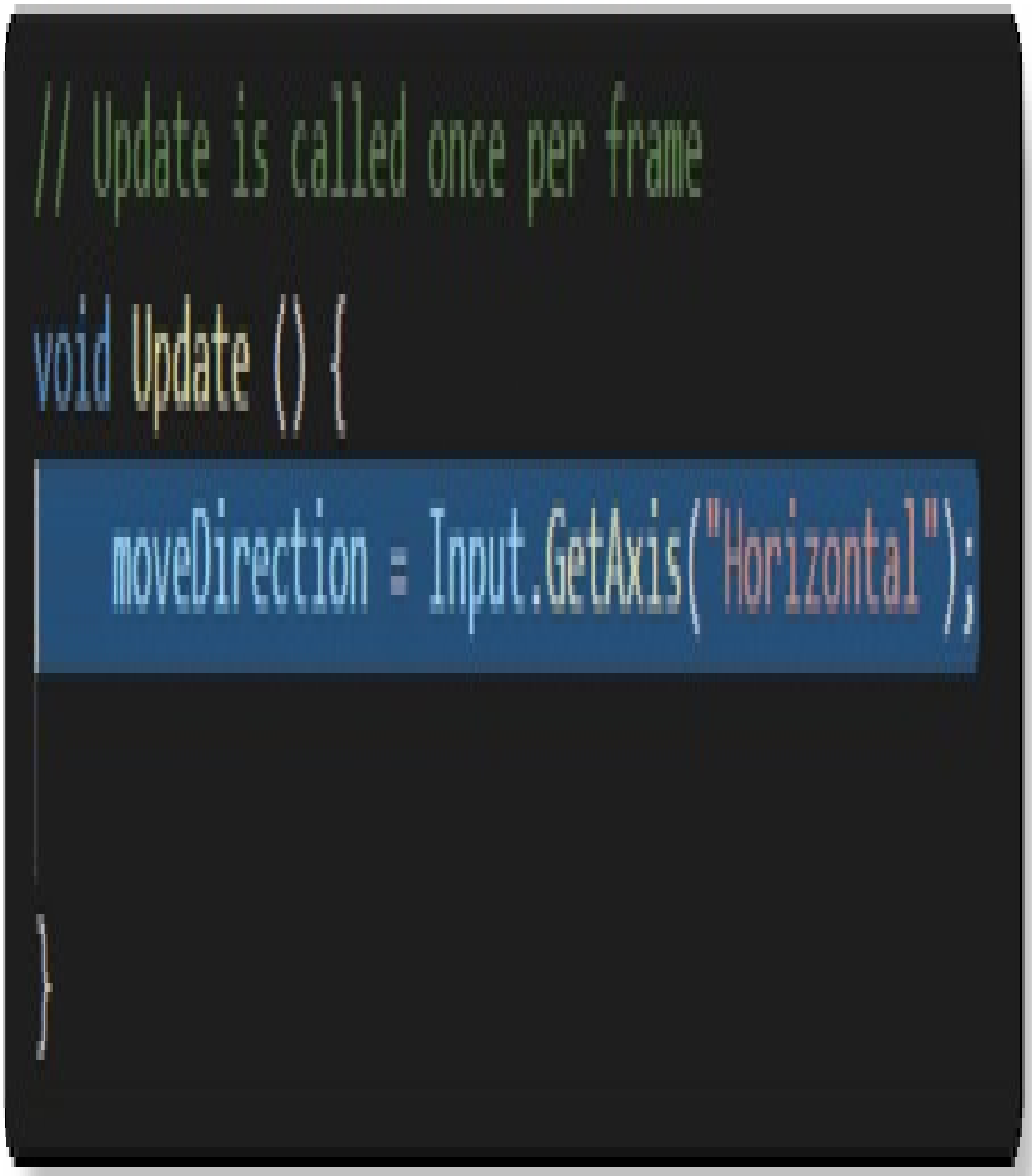
El primer vector X, será afectado por el valor (todavía no asignado) de la variable “moveDirection” por “maxSpeed” que actualmente le asignamos un valor de 6.

void FixedUpdate ()
{
rigidbody.velocity = new Vector2 (moveDirection * maxSpeed, rigidbody.velocity.y);
}

Valor del vector Y

El segundo vector Y, será básicamente el valor único de “y” que se mantendrá constante, ya que con este código únicamente lo queremos mover en el eje X.

Añada el siguiente código en la función “Update” para controlar el movimiento con el teclado.


// Update is called once per frame
void Update () {
moveDirection = Input.GetAxis("Horizontal");
}

Asignación de valor a moveDirection

En esta parte del código asignamos el valor a la variable “moveDirection”.

El comando descrito nos regresa un valor ya preestablecido en el teclado conocido como “Horizontal”, el cual es relativo a la tecla “D” y “A” del teclado y a las flechas derecha e izquierda.

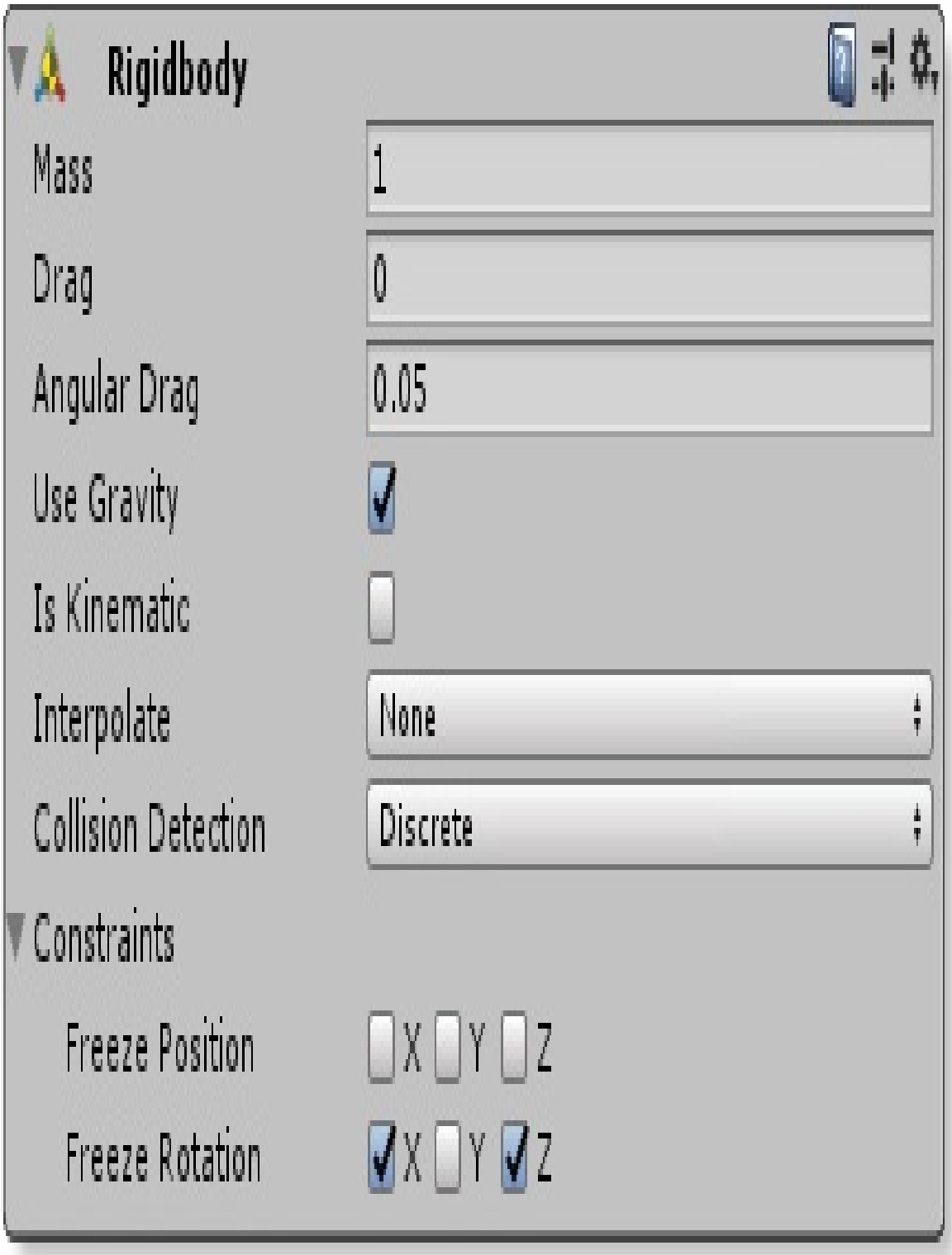
Salve el script

Vaya a Unity, corra el juego para previsualizar el movimiento del “Player” con las teclas A y D o con las flechas derecha o izquierda

PROBLEMA: El personaje rueda en vez de mantenerse de pie.

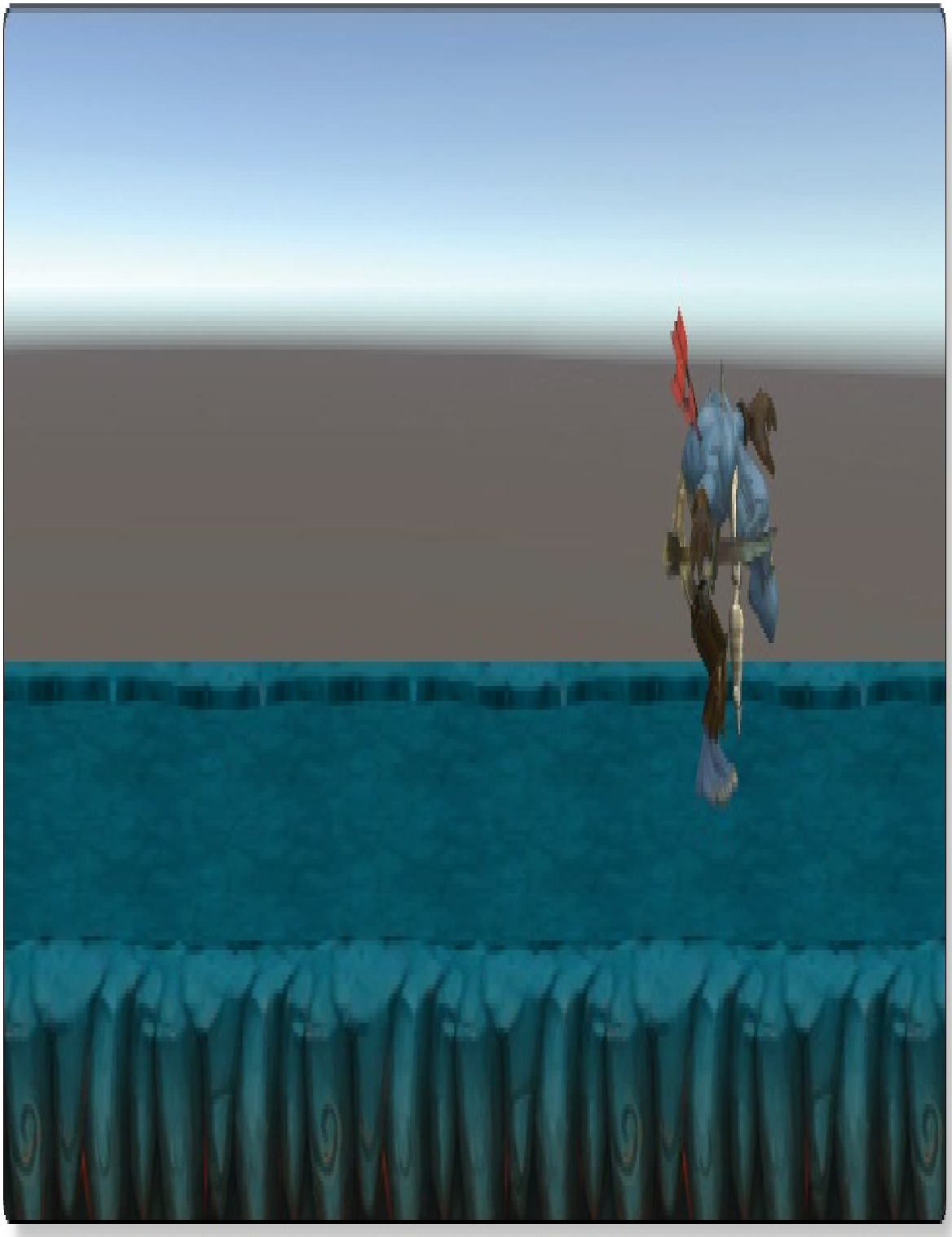
Problema con el movimiento

SOLUCIÓN: Seleccione “Player” y en Inspector asigne los valores que se muestran en Constraint dentro del componente RigidBody.


Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Z
Freeze Rotation
X
Y
Z

Constraints del RigidBody

Previsualice el juego para observar que el personaje se mueve correctamente hacia la derecha y hacia la izquierda.



Movimiento correcto

PROGRAMACIÓN DEL VOLTEO DEL PLAYER

En este apartado abarcaremos los operadores relacionales, así como el uso de las estructuras condicionales para programar el volteo horizontal del Player.

Operadores Relacionales

Los operadores relacionales nos sirven para realizar operaciones con resultados lógicos. La tabla mostrada indica los tipos de operadores que vamos a utilizar, su significado y su uso.

Operador	nombre	ejemplo	significado
<	menor que	*a*<*b*	*a* es menor que *b*
>	mayor que	*a*>*b*	*a* es mayor que *b*
==	igual a	*a*==*b*	*a* es igual a *b*
!=	no igual a	*a*!=*b*	*a* no es igual a *b*
<=	menor que o igual a	*a*<=5	*a* es menor que o igual a *b*
>=	mayor que o igual a	*a*>=*b*	*a* es menor que o igual a *b*

Operadores relacionales

Operadores booleanos y tablas de verdad

Los operadores booleanos son AND, OR y NOT y su funcionamiento obedece a las siguientes tablas de verdad.

El operador lógico AND

x	y	resultado
true	true	true
true	false	false
false	true	false
false	false	false

El operador lógico OR

x	y	resultado
true	true	true
true	false	true
false	true	true
false	false	false

El operador lógico NOT

x	resultado
true	false
false	true

Tablas de verdad

Los operadores AND y OR combinan expresiones relacionales cuyo resultado viene dado por la última columna de sus tablas de verdad. Por ejemplo:

(a<b) && (b<c)

es verdadero (true), si ambas son verdaderas. Si alguna o ambas son falsas el resultado es falso (false). En cambio, la expresión.

(a<b) || (b<c)

es verdadera si una de las dos comparaciones lo es. Si ambas, son falsas, el resultado es falso.

La expresión “ NO a es menor que b”

!(a<b)

Vaya al script de “CharacterMovement” y escriba la siguiente función para programar el volteo horizontal.

void FixedUpdate (){
rigidbody.velocity = new Vector2 (moveDirection * maxSp
}
void Flip (){
facingRight = !facingRight;
transform.Rotate(Vector3.up, 180.0f, Space.World);
}
}

Función Flip

En esta función estamos asignando un nuevo valor a la variable “facingRight” y a su vez le estamos indicando que va a ser rotada 180 grados en su eje Y.

NOTA:

La función es creada, pero por sí sola no realiza nada ya que no la hemos mandado llamar en ningún evento

Estructuras condicionales (if)

Las estructuras condicionales utilizan la nomenclatura “ if “, “then” y “else” que en español se entendería como:

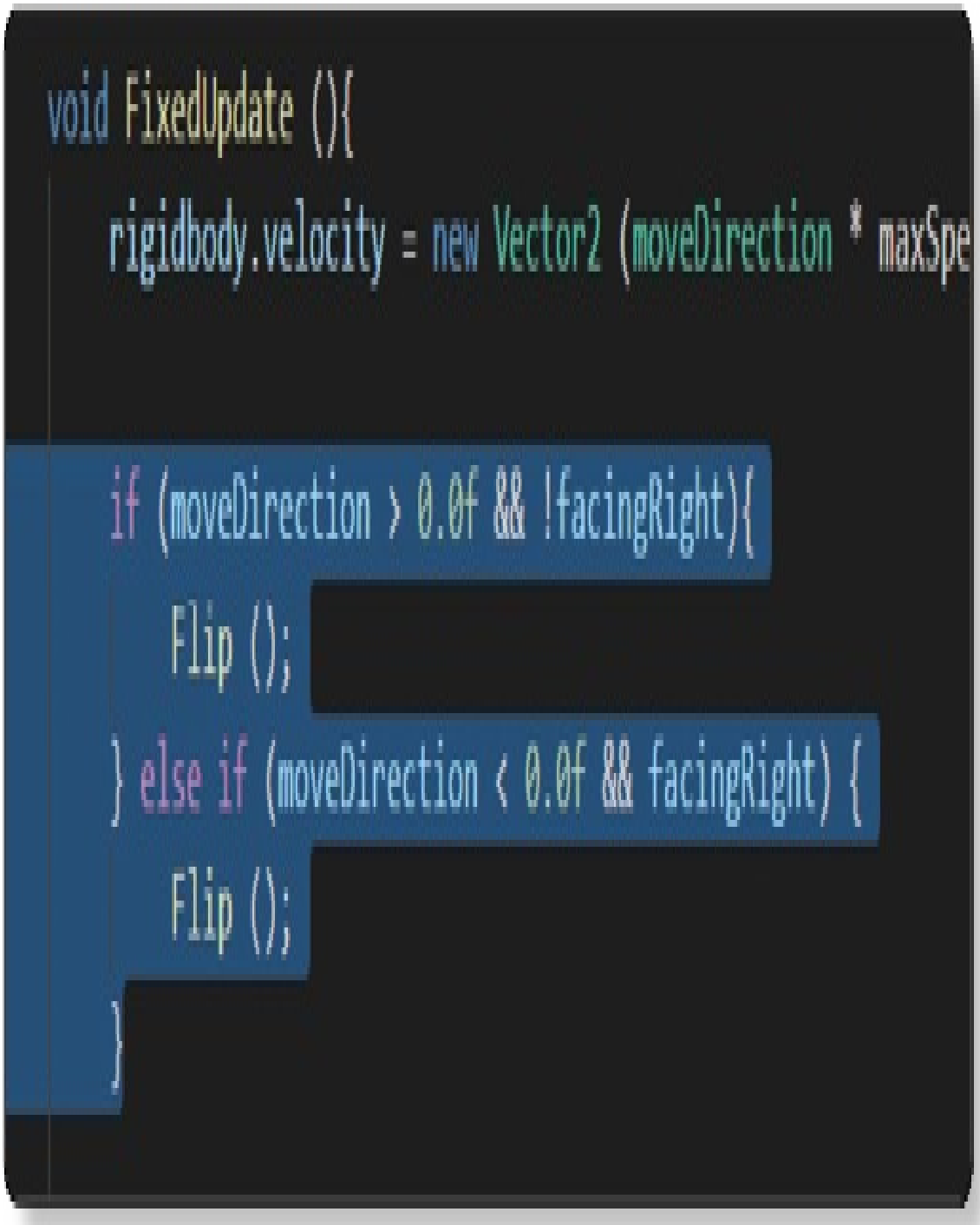
Si “pasa esto” (if)

Entonces (then)

Y si sucede cualquier otra cosa (else)

En la función “FixedUpdate” escriba el siguiente código para mandar llamar la

función creada con la estructura condicional “if”


void FixedUpdate (){
rigidbody.velocity = new Vector2 (moveDirection * maxSpe
if (moveDirection > 0.0f && !facingRight){
Flip ();
} else if (moveDirection < 0.0f && facingRight) {
Flip ();
}

Estructura condicional If

En esta parte del código estamos creando una condición para que cuando la variable “moveDirection” cumpla las condiciones escritas, mande llamar la función Flip.

NOTA:

Puede visualizar el cambio de la variable “moveDirection” en Unity cuando mueve el personaje de atrás hacia adelante.

Previsualice el juego para observar la funcionalidad del script.

Volteo correcto

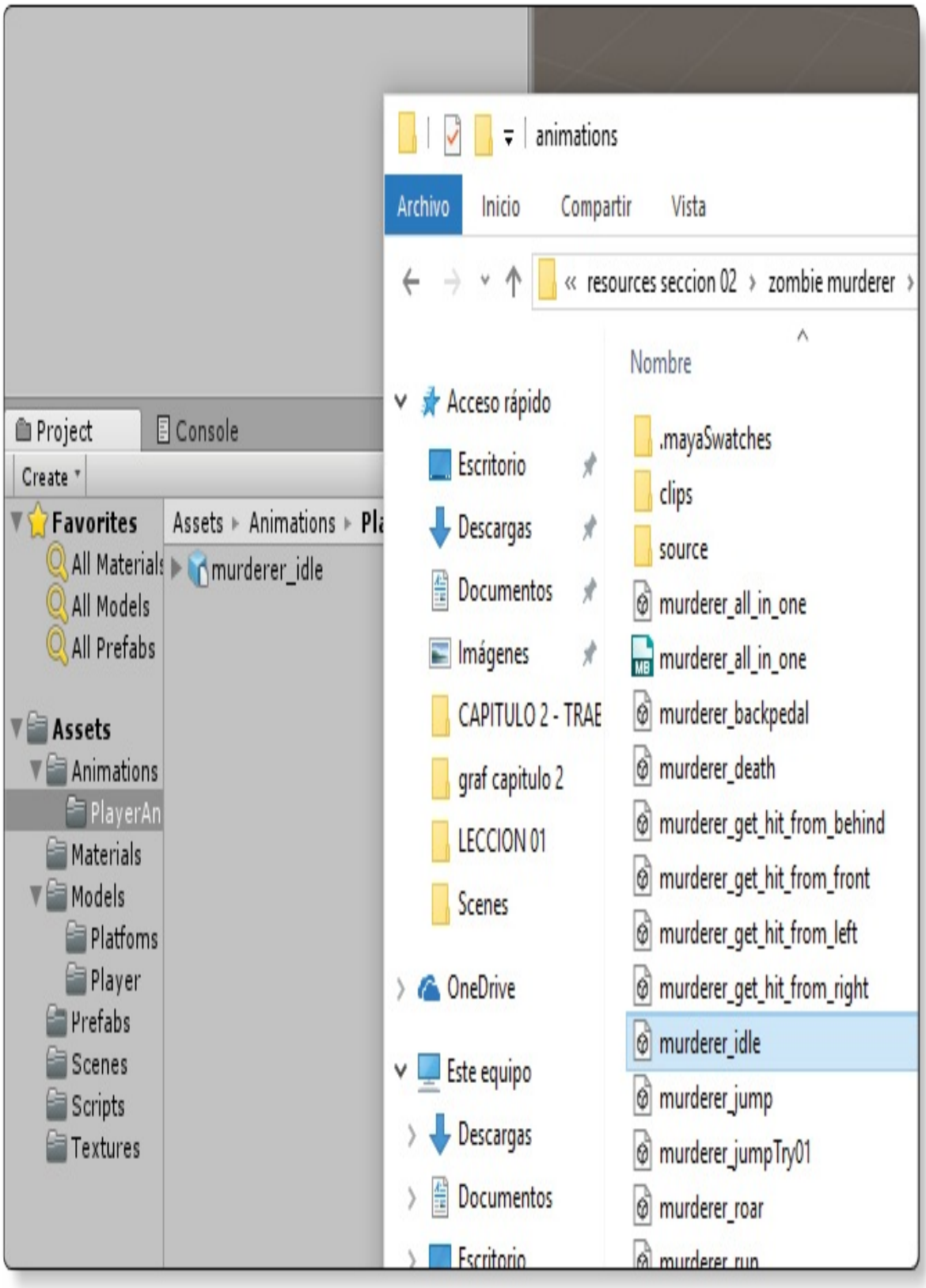
ANIMACIóN IDLE DEL PLAYER

En este apartado conoceremos como importar animaciones a nuestro Player y al uso del componente Animator para controlar sus animaciones.

En el folder de Assets, haga un nuevo folder y nómbrelo “Animations”

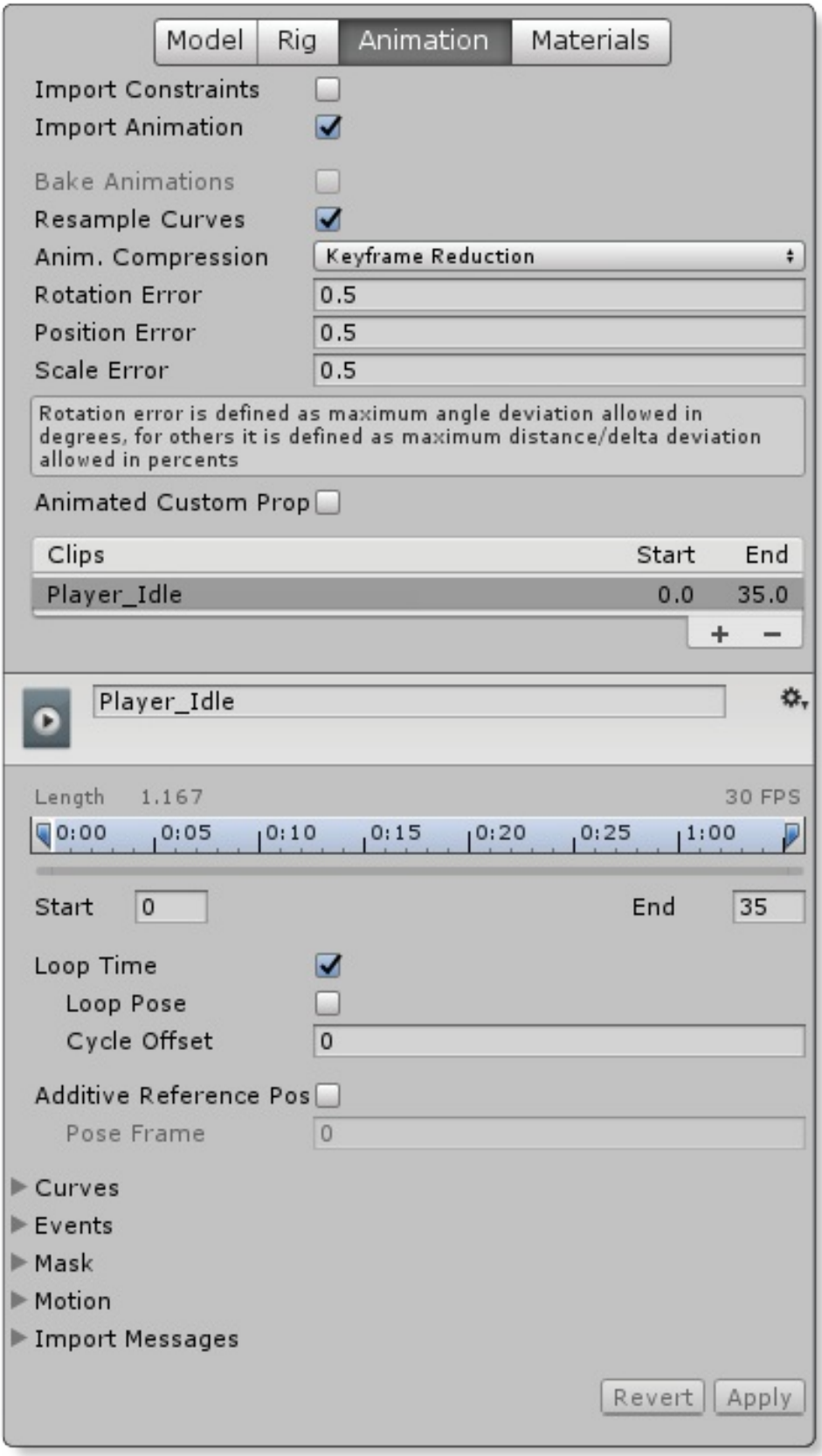
Dentro haga otro folder y nómbrelo “PlayerAnimations”

Importe dentro la animación “murderer_idle” del folder de “resources”.


animations
Archivo
Inicio
Compartir
Vista
« resources seccion 02 › zombie murderer ›
Nombre
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
CAPITULO 2 - TRAE
graf capitulo 2
LECCION 01
Scenes
OneDrive
Este equipo
Descargas
Documentos
Escritorio
.mayaSwatches
clips
source
murderer_all_in_one
murderer_all_in_one
murderer_backpedal
murderer_death
murderer_get_hit_from_behind
murderer_get_hit_from_front
murderer_get_hit_from_left
murderer_get_hit_from_right
murderer_idle
murderer_jump
murderer_jumpTry01
murderer_roar
murderer_run
Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platfoms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Animations ▸ Pl
murderer_idle

Archivo de animación Idle

Seleccione el objeto “murderer_idle” y en “Inspector” clic en la pestaña “Animation” asigne los siguientes valores y APPLY.


Model
Rig
Animation
Materials
Import Constraints
Import Animation
Bake Animations
Resample Curves
Anim. Compression
Keyframe Reduction
Rotation Error
0.5
Position Error
0.5
Scale Error
0.5
Rotation error is defined as maximum angle deviation allowed in degrees, for others it is defined as maximum distance/delta deviation allowed in percents
Animated Custom Prop
Clips
Start
End
Player_Idle
0.0
35.0
Player_Idle
Length 1.167
30 FPS
0:00
0:05
0:10
0:15
0:20
0:25
1:00
Start
0
End
35
Loop Time
Loop Pose
Cycle Offset
0
Additive Reference Pos
Pose Frame
0
Curves
Events
Mask
Motion
Import Messages
Revert
Apply

Creación de clip de animación Idle

De esta manera es como creamos un clip de animación desde una animación previamente generada en un software de animación, llámese Maya, 3D Studio, Blender.

La casilla “Loop” debemos de marcarla cuando se trate de una animación que queremos que se encuentre en ciclo.

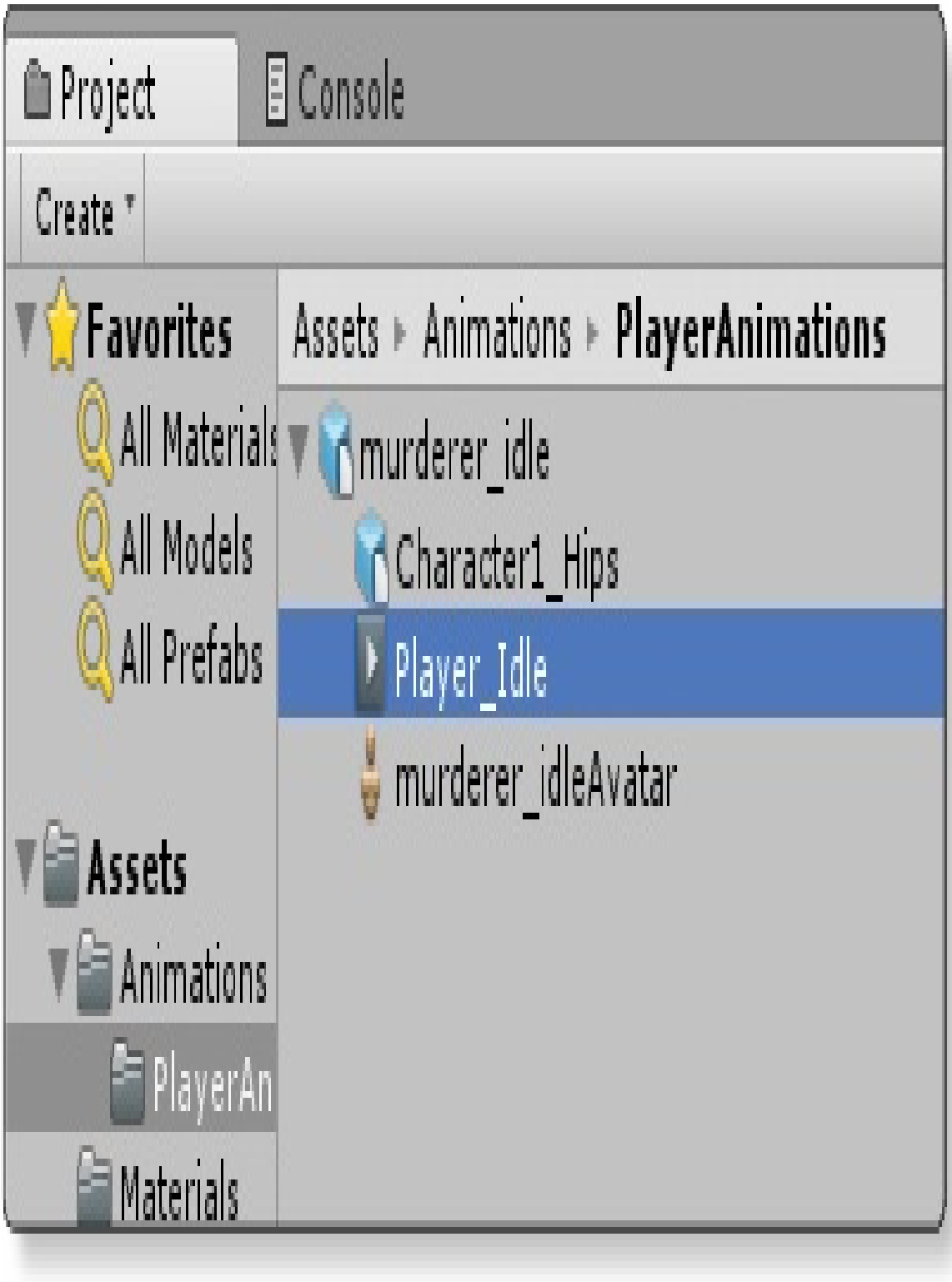
NOTA:

Expanda el indicador azul derecho hasta el cuadro 35.

NOTA:

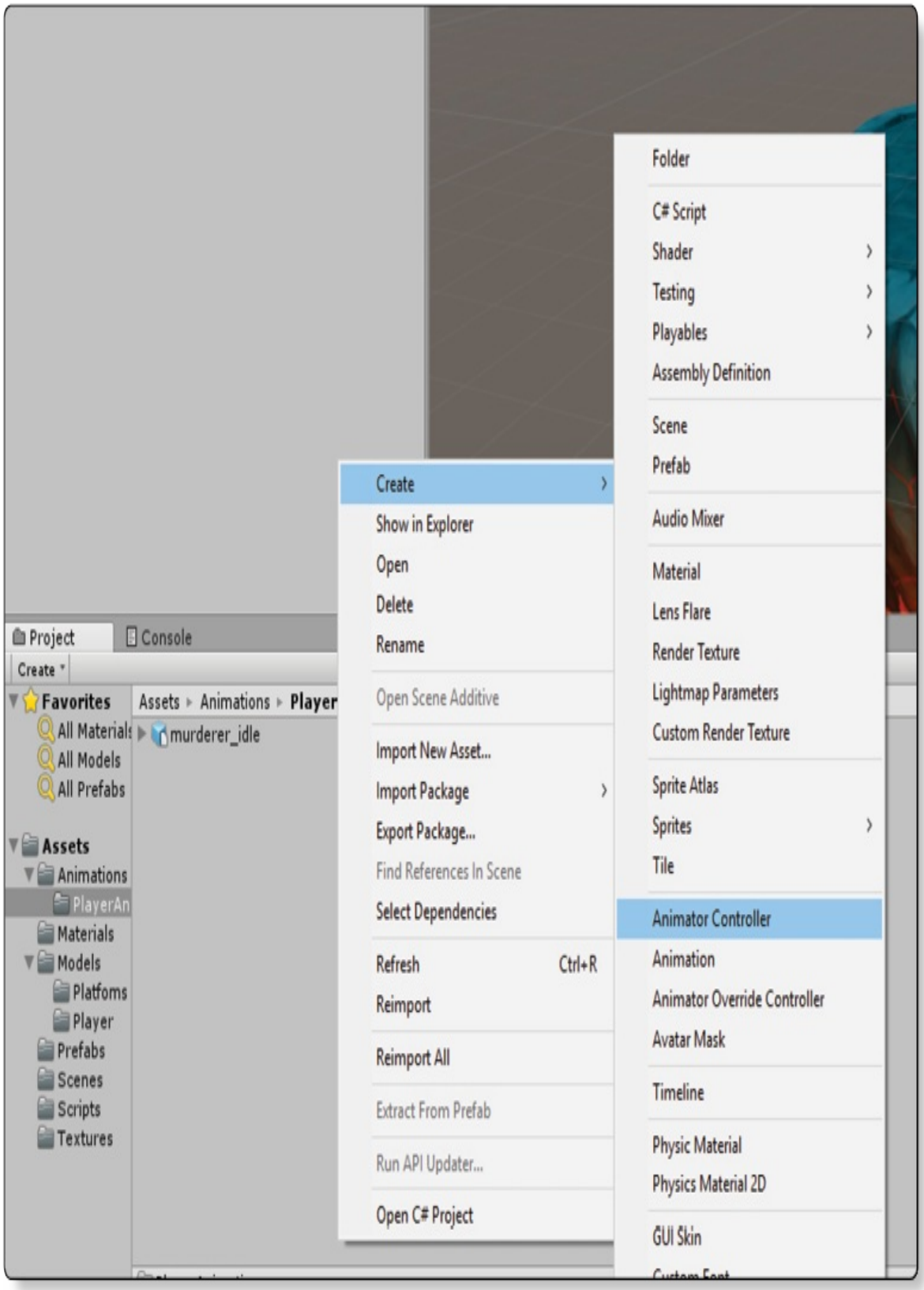
Después de escribir el nombre “Player_Idle” deberá de dar clic en “Enter” en el teclado para que se active la opción “APPLY”.

En la ventana de “Project” expanda “murderer_idle” para observar el clip de animación creado.

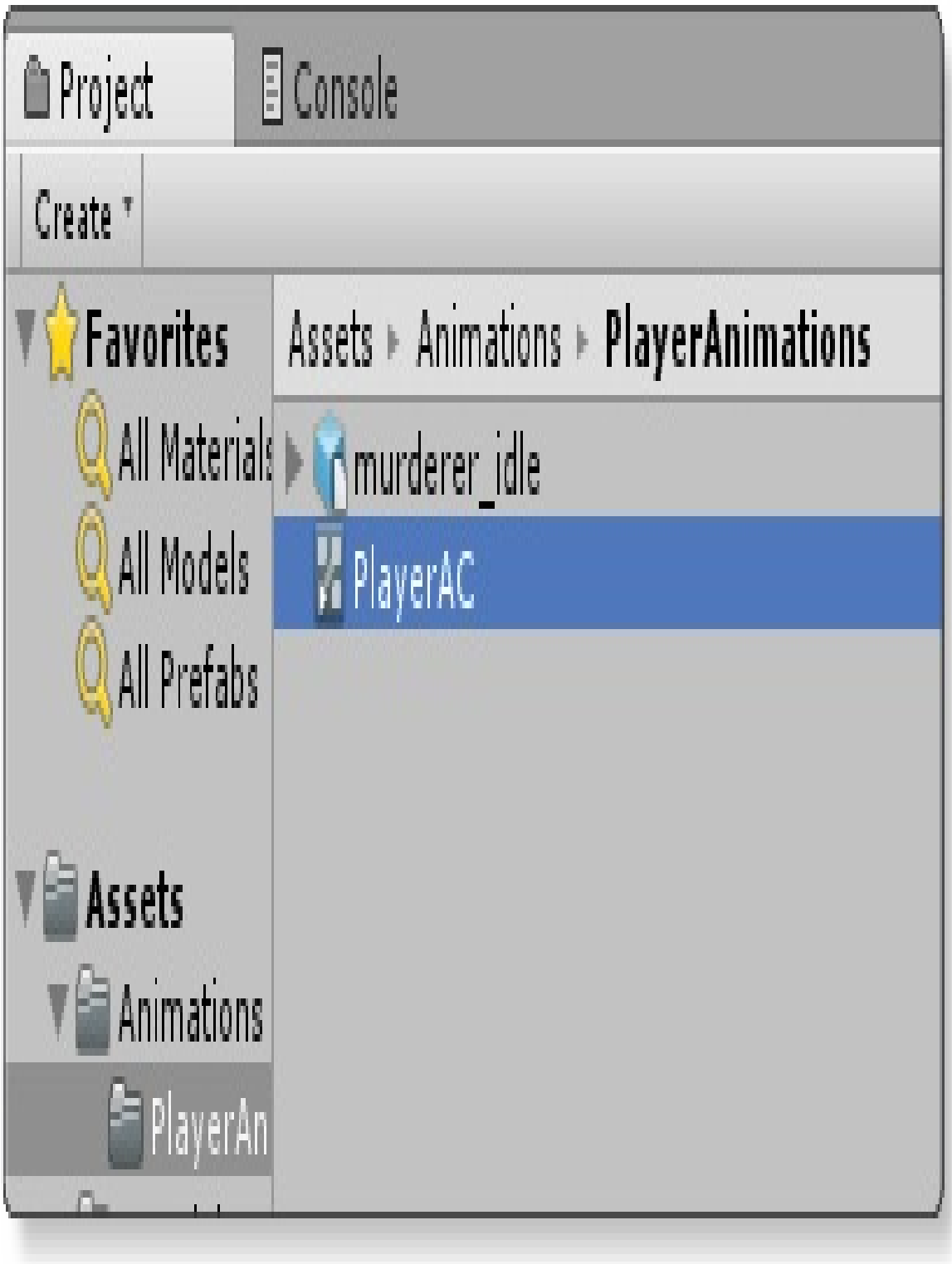

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Assets ▸ Animations ▸ PlayerAnimations
murderer_idle
Character1_Hips
Player_Idle
murderer_idleAvatar

Clip de animación Idle creado

Vaya al folder “PlayerAnimations” y haga un nuevo “AnimatorController” y renómbrelo como “PlayerAC”.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platfoms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Animations ▸ Player
murderer_idle
Create
Show in Explorer
Open
Delete
Rename
Open Scene Additive
Import New Asset...
Import Package
Export Package...
Find References In Scene
Select Dependencies
Refresh Ctrl+R
Reimport
Reimport All
Extract From Prefab
Run API Updater...
Open C# Project
Folder
C# Script
Shader
Testing
Playables
Assembly Definition
Scene
Prefab
Audio Mixer
Material
Lens Flare
Render Texture
Lightmap Parameters
Custom Render Texture
Sprite Atlas
Sprites
Tile
Animator Controller
Animation
Animator Override Controller
Avatar Mask
Timeline
Physic Material
Physics Material 2D
GUI Skin

Creación de un AnimatorController

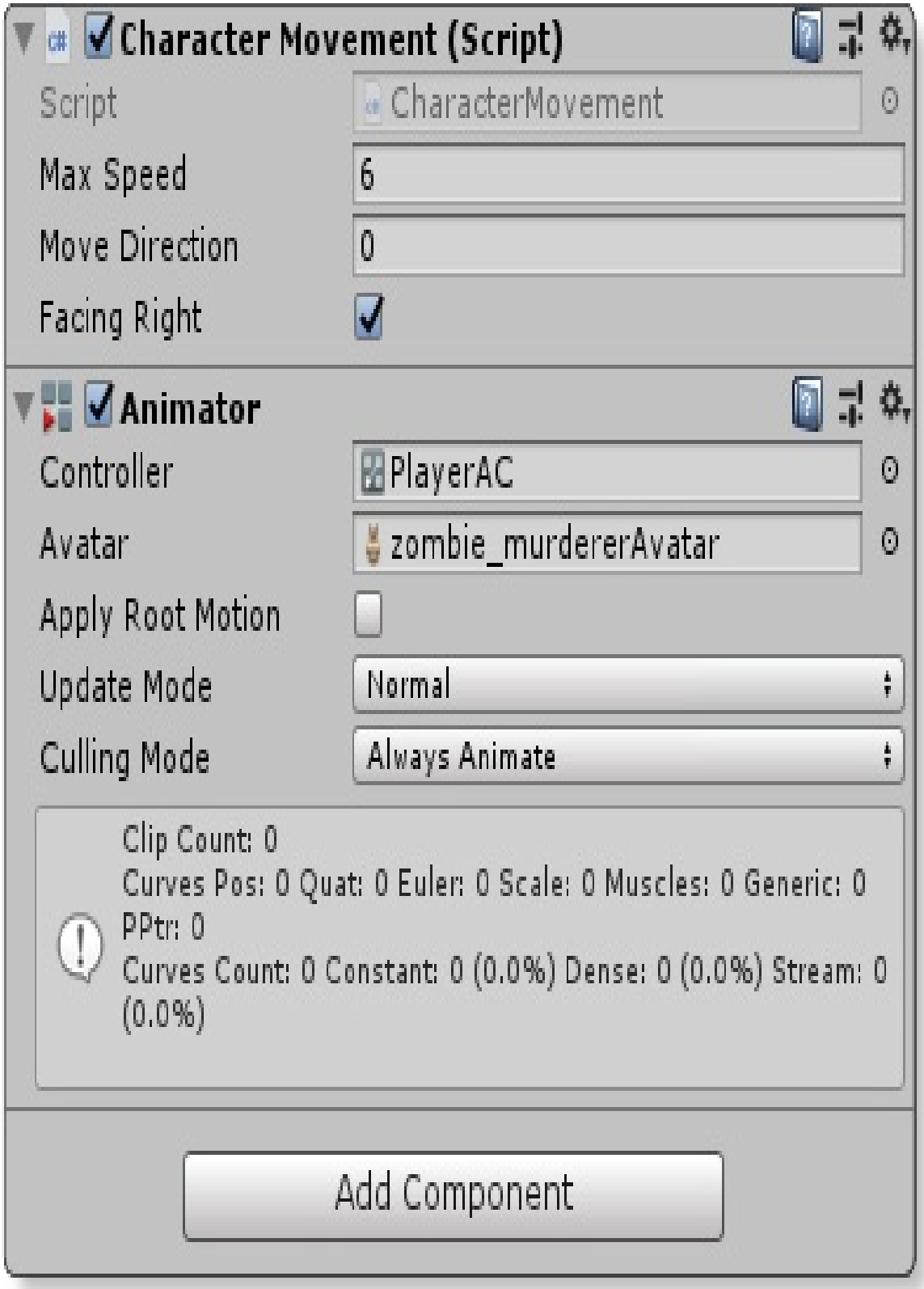

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Assets
Animations
PlayerAnimations
murderer_idle
PlayerAC

AnimatorController creado

Seleccione el Player y asigne un componente “Animator” y asigne como su controlador el “PlayerAC” creado anteriormente y como “Avatar” el “zombie_murdererAvatar”.

Select Avatar
Assets
Scene
None
murderer_idleAvatar
zombie_murdererAvatar

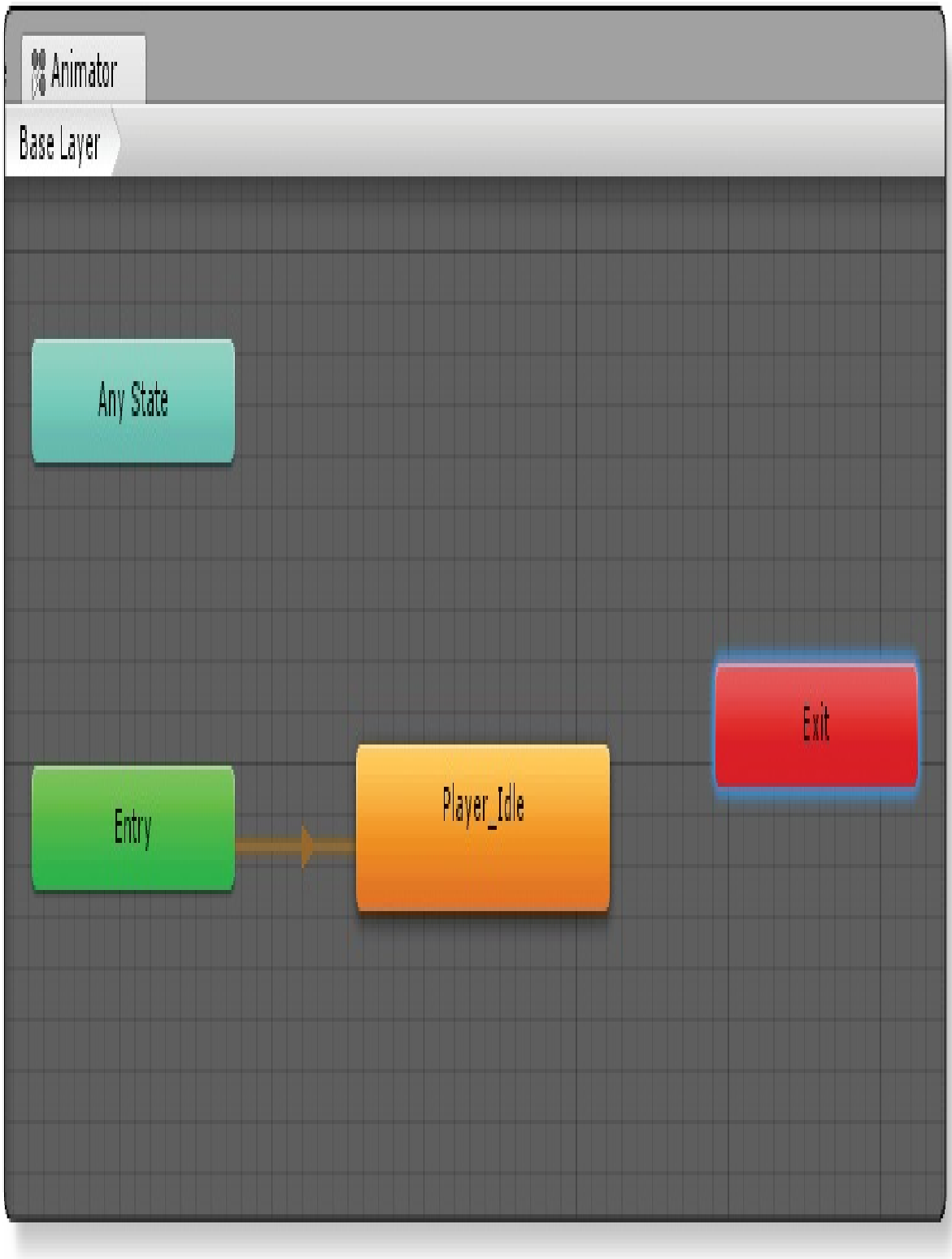
Avatar para el AnimatorController


Character Movement (Script)
Script
CharacterMovement
Max Speed
6
Move Direction
0
Facing Right
Animator
Controller
PlayerAC
Avatar
zombie_murdererAvatar
Apply Root Motion
Update Mode
Normal
Culling Mode
Always Animate
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0
PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0
(0.0%)
Add Component

Asignación de elementos de Animator

En la ventana de “Project” haga doble clic en “PlayerAC” para abrir su ventana “Animator”.

En la ventana “Animator” arrastre el clip creado “Player_Idle”.


Animator
Base Layer
Any State
Exit
Entry
Player_Idle

Ventana Animator

Al arrastrar el primer clip a la ventana Animator, éste aparece de color naranja, lo cual nos indica que es la animación por defecto que va a tener nuestro elemento.

NOTA:

Para navegar en la ventana “Animator” puede hacerlo con Alt + clic izq.

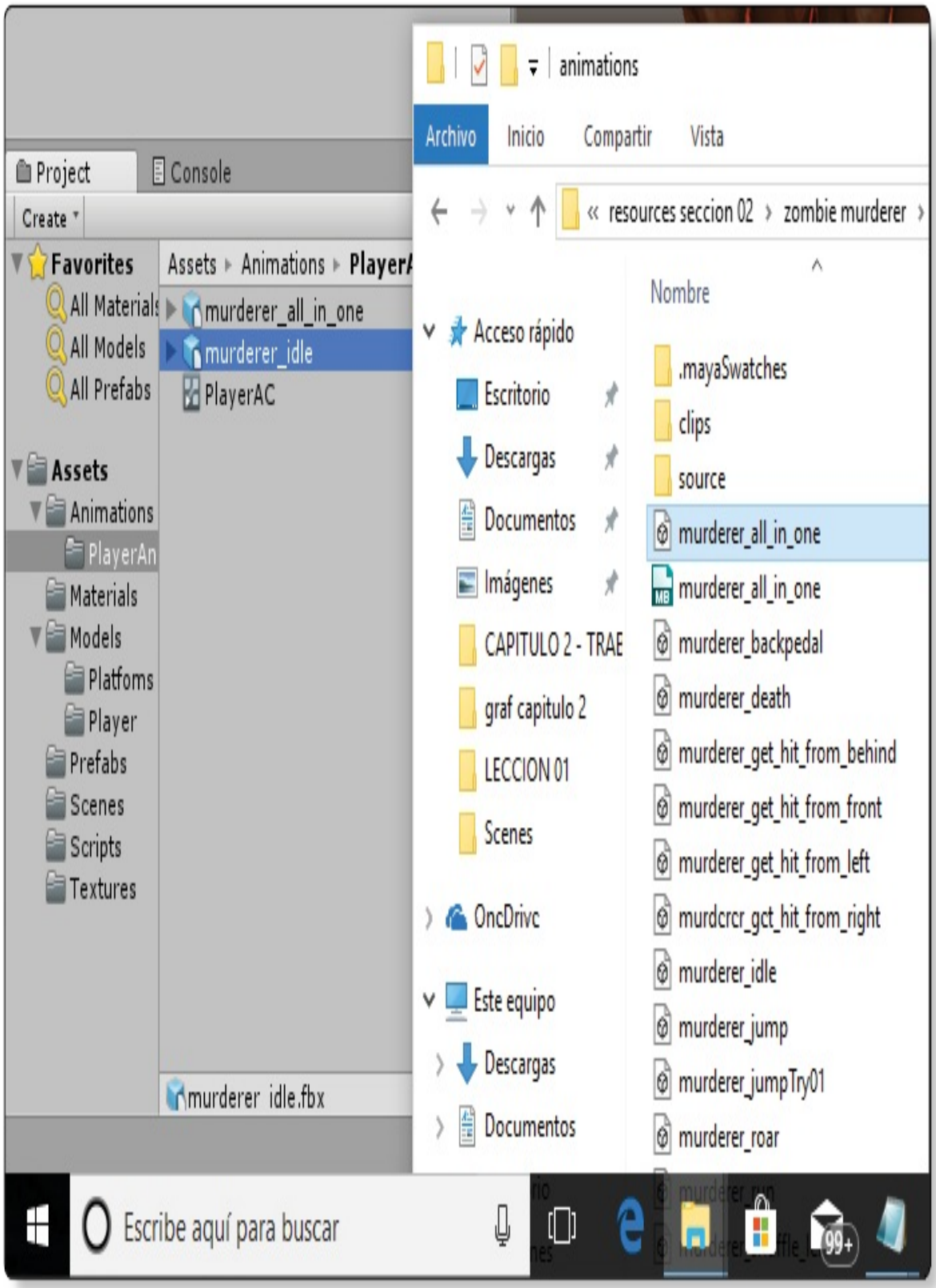
Previsualice el juego para observar la animación “Idle” del “Player”.

Animación Idle

ANIMACIÓN RUNNING DEL PLAYER CON UN BLEND TREE

En este apartado crearemos una nueva animación a partir de un archivo que contiene todas las animaciones del personaje. Se hará un “Blend Tree” en “Animator” que contendrá la animación “Idle” y “Running” del nuestro “Player” y nos introduciremos a los parámetros de animación para controlar las animaciones a través de scripting.

Vaya al folder “PlayerAnimations” e importe el archivo “murderer_all_in_one” de la carpeta “Animations” de “resources”.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platfoms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Animations ▸ PlayerA
murderer_all_in_one
murderer_idle
PlayerAC
murderer idle.fbx
animations
Archivo
Inicio
Compartir
Vista
« resources seccion 02 › zombie murderer ›
Nombre
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
CAPITULO 2 - TRAE
graf capitulo 2
LECCION 01
Scenes
OncDrivc
Este equipo
Descargas
Documentos
.mayaSwatches
clips
source
murderer_all_in_one
murderer_all_in_one
murderer_backpedal
murderer_death
murderer_get_hit_from_behind
murderer_get_hit_from_front
murderer_get_hit_from_left
murdcrcr_gct_hit_from_right
murderer_idle
murderer_jump
murderer_jumpTry01
murderer_roar
Escribe aquí para buscar
99+

Importar archivo con todas las animaciones

Este archivo contiene todas las animaciones del personaje, dentro del mismo folder se puede encontrar el archivo.txt “zombie_murderer_animation_list” que contiene la información de los tiempos específicos en los que corre la animación y mismos que se enumeran a continuación.

Zombie Murderer Animation List

frame: 0030-0059 backpedal (loop)

frame: 0090-0116 murderer death

frame: 0120-0159 get hit from behind

frame: 0165-0209 get hit from front

frame: 0215-0249 get hit from left

frame: 0255-0289 get hit from right

frame: 0329-0363 idle (loop)

frame: 0405-0464 jump

frame: 0470-0526 roar

frame: 0553-0576 run (loop)

frame: 0629-0648 shuffle left (loop)

frame: 0694-0713 shuffle right (loop)

frame: 0740-0784 slash

frame: 0795-0804 slash start

frame: 0833-0862 slash loop (loop)

frame: 0891-0905 slash end

frame: 0959-1008 sneak (loop)

frame: 1060-1179 spawn

frame: 1214-1243 strafe left (loop)

frame: 1309-1338 strafe right (loop)

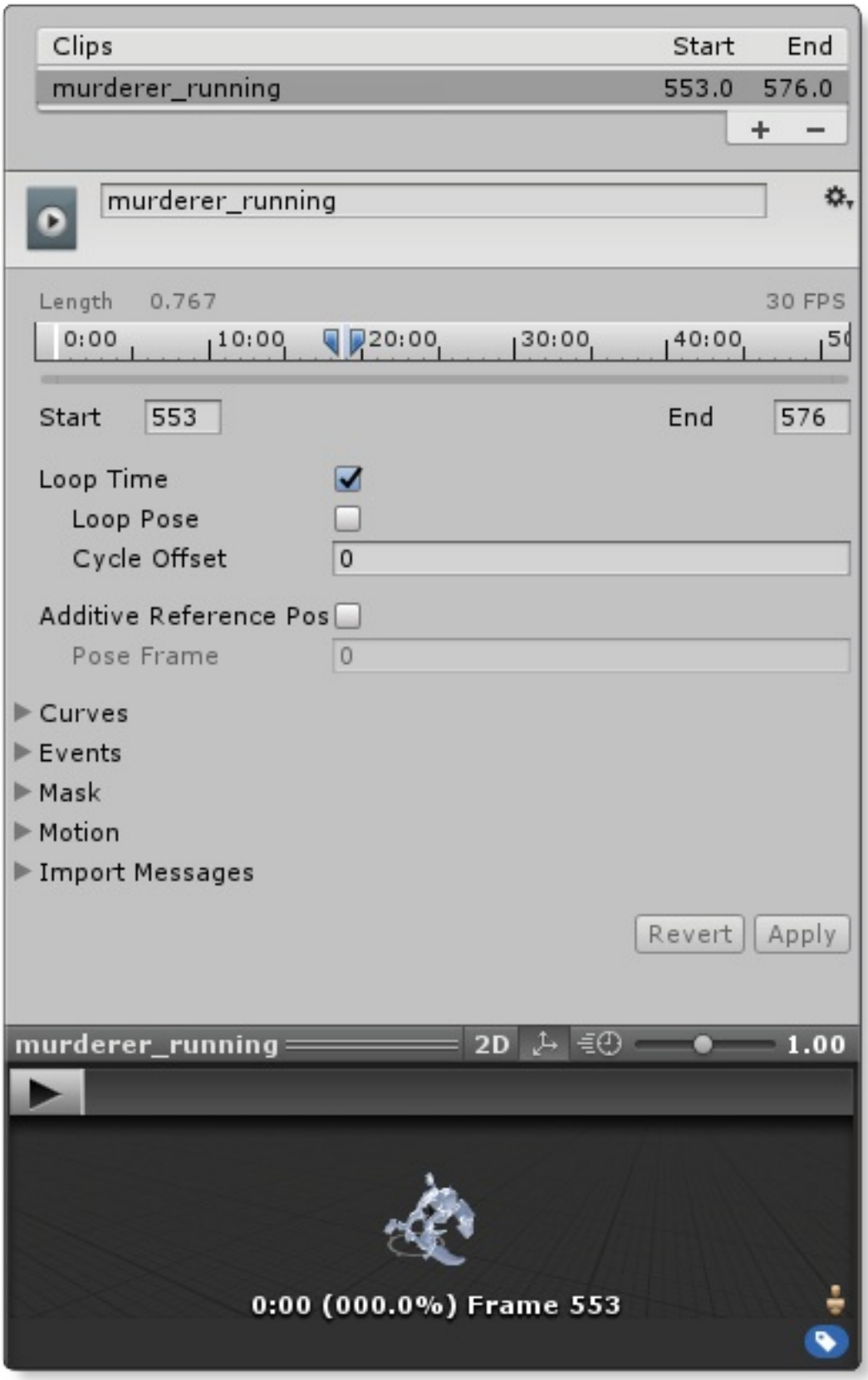
frame: 1375-1420 throw knife

frame: 1459-1488 walk (loop)

Queremos crear el clip de animación relativo al personaje corriendo, para ello tenemos que hacer un clip a partir del archivo “murderer_all_in_one” con solo sus cuadros

frame: 0553-0576 run (loop)

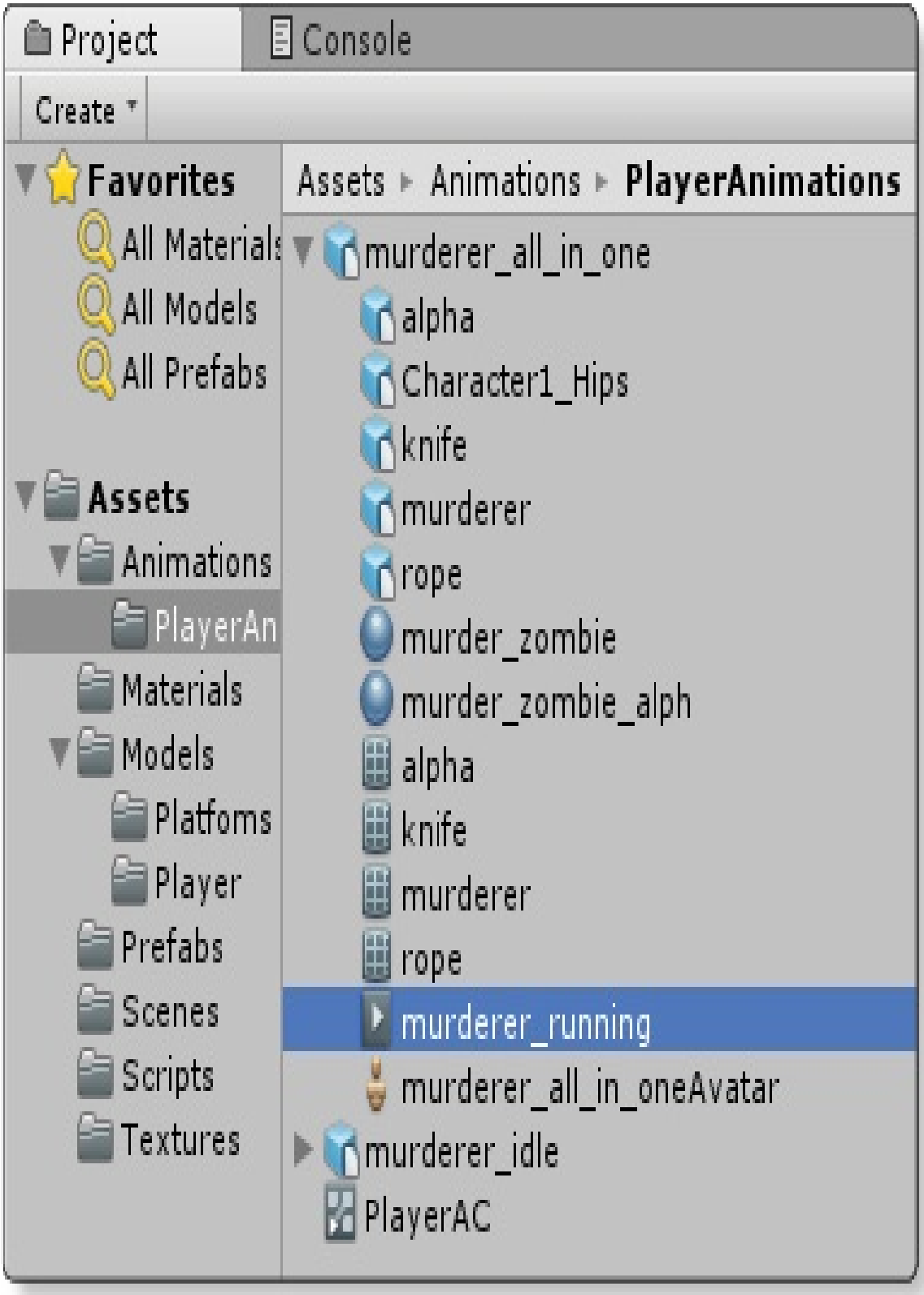
Seleccione el clip “murderer_all_in_one” y en Inspector asigne los siguientes valores y clic en “Apply” para crear el clip de animación.


Clips
Start
End
murderer_running
553.0
576.0
+
−
murderer_running
Length
0.767
30 FPS
0:00
10:00
20:00
30:00
40:00
Start
553
End
576
Loop Time
Loop Pose
Cycle Offset
0
Additive Reference Pos
Pose Frame
0
Curves
Events
Mask
Motion
Import Messages
Revert
Apply
murderer_running
2D
1.00
0:00 (000.0%) Frame 553

Creación de clip “Running”

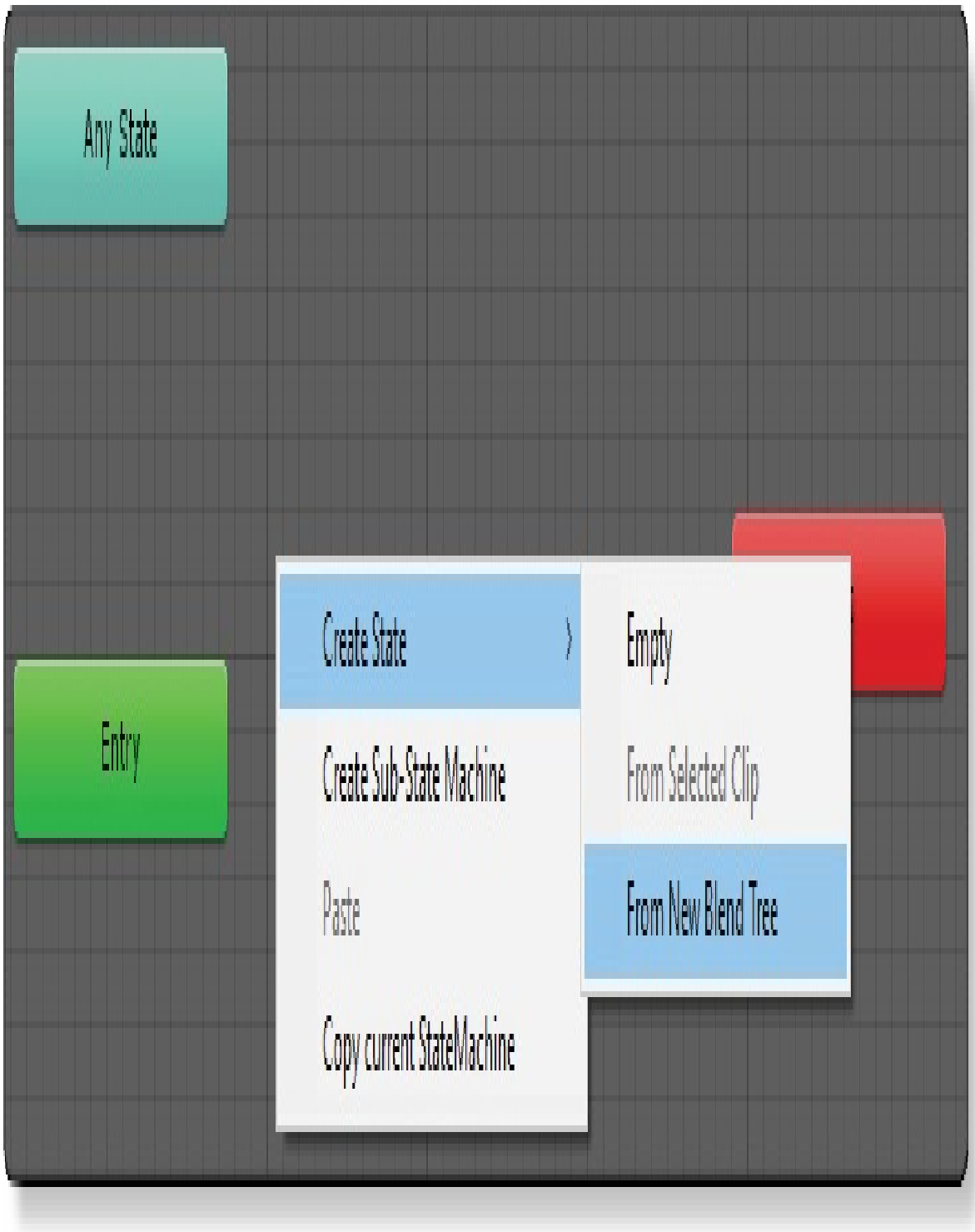
NOTA:

Puede previsualizar la animación en la parte inferior y con el botón “Play” de la ventana.

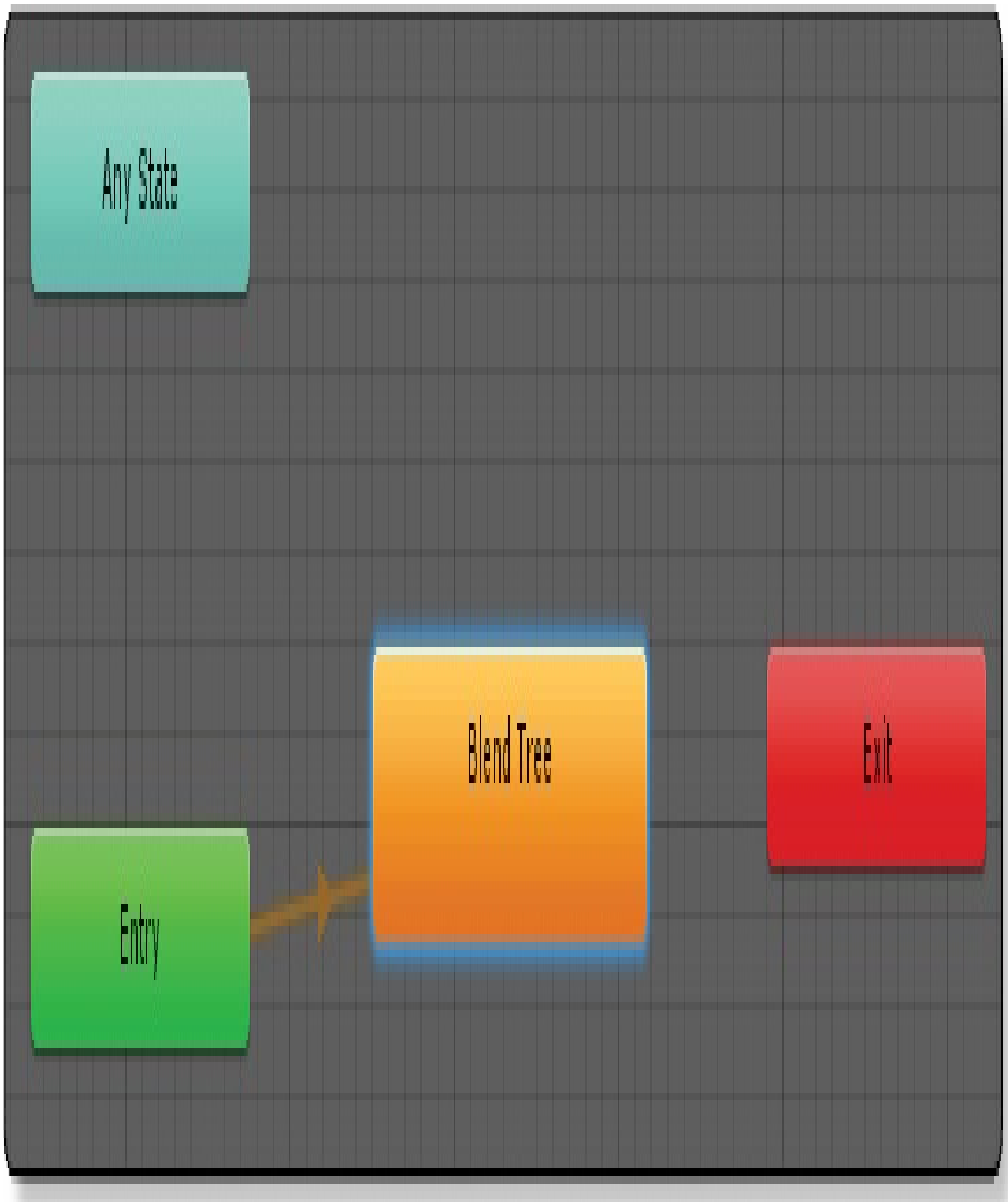

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platfoms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Animations ▸ PlayerAnimations
murderer_all_in_one
alpha
Character1_Hips
knife
murderer
rope
murder_zombie
murder_zombie_alph
alpha
knife
murderer
rope
murderer_running
murderer_all_in_oneAvatar
murderer_idle
PlayerAC

Clip “Running” creado

Vaya a “Animator”, borre la animación “Player_Idle”, asigne clic derecho y haga un “Blend Tree”


Any State
Entry
Create State
Create Sub-State Machine
Paste
Copy current StateMachine
Empty
From Selected Clip
From New Blend Tree

Creación de un “Blend Tree”

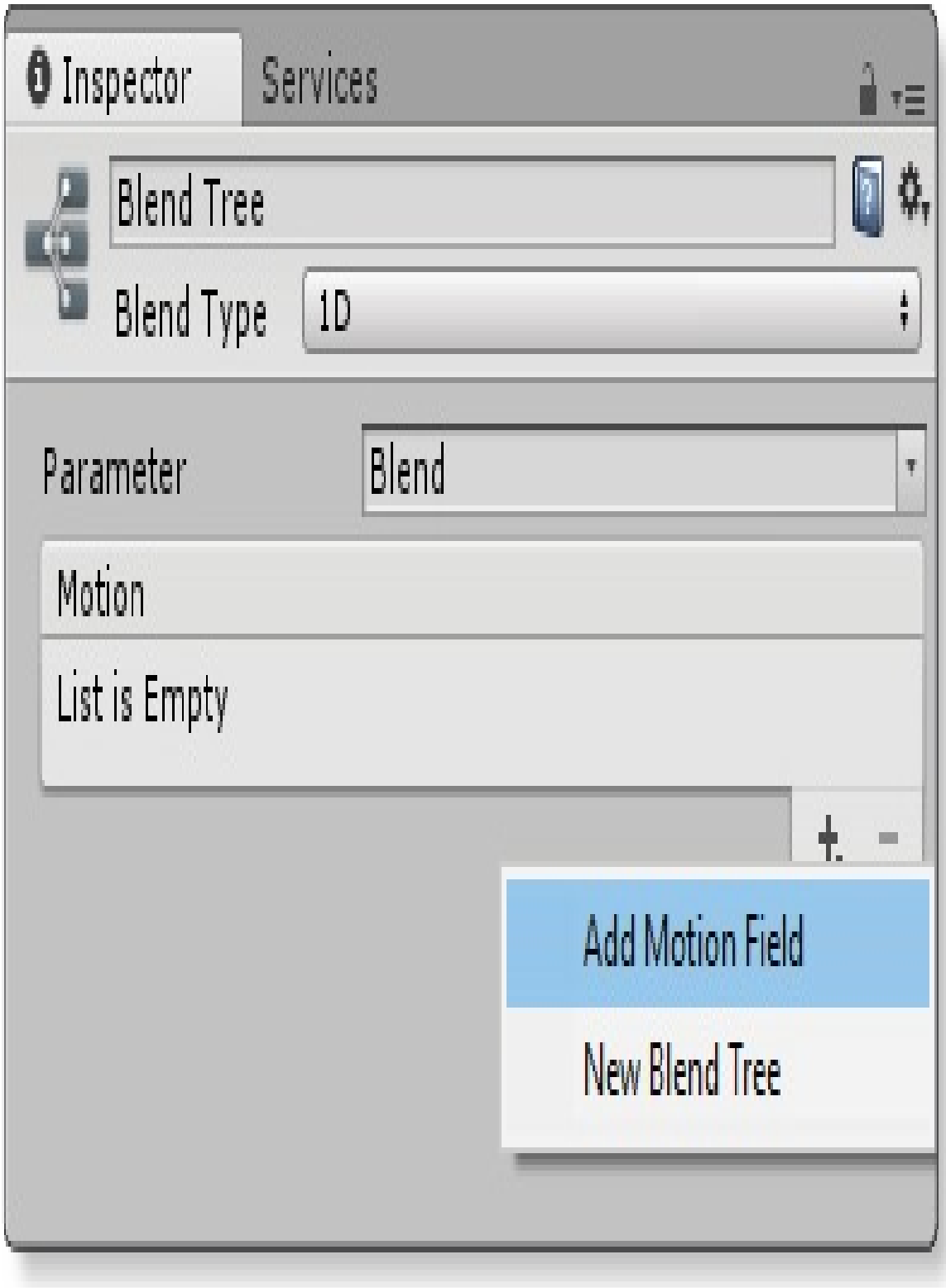

Any State
Blend Tree
Exit
Entry

Blend Tree creado

Dentro de un “Blend Tree” podemos almacenar más de un clip de animación.

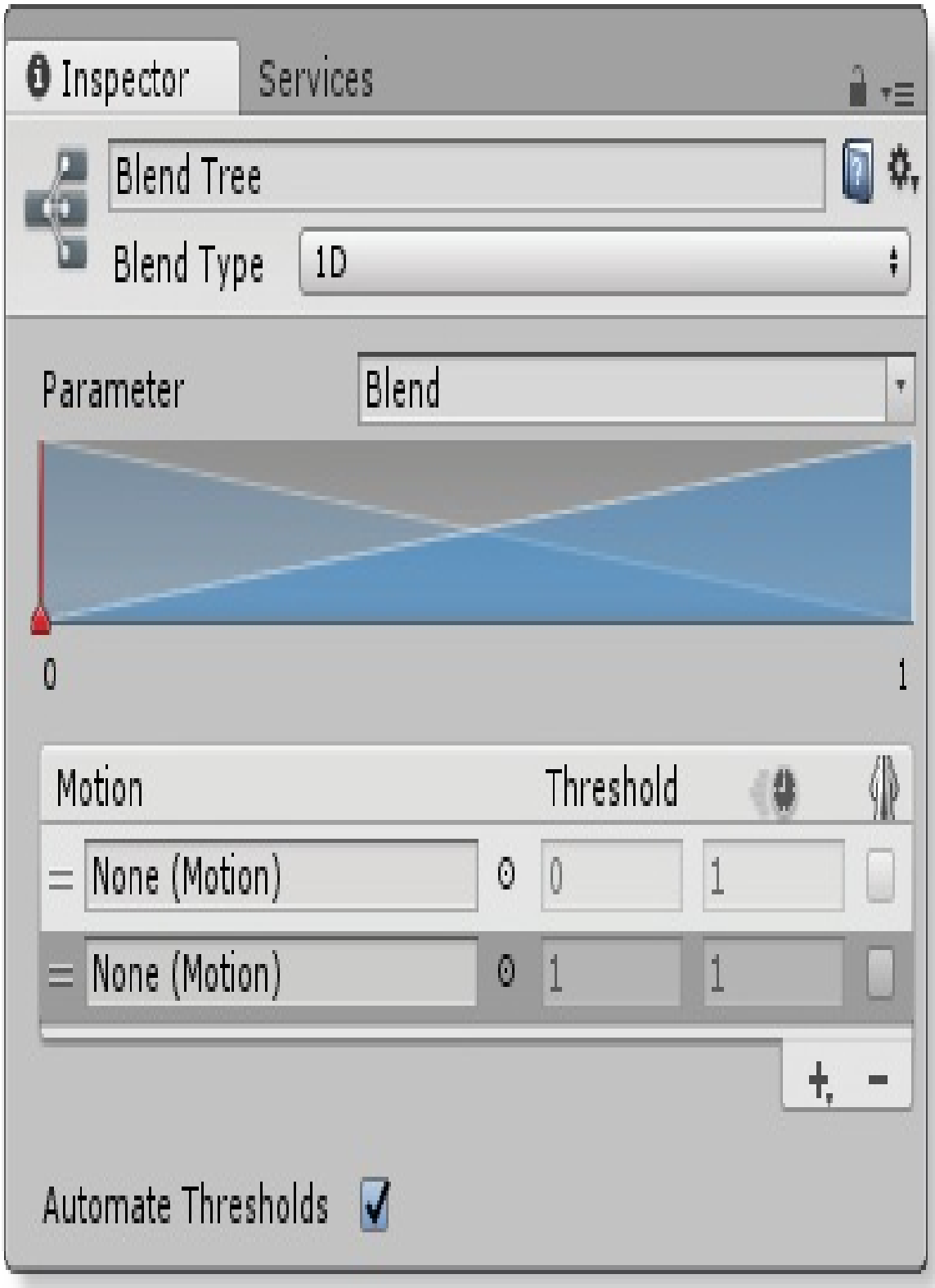
Doble clic en el “Blend Tree” para entrar en su modo de edición

Seleccione “Blend Tree” en “Animator” y en “Inspector” asigne los clips de animación creados.


Inspector
Services
Blend Tree
Blend Type
1D
Parameter
Blend
Motion
List is Empty
Add Motion Field
New Blend Tree

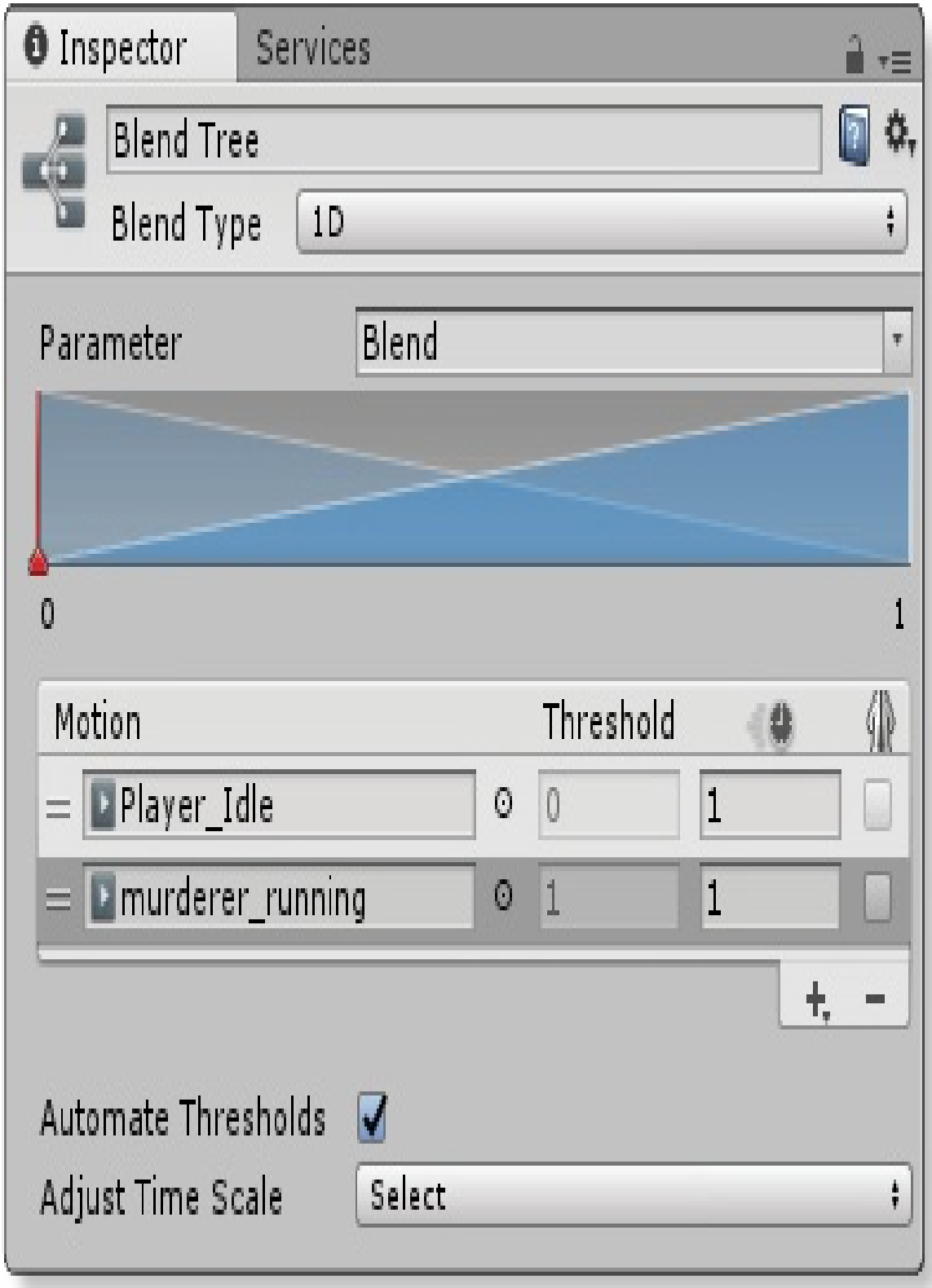
Creación de un nuevo “Motion Field”

Clic en el símbolo de (+) para hacer un nuevo “Motion Field”.


Inspector
Services
Blend Tree
Blend Type
1D
Parameter
Blend
0
1
Motion
Threshold
None (Motion)
0
1
None (Motion)
1
1
Automate Thresholds

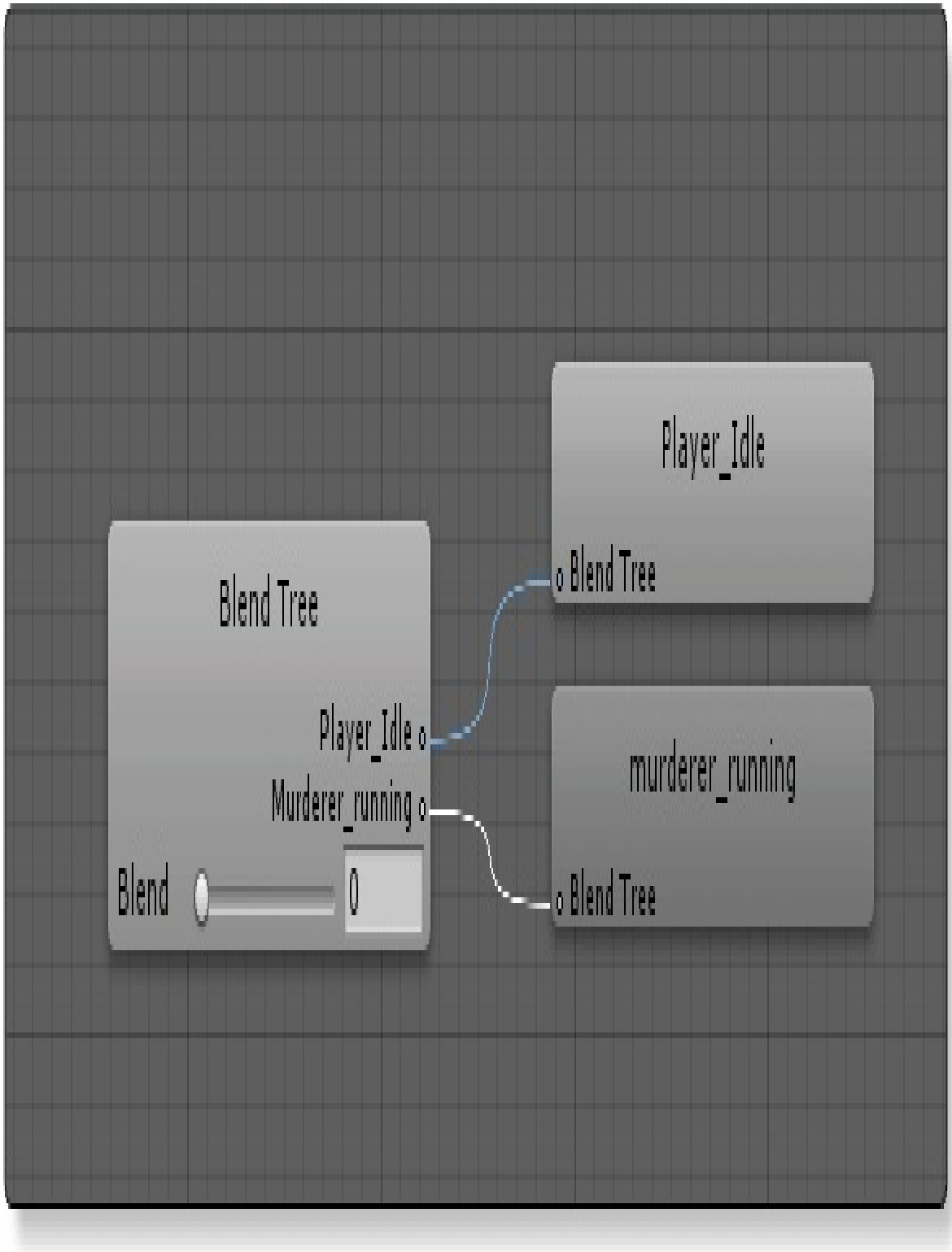
Dos “Motion Fields” creados

Repita el procedimiento para hacer el segundo.


Inspector
Services
Blend Tree
Blend Type
1D
Parameter
Blend
0
1
Motion
Threshold
Player_Idle
0
1
murderer_running
1
1
Automate Thresholds
Adjust Time Scale
Select

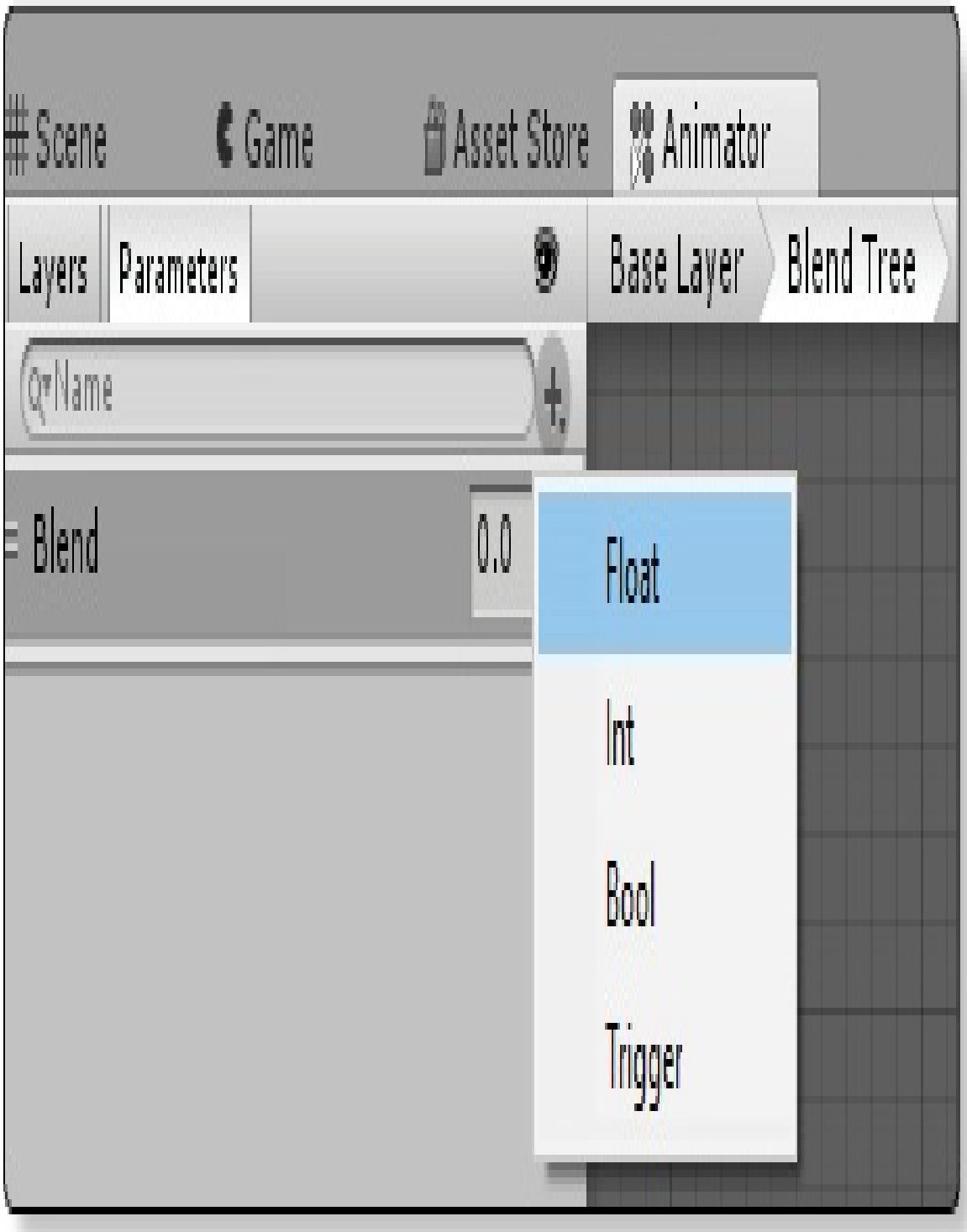
Asignación de animaciones

Añada desde “Player Animations” los clips creados.


Player_Idle
Blend Tree
Blend Tree
Player_Idle
Murderer_running
Blend
0
murderer_running
Blend Tree

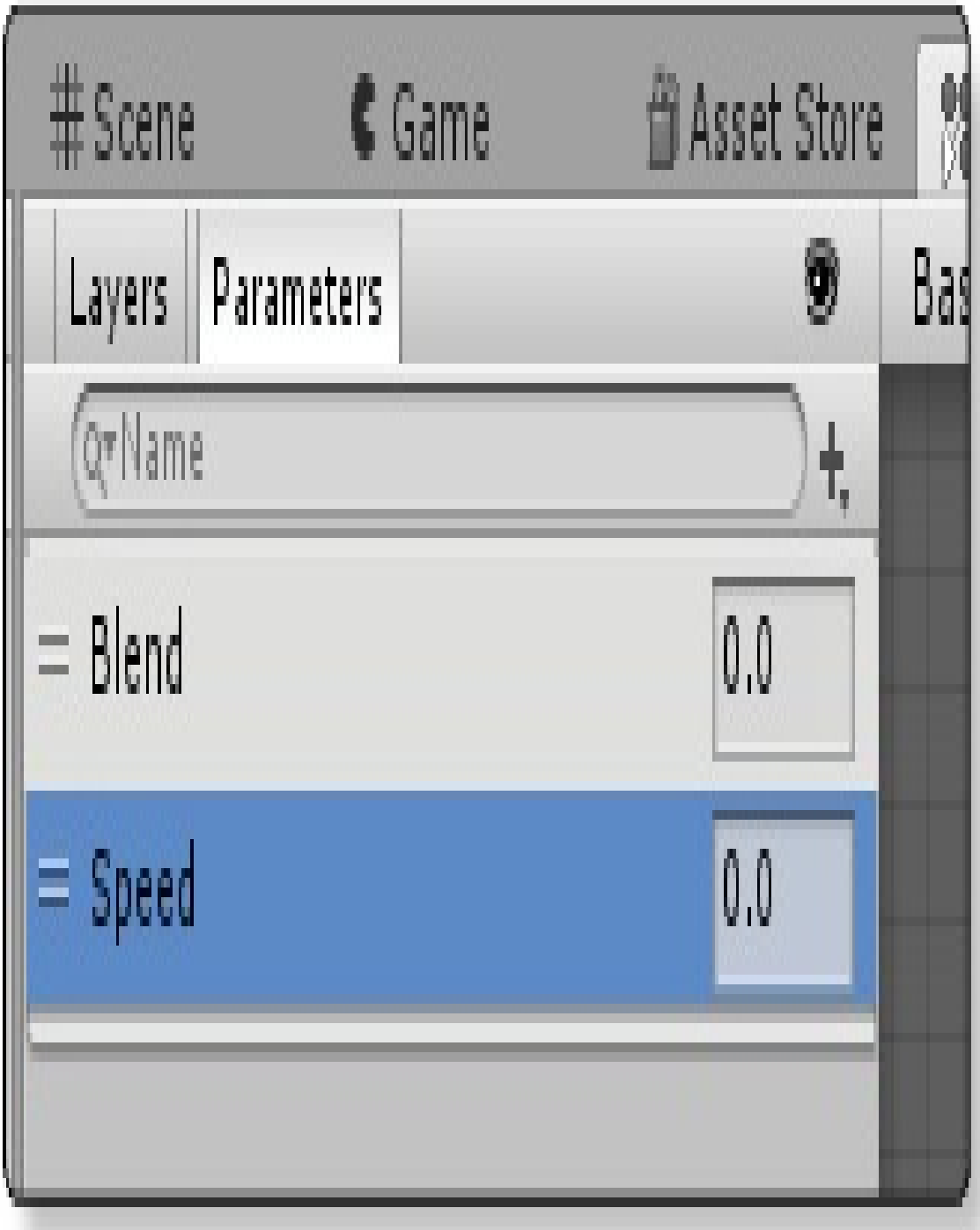
Blend Tree con dos animaciones

En la parte superior izquierda de Animator active la pestaña “Parameters”, clic en el símbolo (+) para crear un nuevo parámetro tipo “Float” y renómbrelo “Speed”.


Scene
Game
Asset Store
Animator
Layers
Parameters
Base Layer
Blend Tree
Name
Blend
0.0
Float
Int
Bool
Trigger

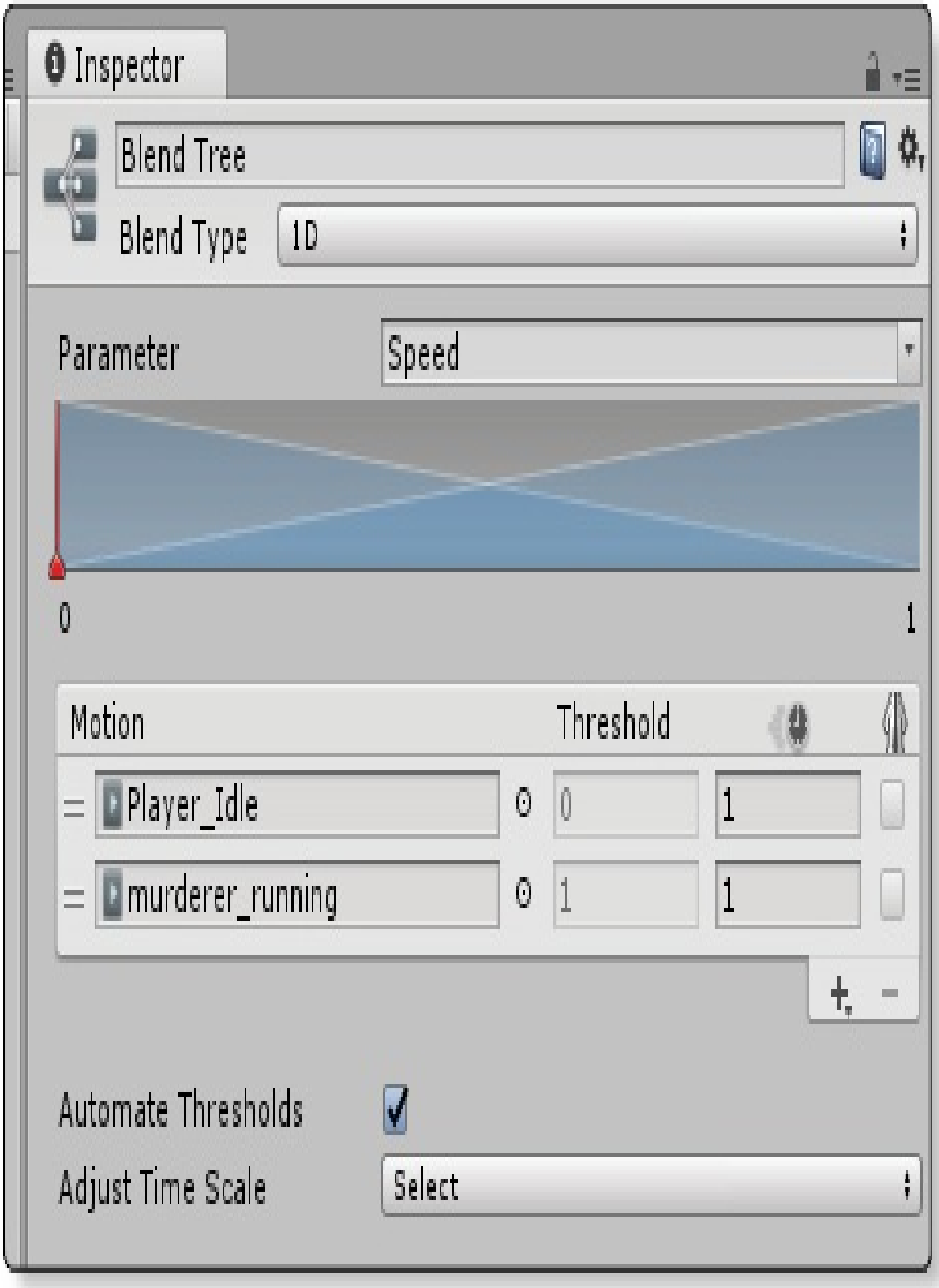
Creación de un parámetro tipo Float

Dentro de los parámetros de “Animator” podemos crear float, int, bool o Trigger ya que los vamos a controlar mediante la programación.


Scene
Game
Asset Store
Layers
Parameters
Name
Blend
0.0
Speed
0.0

Parámetro tipo Float creado

Seleccione “Blend Tree” y en “Inspector” asigne el nuevo parámetro “Speed”.


Inspector
Blend Tree
Blend Type
1D
Parameter
Speed
0
1
Motion
Threshold
Player_Idle
0
1
murderer_running
1
1
Automate Thresholds
Adjust Time Scale
Select

Asignación de parámetro

NOTA:

Estando dentro del modo de edición del “Blend Tree” y deseamos devolvernos al modo principal de edición de “Animator” tiene que dar doble clic en cualquier parte vacía.

Vaya al script “CharacterMovement”, haga la siguiente variable e inicialícela.

```
public float maxSpeed = 6.0f;
public float moveDirection;
public bool facingRight = true;
private Rigidbody rigidbody;
private Animator anim;

// Use this for initialization
void Start () {
    rigidbody = GetComponent<Rigidbody> ();
    anim = GetComponent<Animator>();
}
```

Creación e inicialización de variables

Vaya a la función “FixedUpdate” y añada el siguiente código para activar el parámetro “Speed” del “Blend Tree”.

```
void FixedUpdate (){
    rigidbody.velocity = new Vector2 (moveDirection * maxSpeed

    if (moveDirection > 0.0f && !facingRight){
        Flip ();
    } else if (moveDirection < 0.0f && facingRight) {
        Flip ();
    }
    anim.SetFloat ("Speed", Mathf.Abs (moveDirection));

}
```

Programación para correr la animación

En esta parte del código le estamos diciendo fuera de la estructura condicional que se fijará el valor de la variable “Speed” a su valor matemático absoluto en función a la variable “moveDirection”

Salve el script

Previsualice el juego para observar las animaciones de “Idle” y de “Running” cuando el personaje se encuentra parado o en movimiento relativamente.

Player corriendo hacia la derecha

Player corriendo hacia la izquierda

PROGRAMACIÓN DE LA CÁMARA

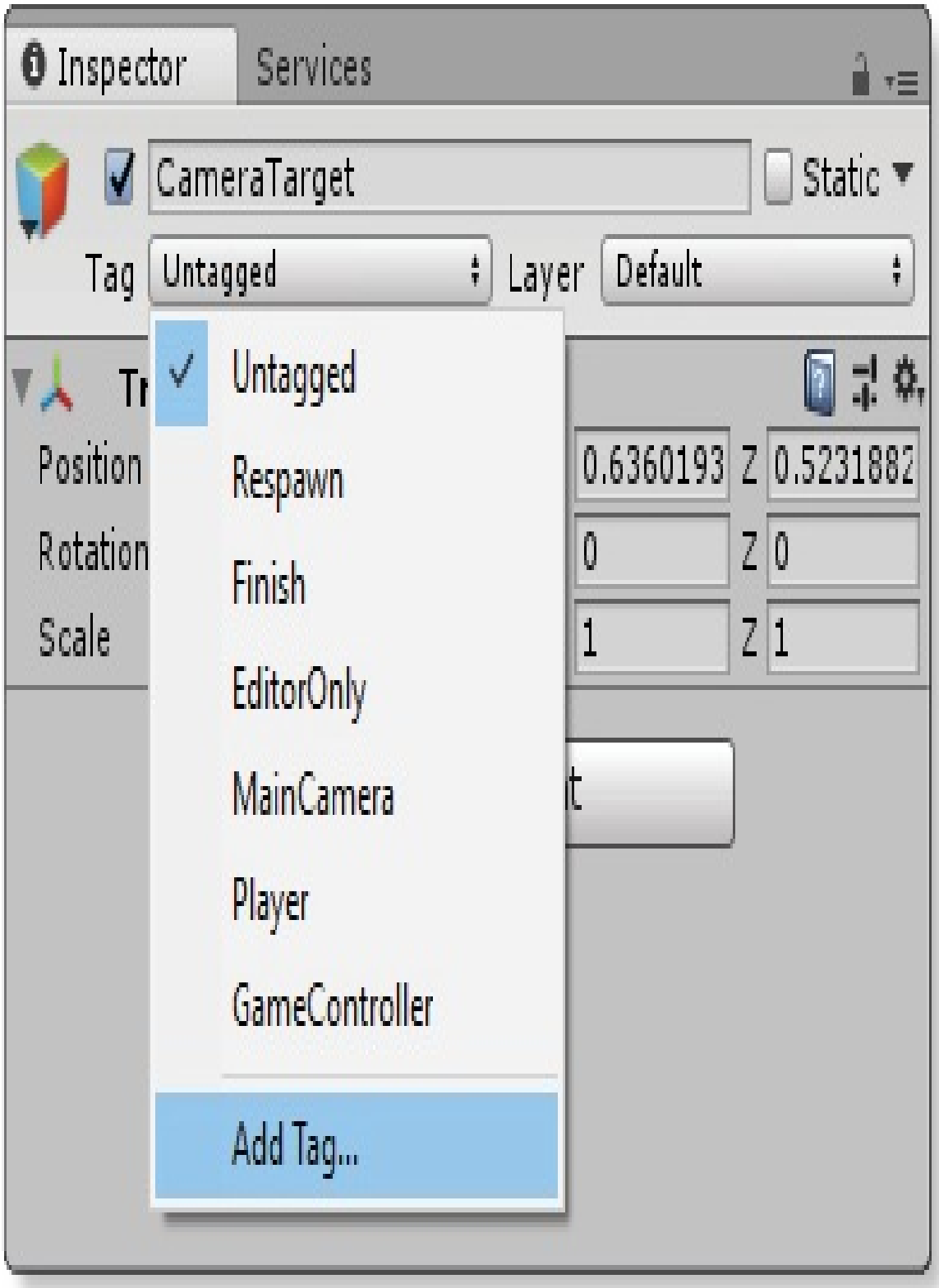
En este apartado programaremos el movimiento de la cámara con un movimiento suave para que siga los movimientos del personaje.

Nos introduciremos a conceptos de “Tags” y “Labels” para objetos, veremos cómo se pueden encapsular bloques de código para crear un sistema programable de ajuste de márgenes para que los obedezca la cámara.

Vaya al folder de “Scripts”, haga un nuevo script C# y nómbrelo “SmoothFollow”

Haga un nuevo “Empty GameObject”, renómbrelo “CameraTarget”

Seleccione el elemento, vaya a “Inspector”, clic en la pestaña “Tag” y clic en la opción “Add Tag…”


Inspector
Services
CameraTarget
Static
Tag
Untagged
Layer
Default
Position
Rotation
Scale
0.6360193
Z 0.5231882
0
Z 0
1
Z 1
Untagged
Respawn
Finish
EditorOnly
MainCamera
Player
GameController
Add Tag...

Creación de un nuevo Tag

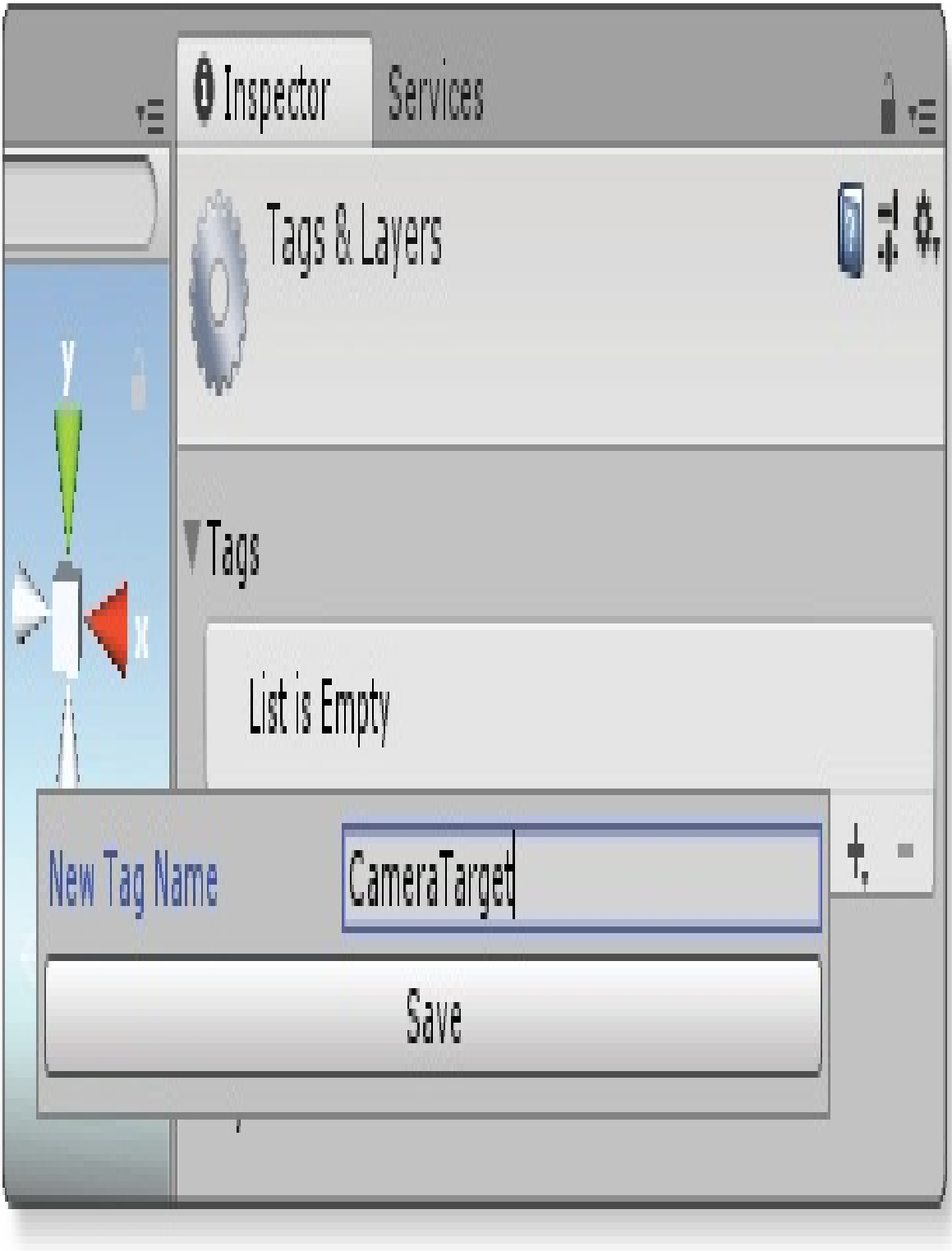
Los “Tags” nos sirven para aplicar una reserva de nombre a uno o a más objetos para poderlos identificar en el scripting.

Un proyecto por defecto ya tiene existentes los “Tags” que se muestran, pero para crear uno nuevo debemos de crearlo de la manera que se muestra.

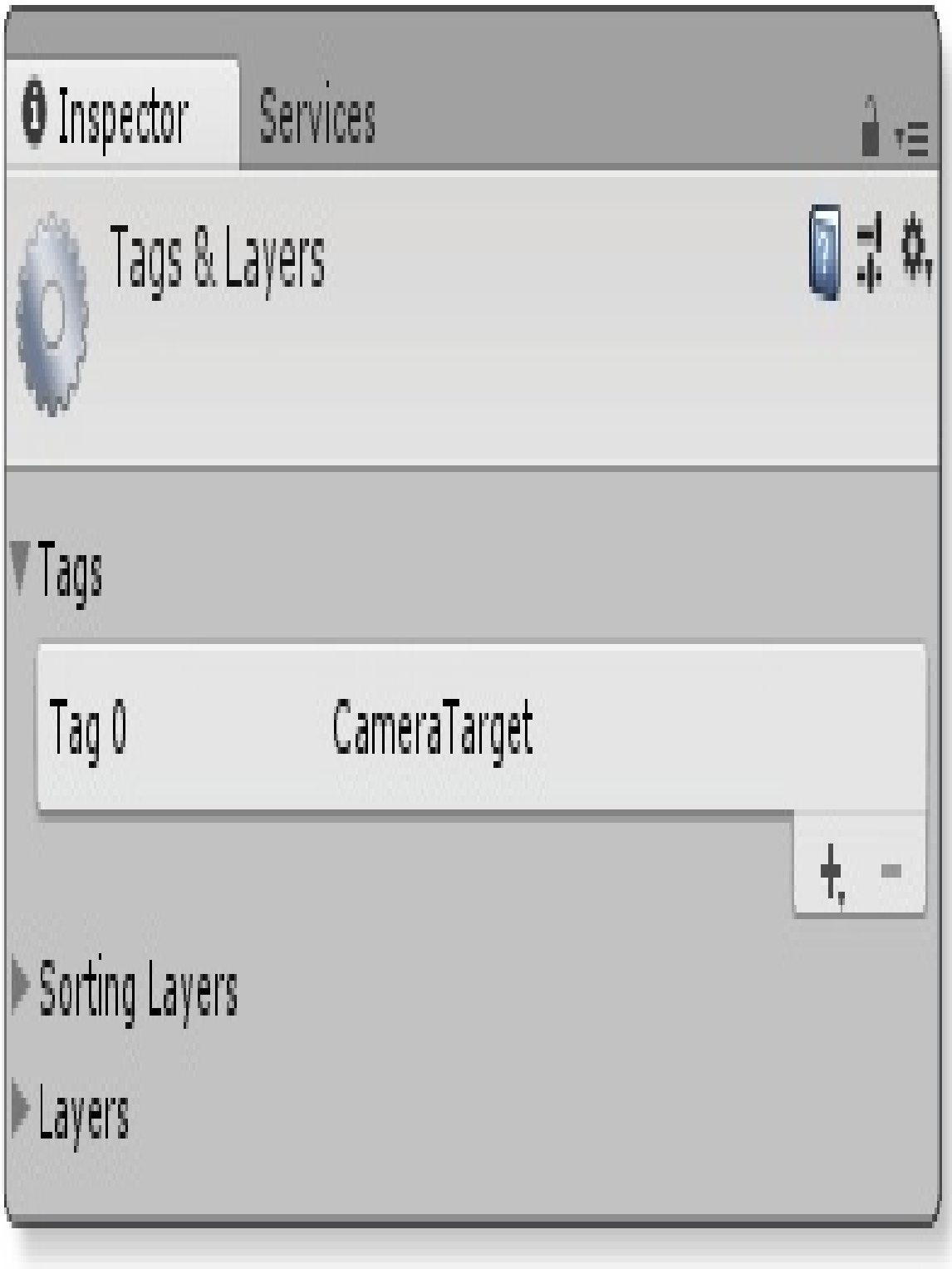
NOTA:

Al crear el nuevo “Tag” no se aplica al objeto, se debe de realizar en una acción siguiente

En la ventana de “Tags & Layers”, clic en el signo de (+) para crear un nuevo “Tag” y nómbrelo “CameraTarget”.

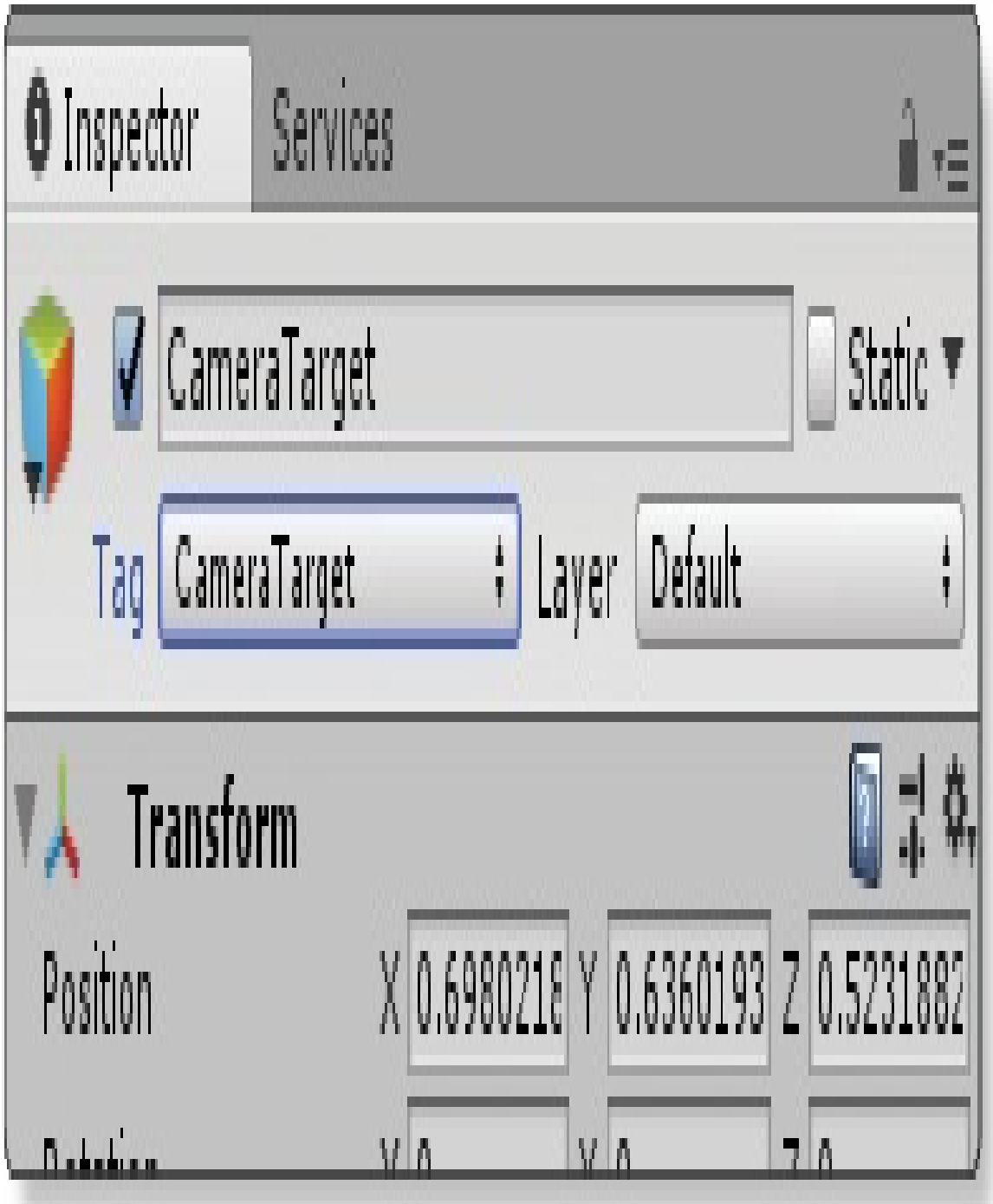

Inspector
Services
Tags & Layers
Tags
List is Empty
New Tag Name
CameraTarget
Save
y
x

Nombrar el nuevo Tag


Inspector
Services
Tags & Layers
Tags
Tag 0
CameraTarget
Sorting Layers
Layers

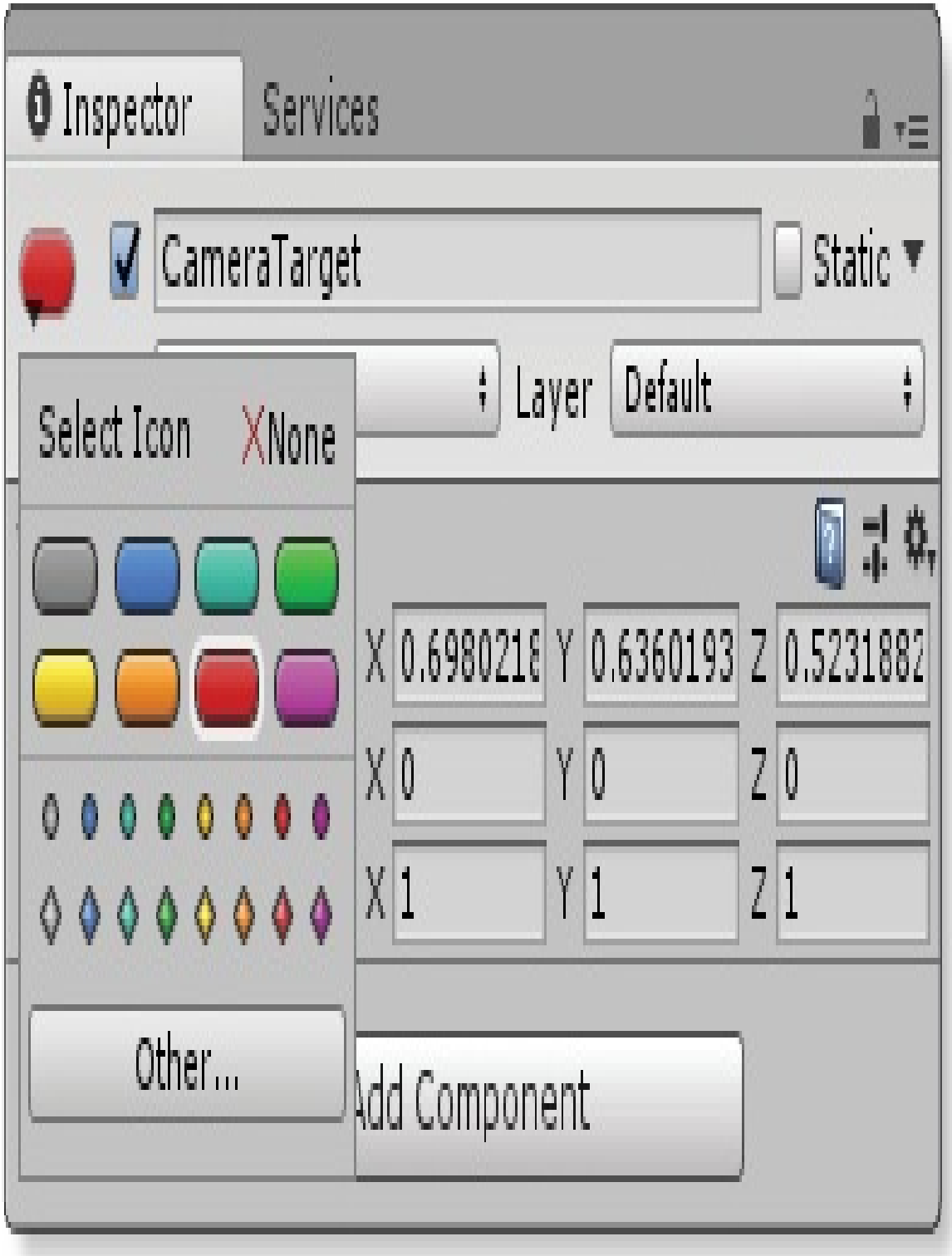
Nuevo Tag creado

Seleccione el objeto “CameraTarget” y asigne el “Tag” creado.


Inspector
Services
CameraTarget
Static
Tag
CameraTarget
Layer
Default
Transform
Position
X 0.6980216
Y 0.6360193
Z 0.5231882

Nuevo Tag asignado

Asigne un “Label” como se muestra.


Inspector
Services
CameraTarget
Static
Layer
Default
Select Icon
None
X 0.6980218
Y 0.6360193
Z 0.5231882
X 0
Y 0
Z 0
X 1
Y 1
Z 1
Other...
Add Component

Creación de un nuevo label

CameraTarget

Nuevo label para “CameraTarget”

Los “Labels” o etiquetas nos permiten visualizar en la ventana de “Scene” objetos que no contienen ningún elemento físico, y nos lo muestran con el color de la etiqueta que hayamos elegido.

Seleccione la vista 2D en la parte superior izquierda de la ventana “Scene” y posicione el elemento “CameraTarget” donde se muestra.

Scene
Game
Asset Store
Animator
Shaded
2D
CameraTarget

Posicionamiento en 2D de CameraTarget

Empareje “CameraTarget” con “Player”

Abra el script “SmoothFollow” y haga las siguientes variables

```
public class SmoothFollow : MonoBehaviour {

    public float xMargin = 1.0f;
    public float yMargin = 1.0f;

    public float xSmooth = 10.0f;
    public float ySmooth = 10.0f;

    public Vector2 maxXandY;
    public Vector2 minXandY;

    public Transform cameraTarget;

    // Use this for initialization
    void Start () {
```

Variables para SmoothFollow

Haga la siguiente función “Awake” para inicializar la variable

```
public Vector2 maxXandY;
public Vector2 minXandY;


public Transform cameraTarget;


void Awake (){
    cameraTarget = GameObject.FindGameObjectWithTag ("CameraTarget").transform;
}


// Use this for initialization
void Start () {
```

Función Awake

La función “Awake” en diferencia con la función “Start” es que ésta ocurre antes de la función “Start”.

Al momento de estar inicializando la variable “cameraTarget” estamos buscando un objeto dentro del juego con el tag “CameraTarget” que previamente creamos y asignamos. Al objeto con dicho tag estamos inicializando la variable conforme a su componente “Transform” para poderlo mover.

NOTA:

En seguida crearemos un sistema para la movilización de la cámara. Si esta es tu primera vez en Unity y en Scripting podrán ser un poco confusos, pero no importa si en esta etapa de tu aprendizaje no los comprendes al 100%, los puedes utilizar como bloques de código. Conforme vayamos avanzando en el curso y aprendiendo más ecuaciones los vas a poder ir interpretando con mayor fluidez.

Escriba las siguientes funciones booleanas para evaluar si los valores de los márgenes son verdaderos o falsos conforme a su posición en X y Y

```
void Awake (){
    cameraTarget = GameObject.FindGameObjectWithTag ("CameraTarget").transform;
}


bool CheckXMargin (){
    return Mathf.Abs (transform.position.x - cameraTarget.position.x) > xMargin;
}

bool CheckYMargin (){
    return Mathf.Abs (transform.position.y - cameraTarget.position.y) > yMargin;
}

// Use this for initialization
void Start () {
```

Funciones booleanas

Las dos funciones son tipo booleanas, es decir que nos van a arrojar un resultado verdadero o falso y nos van a regresar un valor matemático absoluto (Math.Abs).

Escriba la siguiente función para posicionar la cámara conforme a la posición del objeto “CameraTarget”

```
    // Update is called once per frame
    void Update () {

    }

    void TrackPlayer (){
        float targetX = transform.position.x;
        float targetY = transform.position.y;

        if (CheckXMargin ()) {
            targetX = Mathf.Lerp (transform.position.x, cameraTarget.position.x, xSmooth * Time.deltaTime);
        }

        if (CheckYMargin ()) {
            targetY = Mathf.Lerp (transform.position.y, cameraTarget.position.y, ySmooth * Time.deltaTime);
        }

        targetX = Mathf.Clamp (targetX, minXandY.x, maxXandY.x);
        targetY = Mathf.Clamp (targetY, minXandY.y, maxXandY.y);

        transform.position = new Vector3 (targetX, targetY, transform.position.z);
    }
}
```

Función TrackPlayer

Dentro de las funciones también se pueden declarar variables como en este caso “targetX” y “targetY” mismas que a su vez se les está dando un valor de posición en X y Y relativamente.

El comando Mathf.Lerp lo utilizamos cuando queremos crear una transición suave (smooth).

Time.deltaTime lo utilizamos cuando queremos tomar el tiempo real de nuestro videojuego.

El comando Mathf.Clamp lo utilizamos cuando queremos establecer un valor que no exceda los límites mínimos y máximos (en este caso los valores de “minXandY” y “maxXandY”.

Haga la siguiente función “FixedUpdate” y mande llamar la función creada.

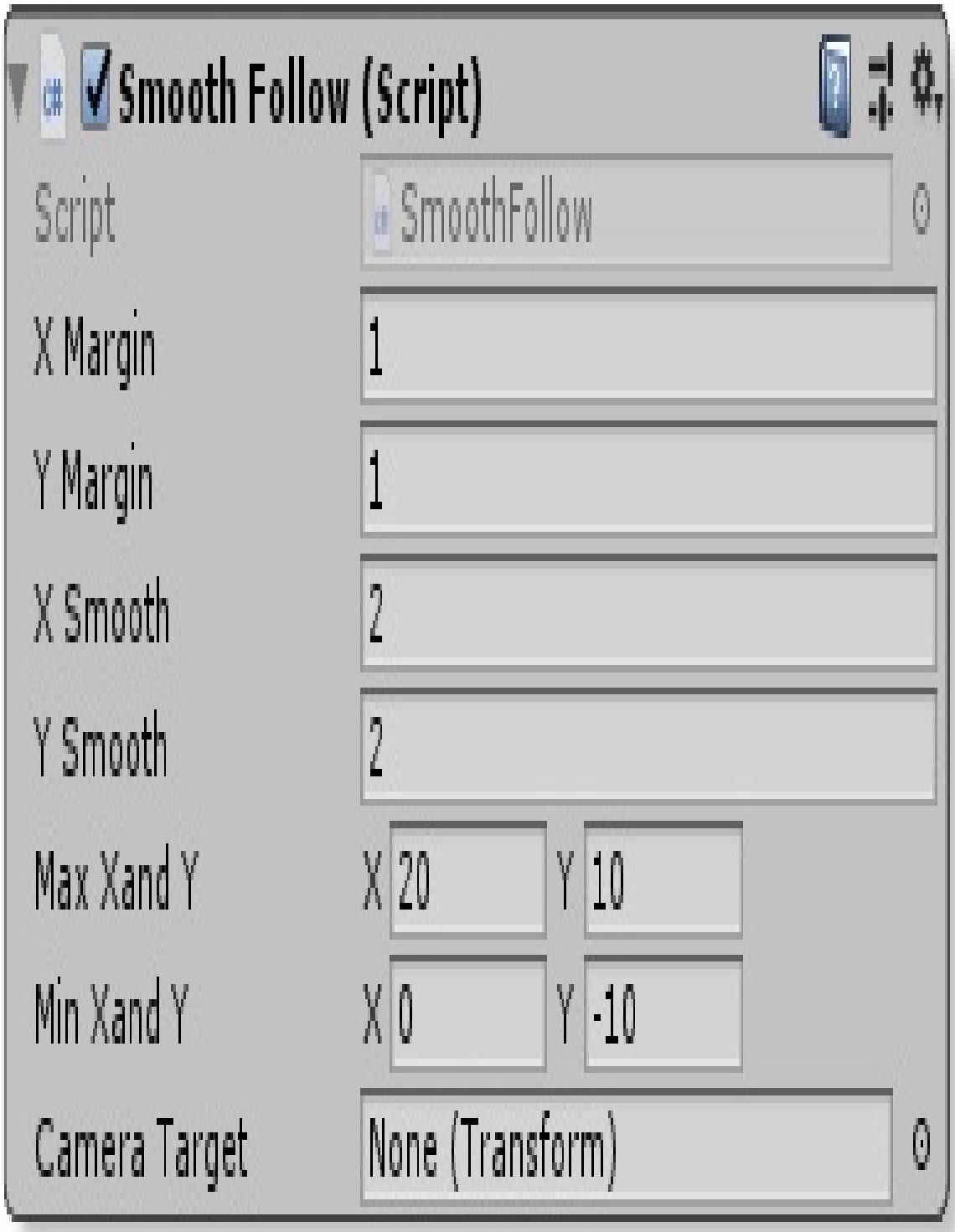
```
// Update is called once per frame
void Update () {

}

void FixedUpdate () {
    TrackPlayer ();
}

void TrackPlayer (){
    float targetX = transform.position.x;
    float targetY = transform.position.y;
```

Llamar TrackPlayer

Salve el script

Vaya a Unity, asigne el script “SmoothFollow” y asígnelo al objeto “Main Camera”

Asigne los valores que se muestran al script.


Smooth Follow (Script)
Script
SmoothFollow
X Margin
1
Y Margin
1
X Smooth
2
Y Smooth
2
Max Xand Y
X 20
Y 10
Min Xand Y
X 0
Y -10
Camera Target
None (Transform)

Script en Unity SmoothFollow

Testing

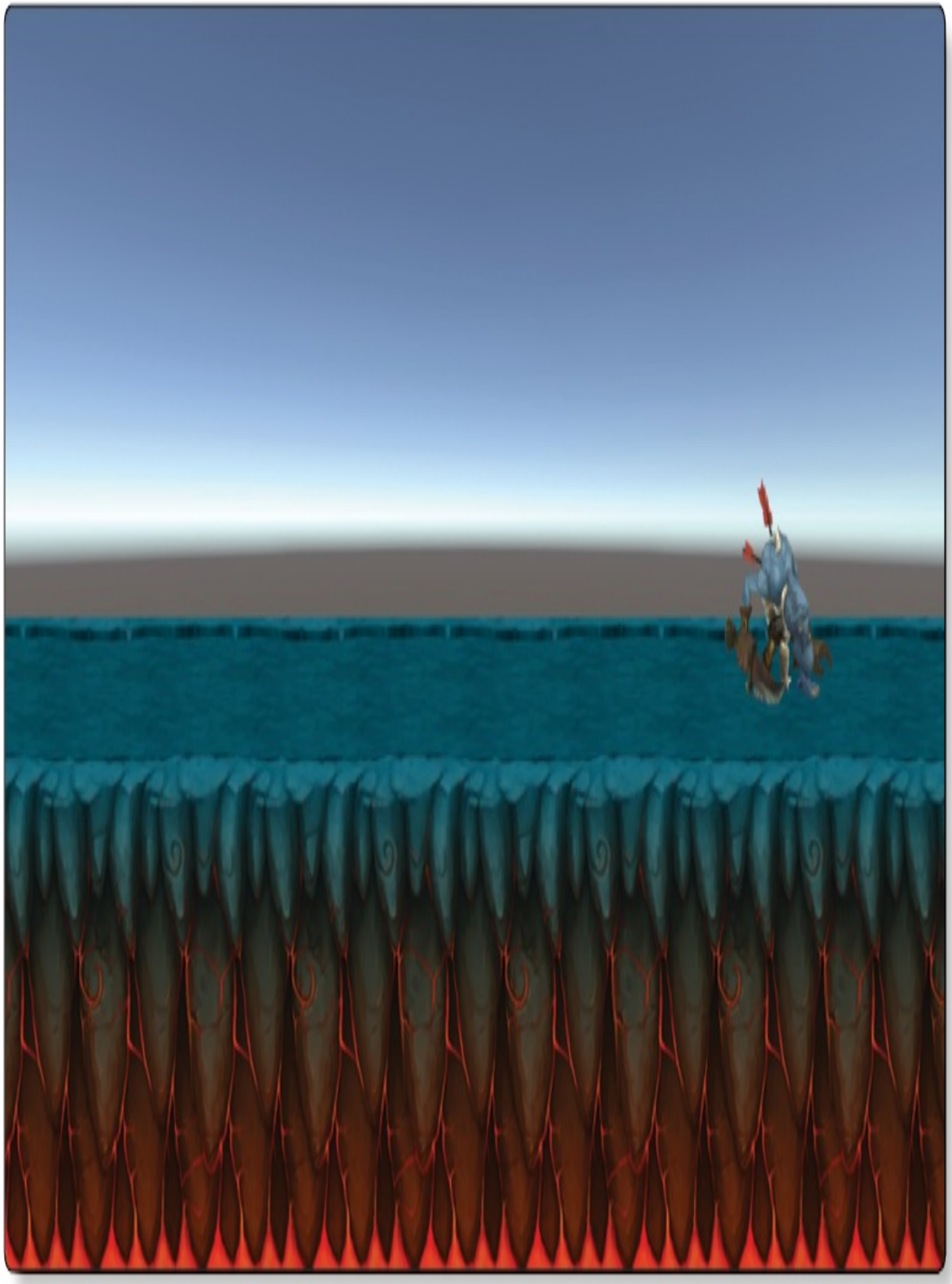
Cuando queremos hacer testing de valores a tiempo real debemos de tomar en cuenta que los cambios hechos en los valores de los componentes o de los scripts que hagamos en el modo “Play”, automáticamente vuelven a su original una vez que hayamos pulsado “Stop” y regresemos al modo de edición.

Para hacer “testing” de los valores de los márgenes en este ejercicio siga los siguientes pasos:

Clic en Play

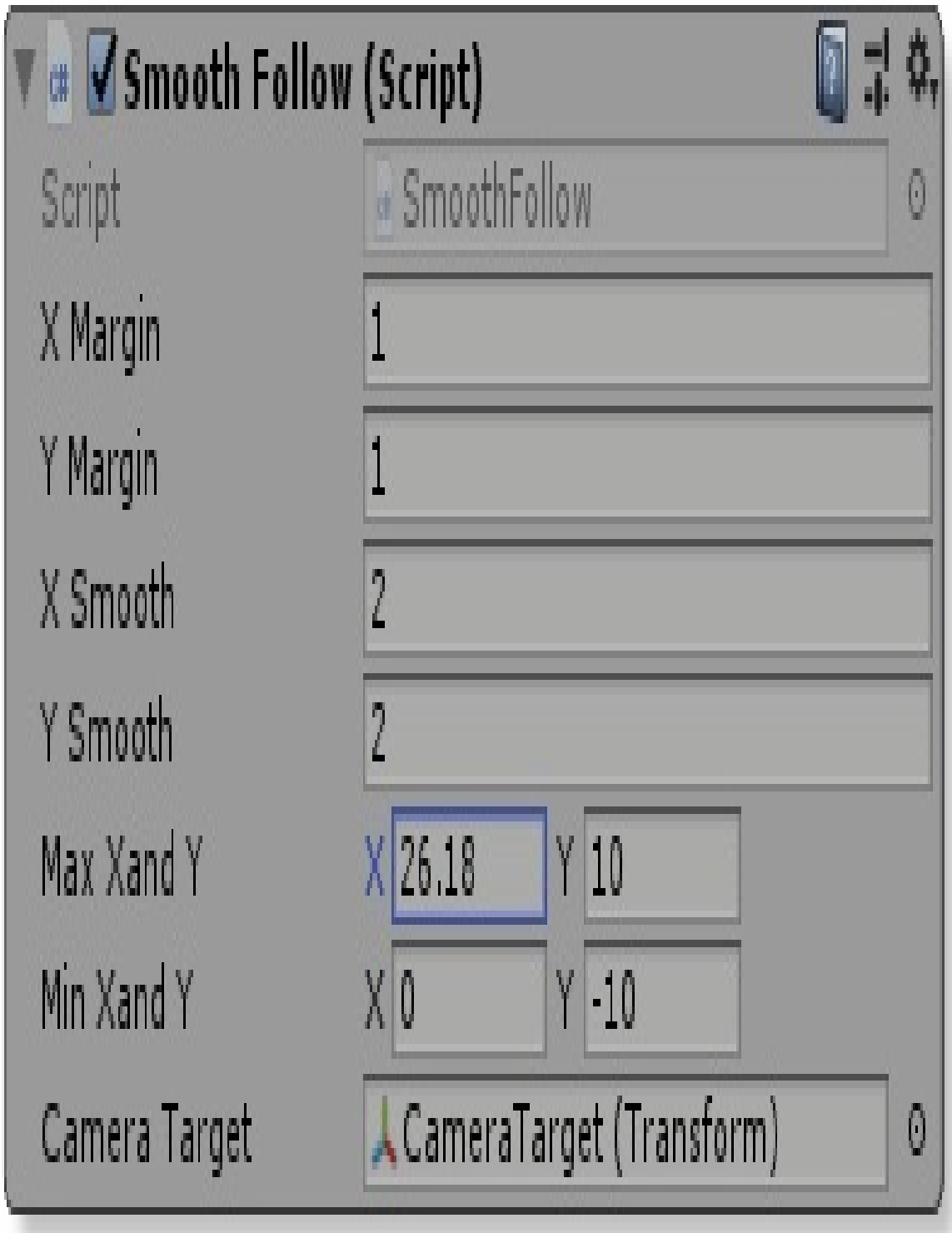
Mueva al “Player” para observar que la cámara lo sigue

Mueva el personaje hacia la derecha hasta que se detenga la cámara (el margen máximo de X).

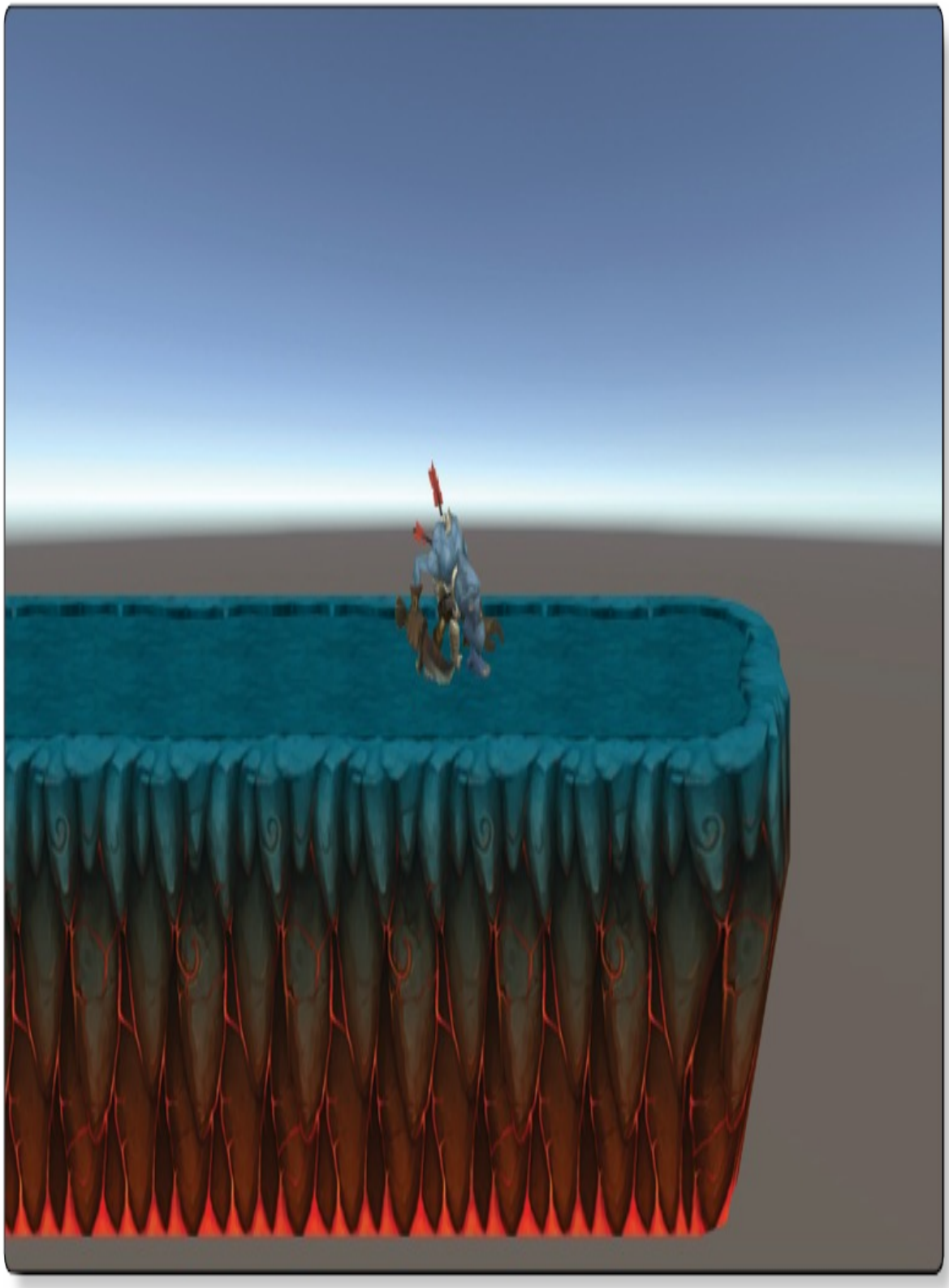


Límite de margen X

Modifique el valor máximo de X en el script.


Smooth Follow (Script)
Script
SmoothFollow
X Margin
1
Y Margin
1
X Smooth
2
Y Smooth
2
Max Xand Y
X 26.18
Y 10
Min Xand Y
X 0
Y -10
Camera Target
CameraTarget (Transform)

Expansión de margen X



Margen X expandido

Tome nota del nuevo valor ya que al finalizar su previsualización regresara a su estado anterior (20)

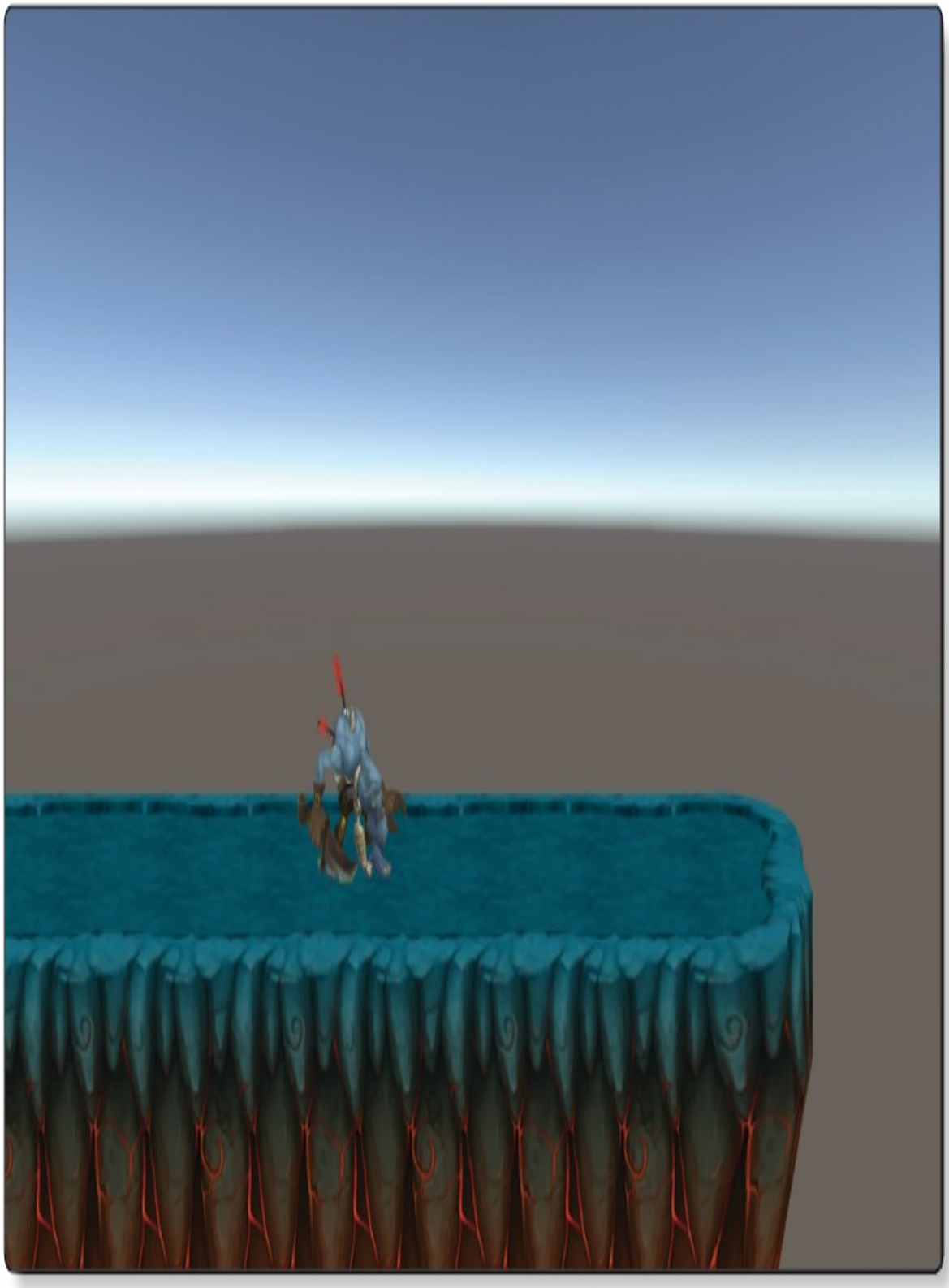
Detenga la previsualización y asigne el nuevo valor máximo de X (26.18)

Previsualice nuevamente el juego para observar los cambios

Repita el procedimiento para Y.

Smooth Follow (Script)
Script
SmoothFollow
X Margin
1
Y Margin
1
X Smooth
2
Y Smooth
2
Max Xand Y
X 26.18
Y 10
Min Xand Y
X 0
Y 3.45
Camera Target
CameraTarget (Transform)

Expansión de margen Y

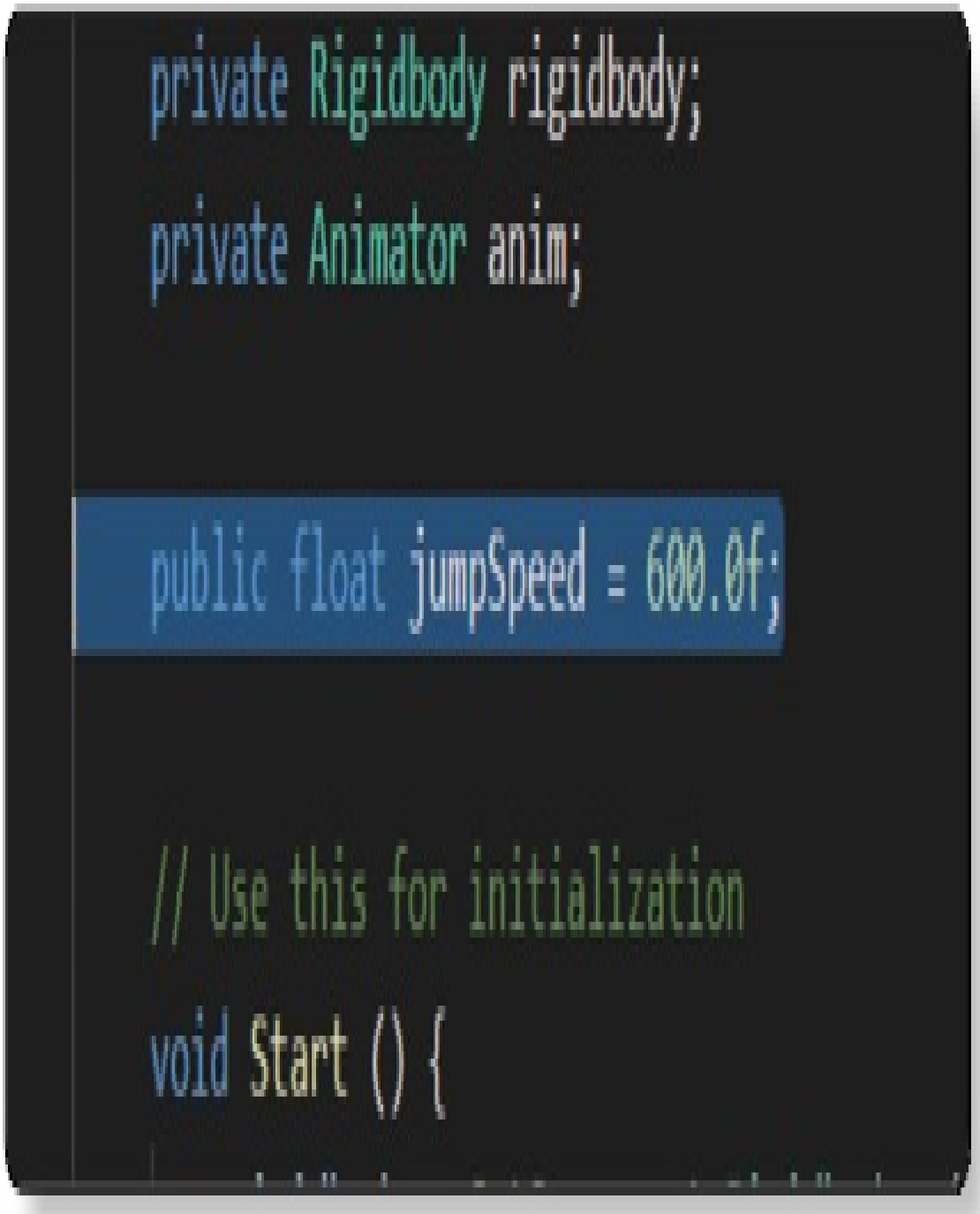


Margen Y expandido

PROGRAMACIÓN DE SALTO DEL PLAYER

En este apartado programaremos el salto del personaje conforme a aplicarle una fuerza y a la programación de un chequeo que nos permita saber si nuestro personaje se encuentra en el suelo o no.

Vaya al script “CharacterMovement” y haga la siguiente variable.


private Rigidbody rigidbody;
private Animator anim;
public float jumpSpeed = 600.0f;
// Use this for initialization
void Start () {

Nueva variable

En la función “Update” escriba el siguiente código para programar el brinco con la barra espaciadora.

```
// Update is called once per frame
void Update () {
    moveDirection = Input.GetAxis("Horizontal");


    if (Input.GetButtonDown ("Jump")){
        rigidbody.AddForce (new Vector2 (0, jumpSpeed));
    }

}
```

Código para el salto

En este código estamos indicando que cuando oprimamos el botón preestablecido con la función “Jump” que en este caso sería nuestra barra espaciadora, entonces aplicarle una fuerza a nuestro “rigidbody” en dos vectores (X y Y) las cuales serían de cero en X y de la variable “jumpSpeed” en Y.

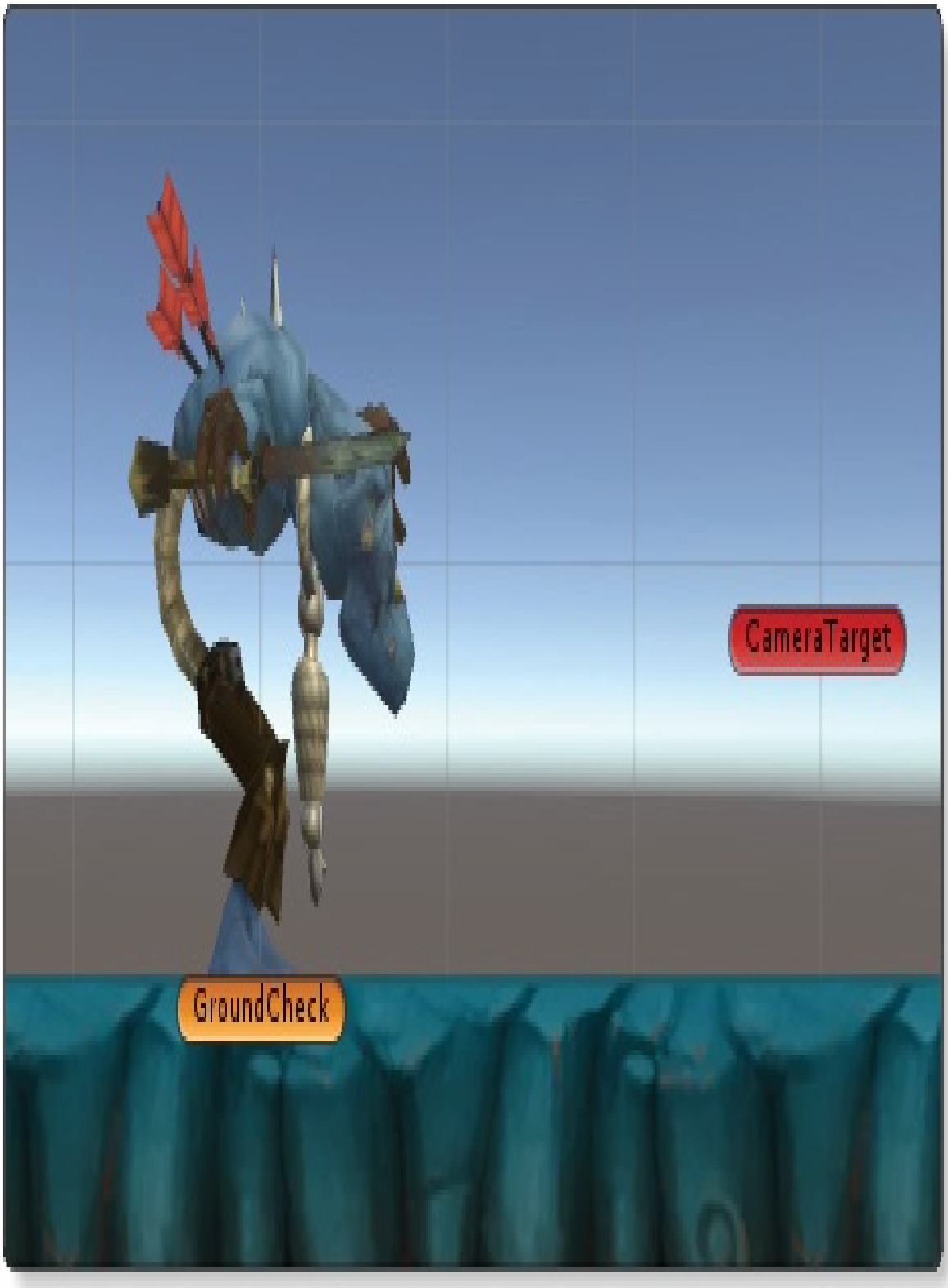
Salve el script

Previsualice el juego para observar que el “Player” brinca.

Player brincando

PROBLEMAS: El “Player” aunque siga en el aire puede volver a brincar sin límite

SOLUCIÓN: Haga un nuevo “EmptyGameObject”, renómbrelo “GroundCheck” asigne un Label naranja, emparéjelo con “Player” y colóquelo donde se muestra.


CameraTarget
GroundCheck

Objeto GroundCheck

Hierarchy
Create
All
206*
Main Camera
Directional Light
Platform
Player
zombie_murderer
CameraTarget
GroundCheck

Emparentamiento de GroundCheck

Vaya al script “CharacterMovement”, escriba las siguientes variables e inicialice.

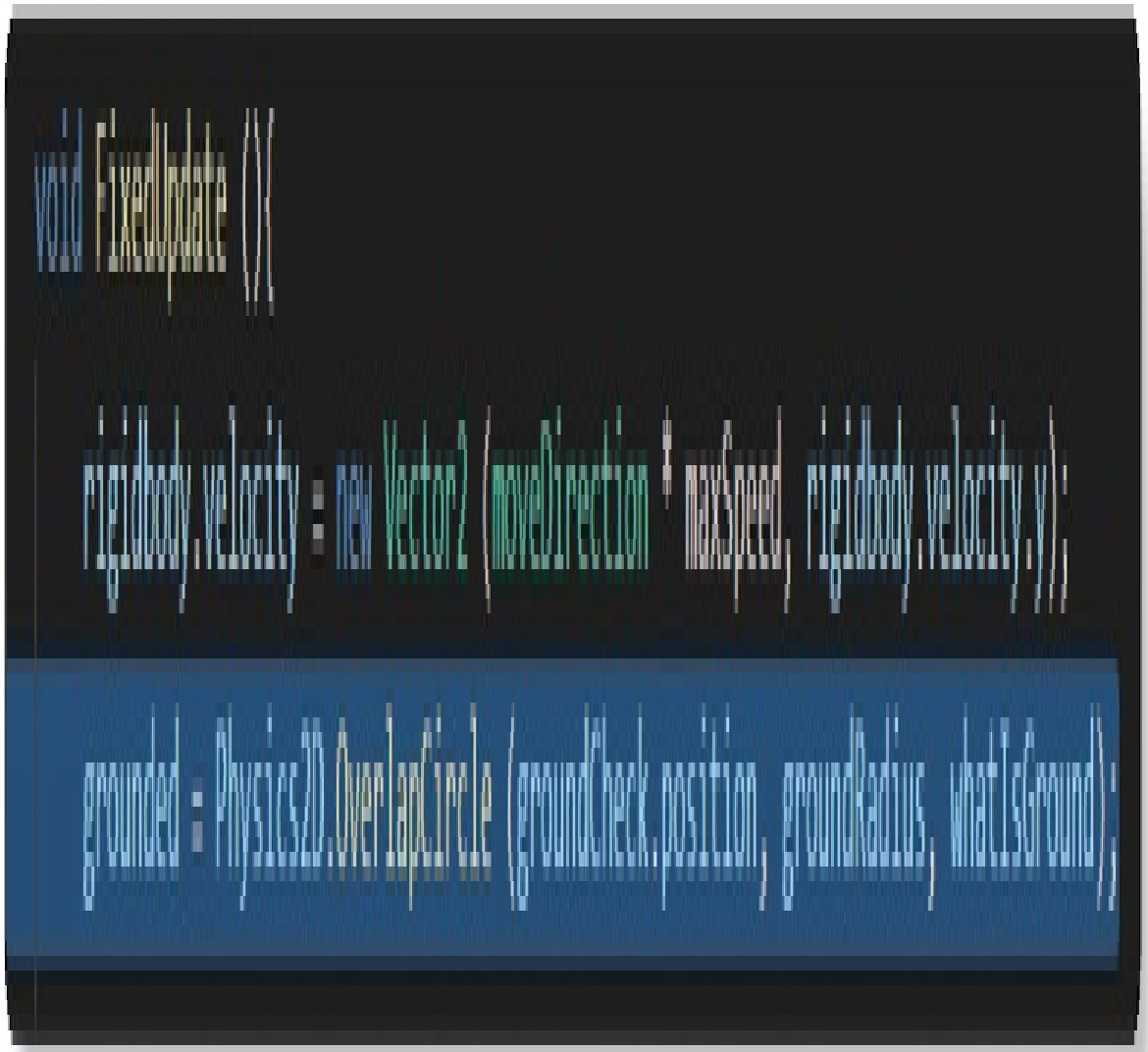
```
    private Animator anim;
    public float jumpSpeed = 600.0f;

    public bool grounded = false;
    public Transform groundCheck;
    public float groundRadius = 0.2f;
    public LayerMask whatIsGround;

    // Use this for initialization
    void Start () {
        rigidbody = GetComponent<Rigidbody> ();
        anim = GetComponent<Animator>();
        groundCheck = GameObject.Find ("GroundCheck").transform;
    }
```

Variables para el GroundCheck

Tenemos un nuevo tipo objeto de variable que es “LayerMask” y la utilizamos cuando queremos que nos afecte alguna parte o extensión de alguno objeto, en este caso específico será el suelo.

En la inicialización podemos observar otra forma para mandar llamar un objeto específico con el comando GameObject.Find para nombrar un objeto tal cual aparece en Hierarchy, en este caso específico “GroundCheck”.

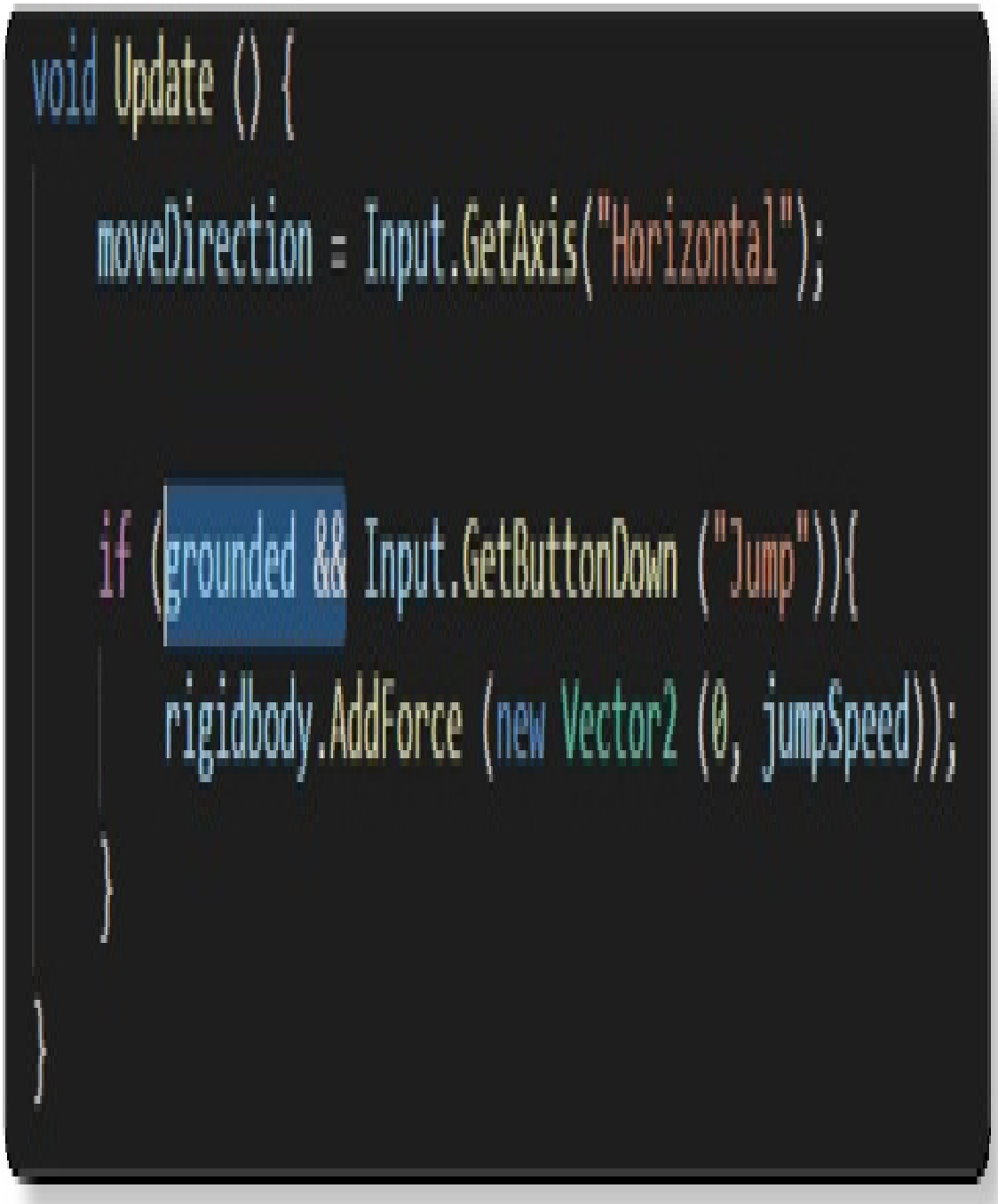
Añada la siguiente línea de código en la función “FixedUpdate” para asignar valor a la variable “grounded”.



Valor para “grounded”

En este código estamos asignando a la variable “grounded” un circulo alrededor de un objeto físico 2D (que todavía no asignamos en Unity) para que forme parte del check “whatIsGround”.

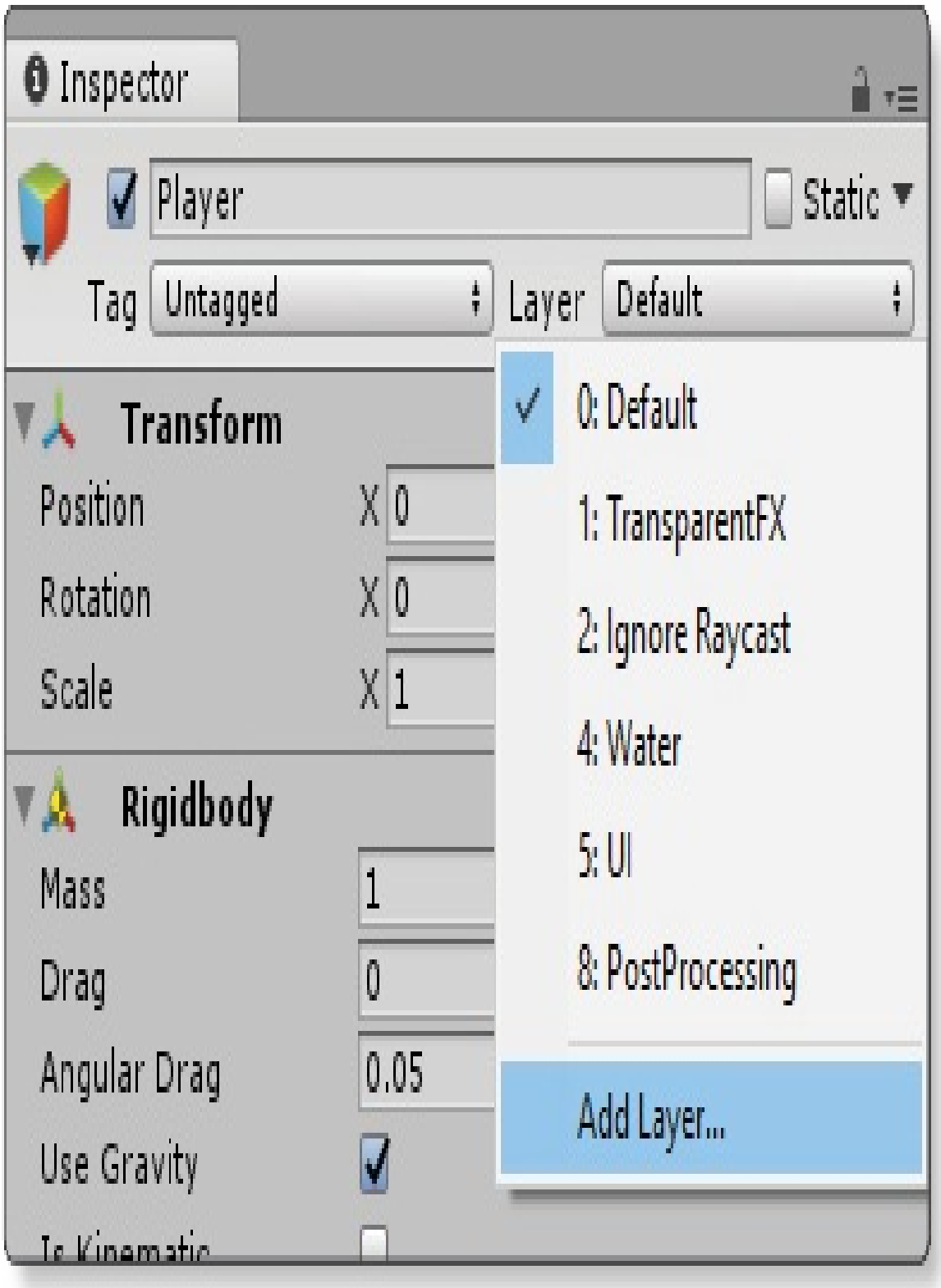
Modifique la estructura condicional de “Update”.


void Update () {
moveDirection = Input.GetAxis("Horizontal");
if (grounded && Input.GetButtonDown ("Jump")){
rigidbody.AddForce (new Vector2 (0, jumpSpeed));
}
}

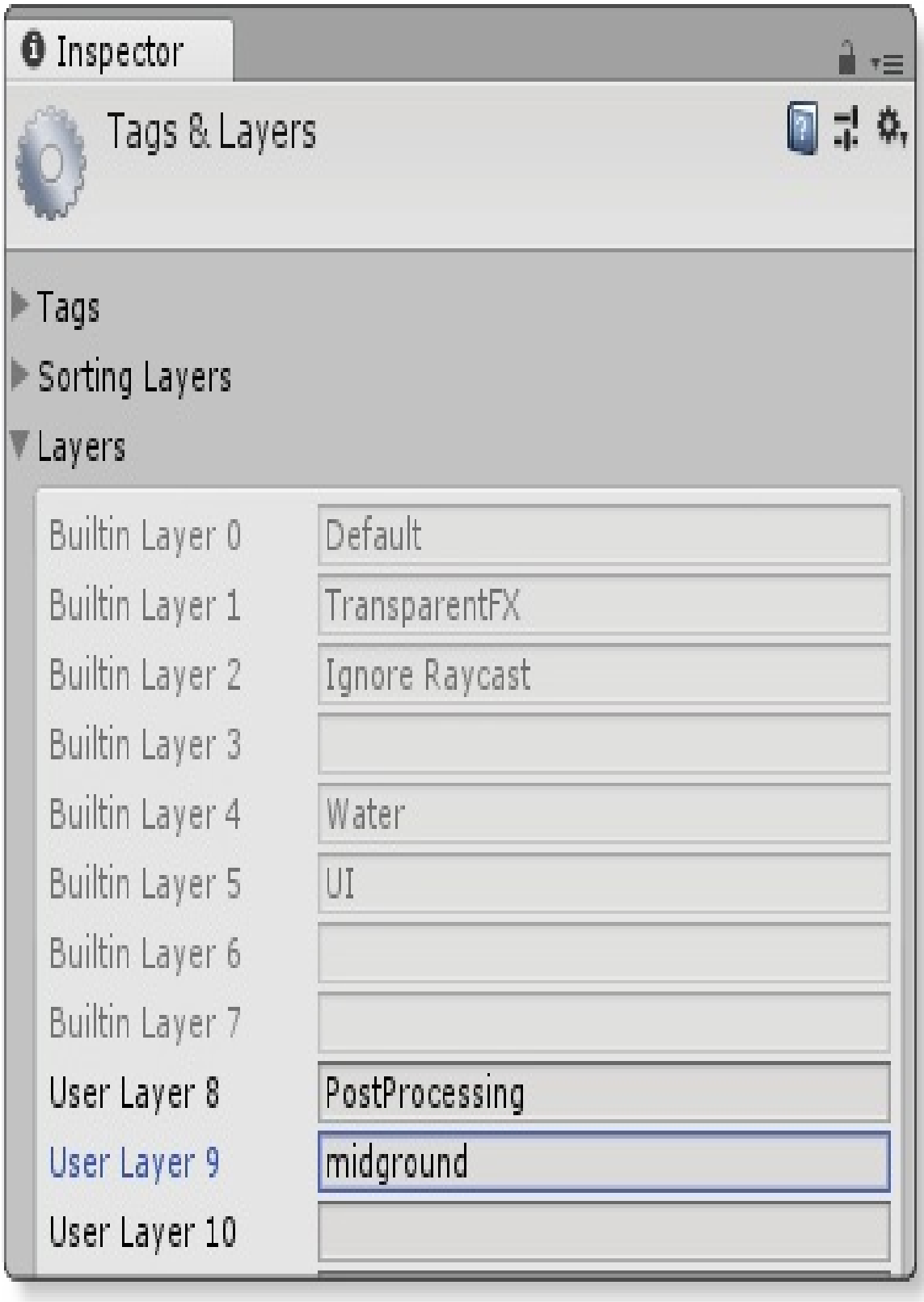
Condición “grounded”

Estamos añadiendo la condición “grounded” para que pueda realizar el salto.

Vaya a Unity y haga un nuevo layer (de la misma forma que se hace un nuevo “Tag”) y nómbrelo “midground”.


Inspector
Player
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Rotation
X 0
Scale
X 1
Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
0: Default
1: TransparentFX
2: Ignore Raycast
4: Water
5: UI
8: PostProcessing
Add Layer...

Creación de un nuevo layer

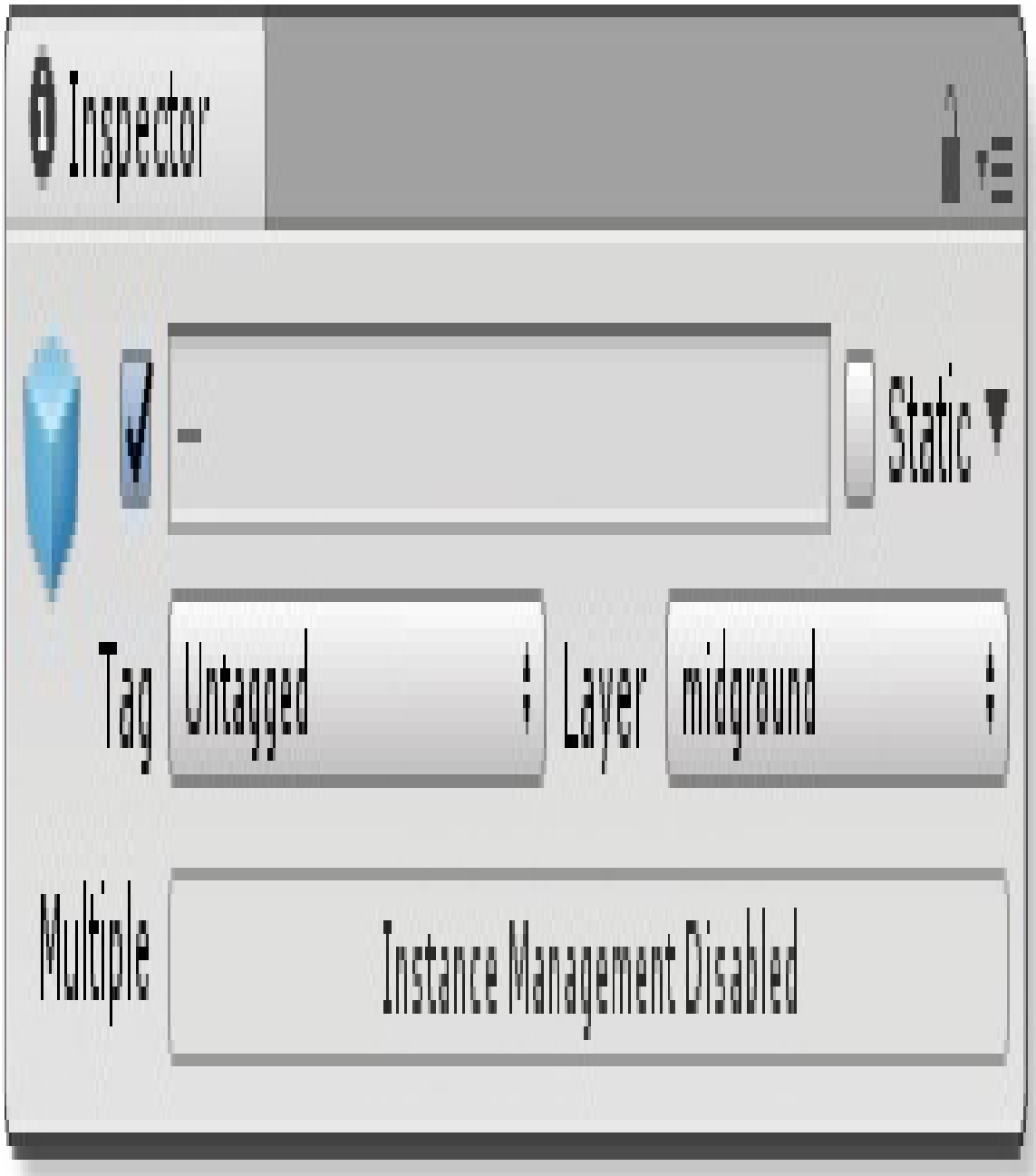

Inspector
Tags & Layers
Tags
Sorting Layers
Layers
Builtin Layer 0
Default
Builtin Layer 1
TransparentFX
Builtin Layer 2
Ignore Raycast
Builtin Layer 3
Builtin Layer 4
Water
Builtin Layer 5
UI
Builtin Layer 6
Builtin Layer 7
User Layer 8
PostProcessing
User Layer 9
midground
User Layer 10

Nuevo Layer creado

Seleccione los objetos hijos que conforman la plataforma y asigne el layer “midground”.

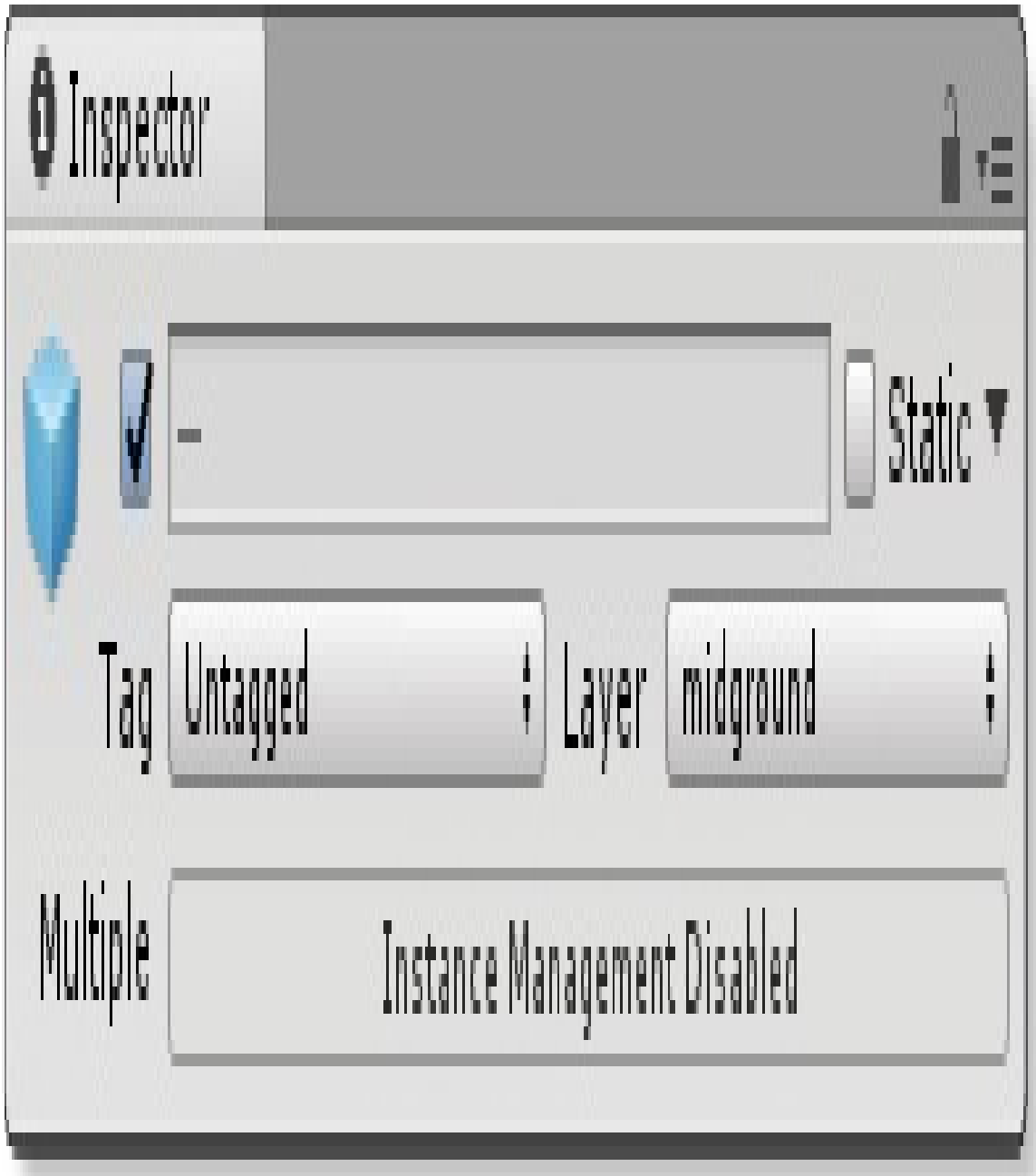
Hierarchy
Create
All
206*
Main Camera
Directional Light
Platform
module_tile_02_bd
module_tile_02_bd (1)
module_tile_02_bd (2)
module_tile_02_bd (3)
module_tile_02_bd (4)
module_tile_02_bd (5)
module_tile_02_bd (6)
module_tile_02_bd (7)
module_end_b
module_end_d
Player
zombie_murderer
CameraTarget
GroundCheck

Objetos hijos de Platform


Inspector
Static
Tag
Untagged
Layer
midground
Multiple
Instance Management Disabled

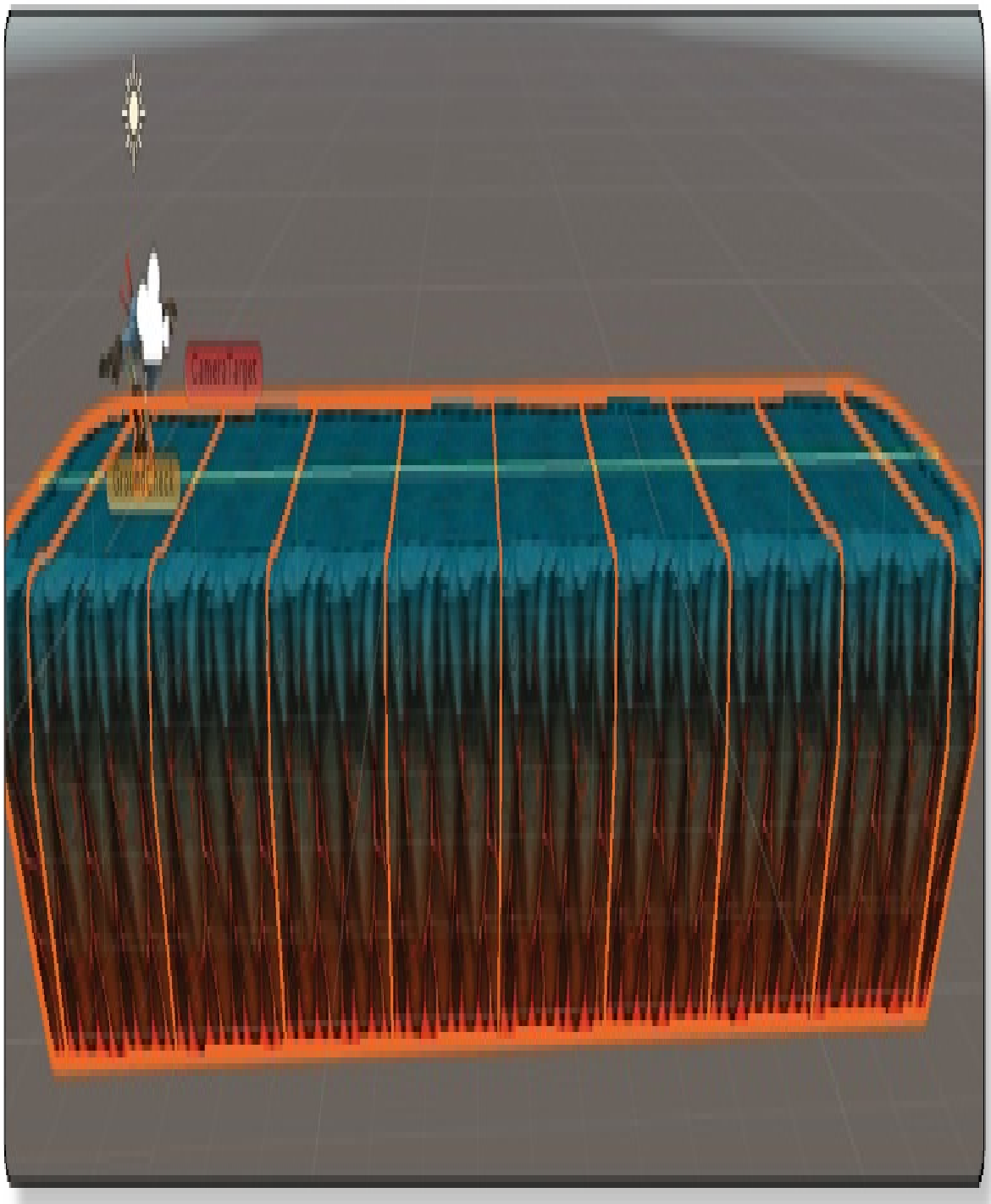
Asignación de Layer

Con los mismos elementos seleccionados asigne su componente “BoxCollider2D”


Inspector
Static
Tag
Untagged
Layer
midground
Multiple
Instance Management Disabled

Add Component
box
Search
Box Collider
Box Collider 2D
Skybox
New script

Asignación de un BoxCollider2D


CameraTarget

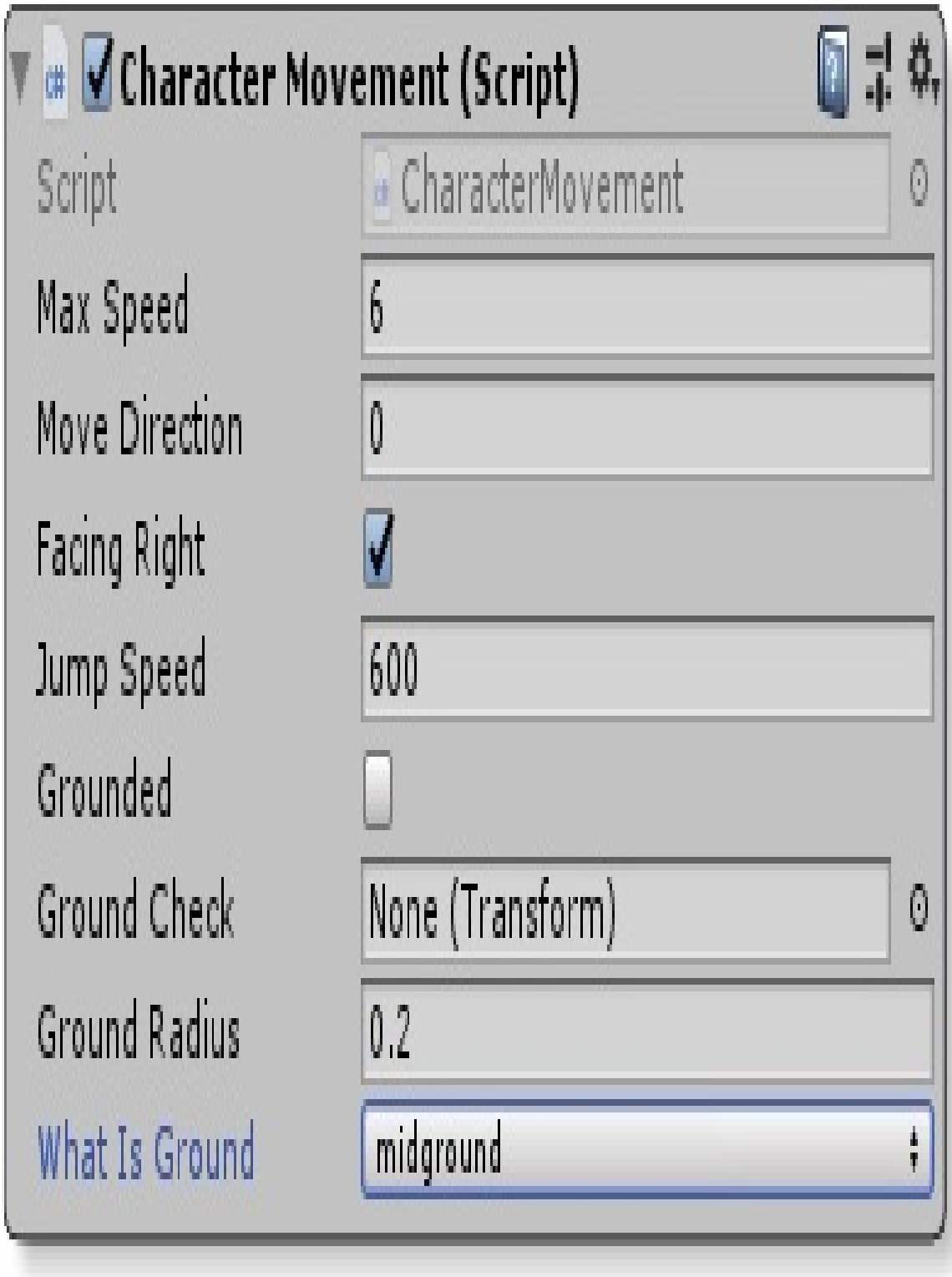
BoxCollider2D asignado

Al asignar el componente podemos observar una línea atravesando la mitad de la plataforma, ese es nuestro BoxCollider 2D mismo que inicializamos anteriormente para la variable “grounded”.

NOTA:

El BoxCollider2D únicamente funciona con objetos 3D que tienen 0,0,0 de valor de rotación y por eso no es conveniente rotar en Unity un objeto que va a funcionar como suelo en un videojuego de plataformas. En caso de necesitar rotarlo se recomienda hacerlo desde su programa origen.

Seleccione “Player” y en “Inspector” asigne el valor mostrado en “What Is Ground”.


Character Movement (Script)
Script
CharacterMovement
Max Speed
6
Move Direction
0
Facing Right
Jump Speed
600
Grounded
Ground Check
None (Transform)
Ground Radius
0.2
What Is Ground
midground

Asignación de layer al LayerMask

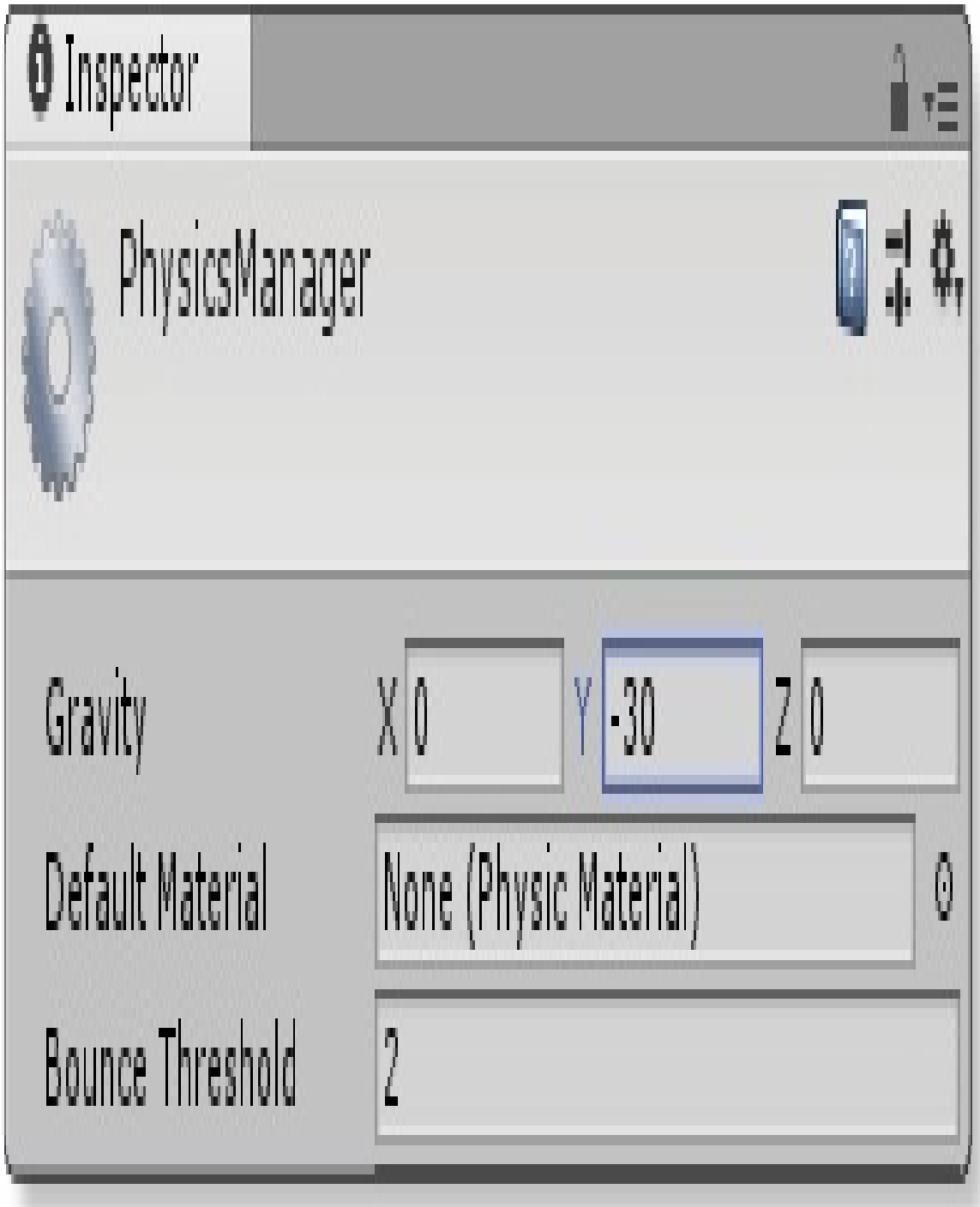
Previsualice el juego para observar que cuando el “Player” se encuentra en el suelo, la casilla de “Grounded” permanece marcada y cuando brinca se desactiva.

Facing Right
Jump Speed
600
Grounded
Ground Check
GroundCheck
Ground Radius
0.2
What Is Ground
midground

Grounded activado

PROBLEMA: El personaje salta muy alto.

SOLUCIÓN: En el menú general de Unity vaya a “Edit/ Project Settings/ Physics” y cambie la gravedad a -30 (la gravedad simulada a la realidad es de -9.81).


Inspector
PhysicsManager
Gravity
X 0
Y -30
Z 0
Default Material
None (Physic Material)
Bounce Threshold
2

Cambio de gravedad

Cambie el “jumpSpeed” del “Player” a 800 en el script

Previsualice el juego para observar que el “Player” salta correctamente.

Player brincando correctamente

PROGRAMACIÓN DE LA ANIMACIÓN DEL SALTO

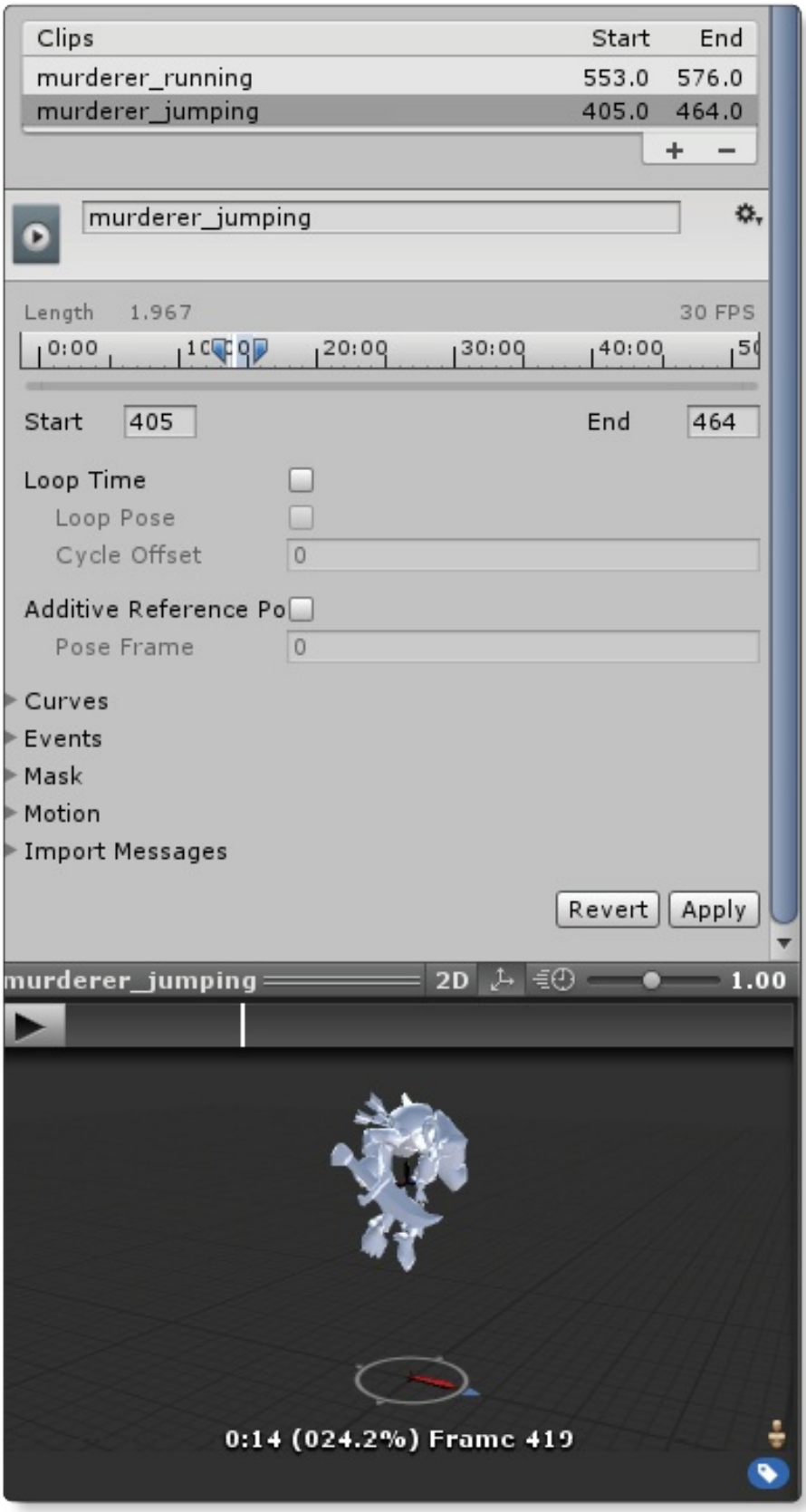
En este apartado nos ocuparemos de la animación del salto del personaje, así como de resolver algunos problemas visuales que nos podrán ocurrir al trabajar con animaciones hechas externamente.

En la ventana “Proyect” vaya al folder de “PlayerAnimations” y seleccione “murderer_all_in_one”

En la pestaña de “Animations” de Inspector haga clic en el símbolo de (+) que se muestra y haga la nueva animación respectiva del salto, tomando en cuenta que los cuadros correspondientes son:

frame: 0405-0464 jump.

Y renómbrela como “murderer_jumping” y clic en Apply.

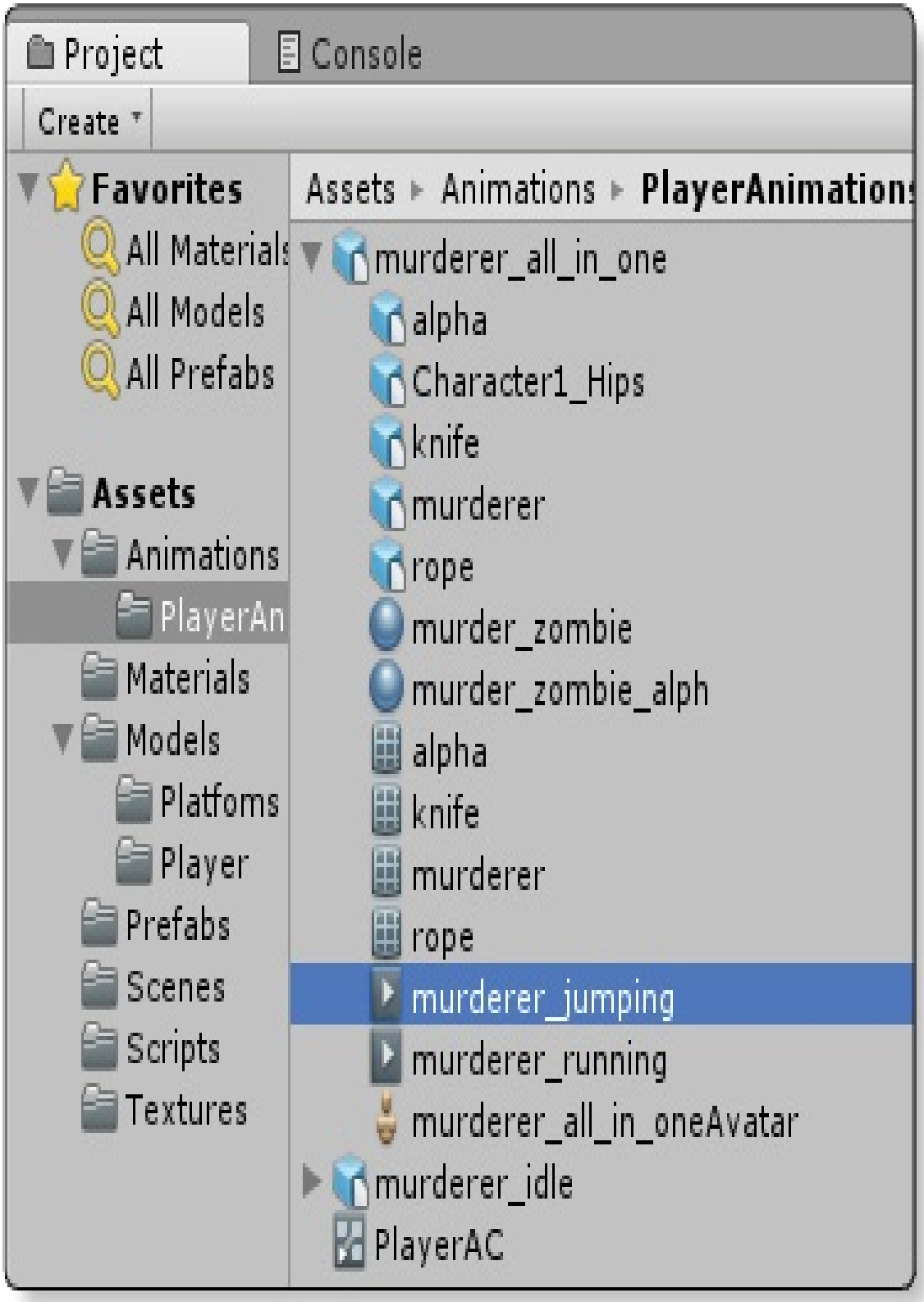

Clips
Start
End
murderer_running
553.0
576.0
murderer_jumping
405.0
464.0
murderer_jumping
Length 1.967
30 FPS
0:00
20:00
30:00
40:00
Start
405
End
464
Loop Time
Loop Pose
Cycle Offset
0
Additive Reference Po
Pose Frame
0
Curves
Events
Mask
Motion
Import Messages
Revert
Apply
murderer_jumping
2D
1.00
0:14 (024.2%) Frame 419

Creación del clip de salto del Player

NOTA:

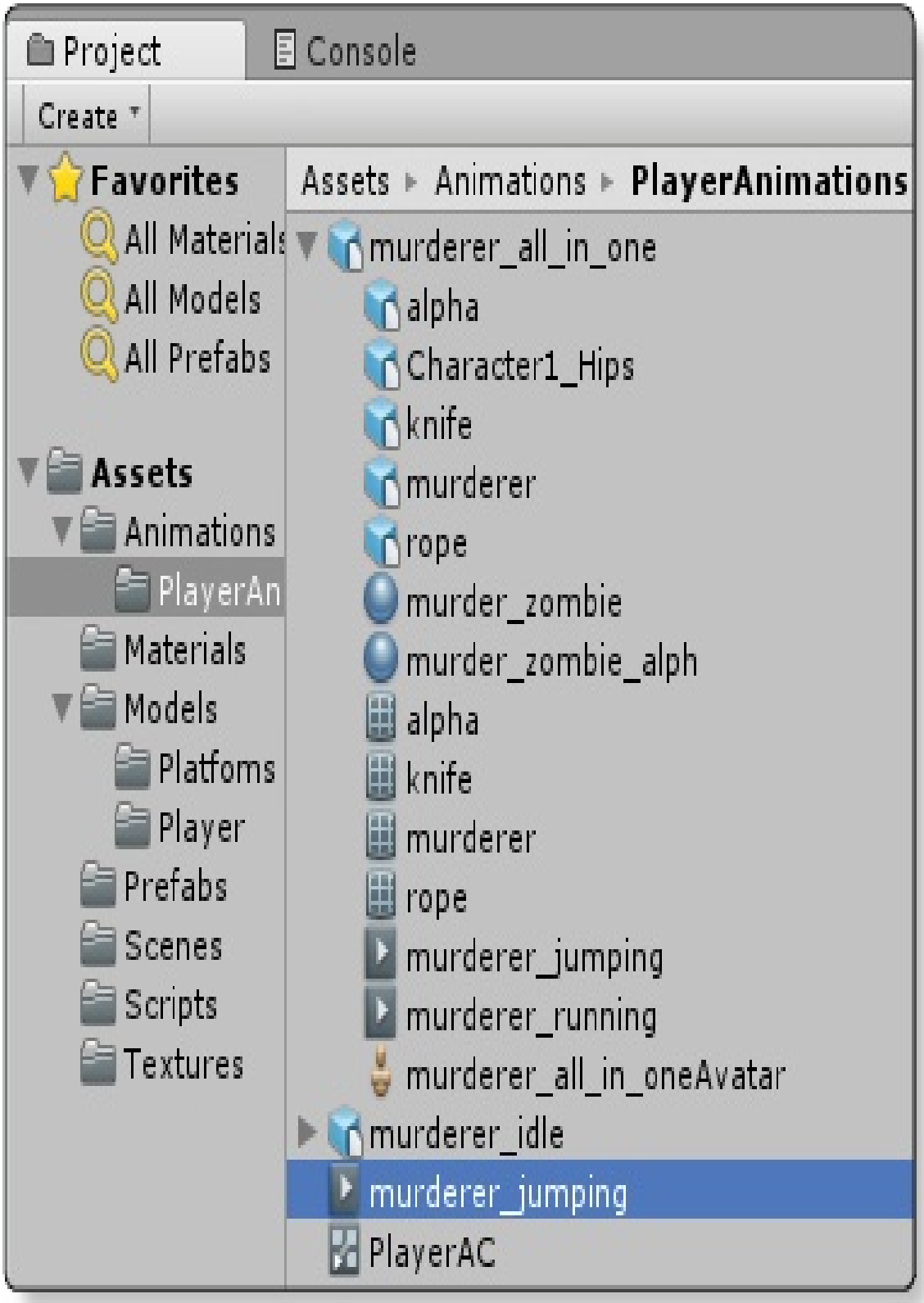
Dado que más adelante vamos a editar este clip de animación, debemos de hacer un duplicado de ella, si no, no nos va a permitir la edición.

En la ventana de “Proyect” seleccione el clip “murderer_jumping” creado y asigne Control + D para duplicarla.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platfoms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Animations ▸ PlayerAnimations
murderer_all_in_one
alpha
Character1_Hips
knife
murderer
rope
murder_zombie
murder_zombie_alph
alpha
knife
murderer
rope
murderer_jumping
murderer_running
murderer_all_in_oneAvatar
murderer_idle
PlayerAC

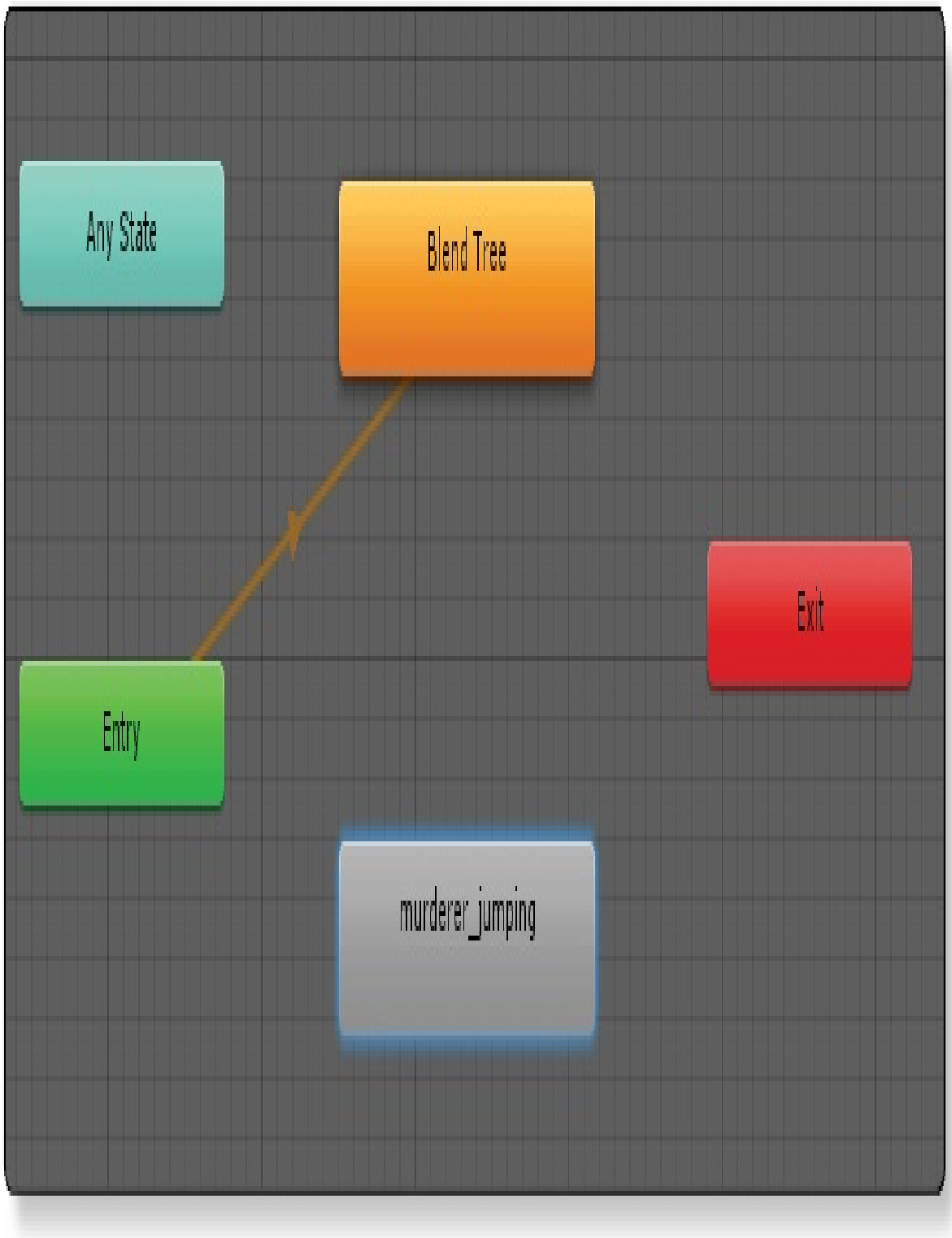
Duplicado del clip “murderer_jumping”

El duplicado aparecerá fuera de la jerarquía.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platfoms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Animations ▸ PlayerAnimations
murderer_all_in_one
alpha
Character1_Hips
knife
murderer
rope
murder_zombie
murder_zombie_alph
alpha
knife
murderer
rope
murderer_jumping
murderer_running
murderer_all_in_oneAvatar
murderer_idle
murderer_jumping
PlayerAC

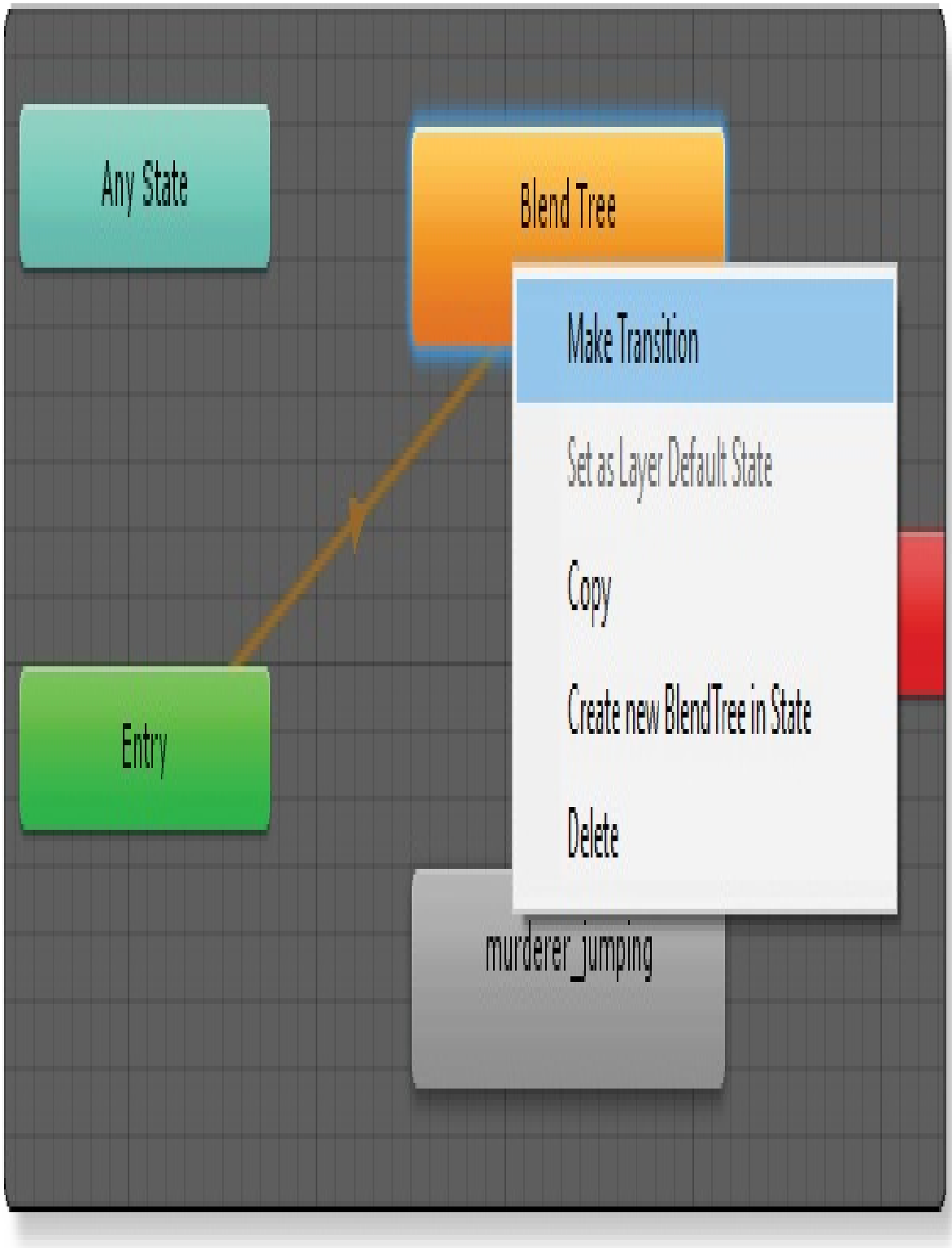
Reposición del clip “murderer_jumping”

Vaya a la ventana “Animator” y coloque el duplicado de la animación recién hecho.

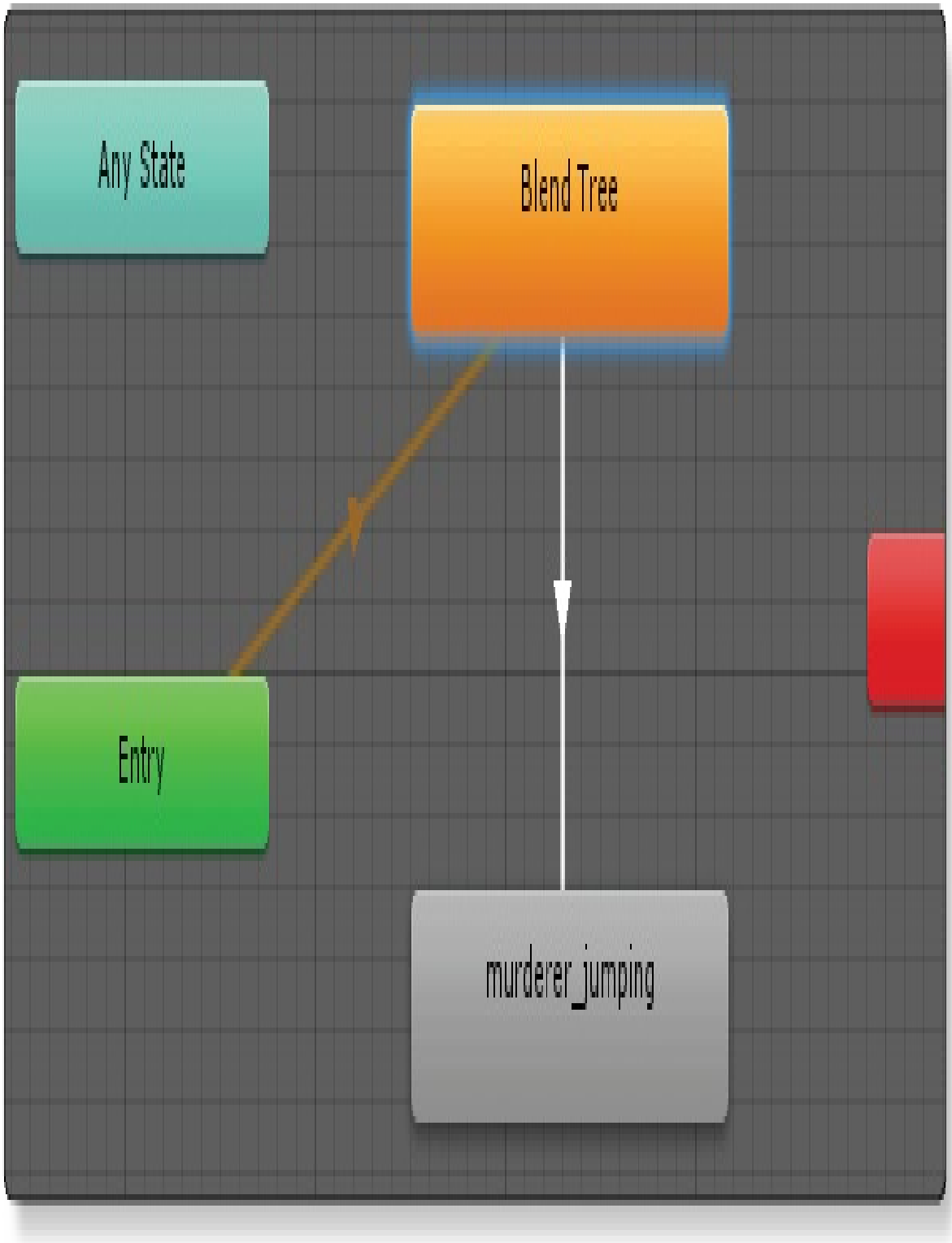

Any State
Blend Tree
Exit
Entry
murderer_jumping

Importar el clip a Animator

Clic derecho en “BlendTree” y asigne “MakeTransicion” y arrástrela al clip “murderer_jumping”.

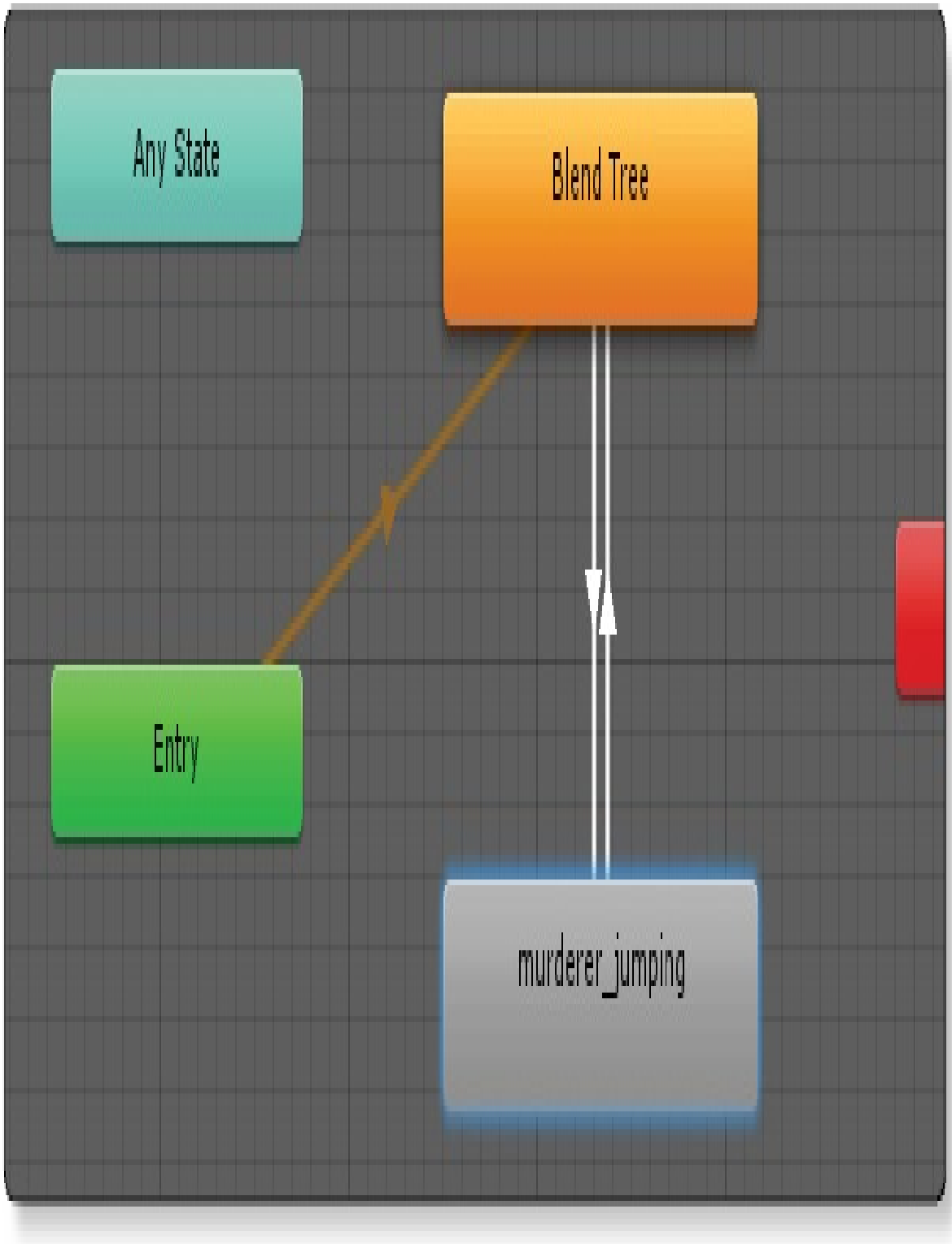

Any State
Blend Tree
Make Transition
Set as Layer Default State
Copy
Create new BlendTree in State
Delete
Entry
murderer_jumping

Creación de transición


Any State
Blend Tree
Entry
murderer_jumping

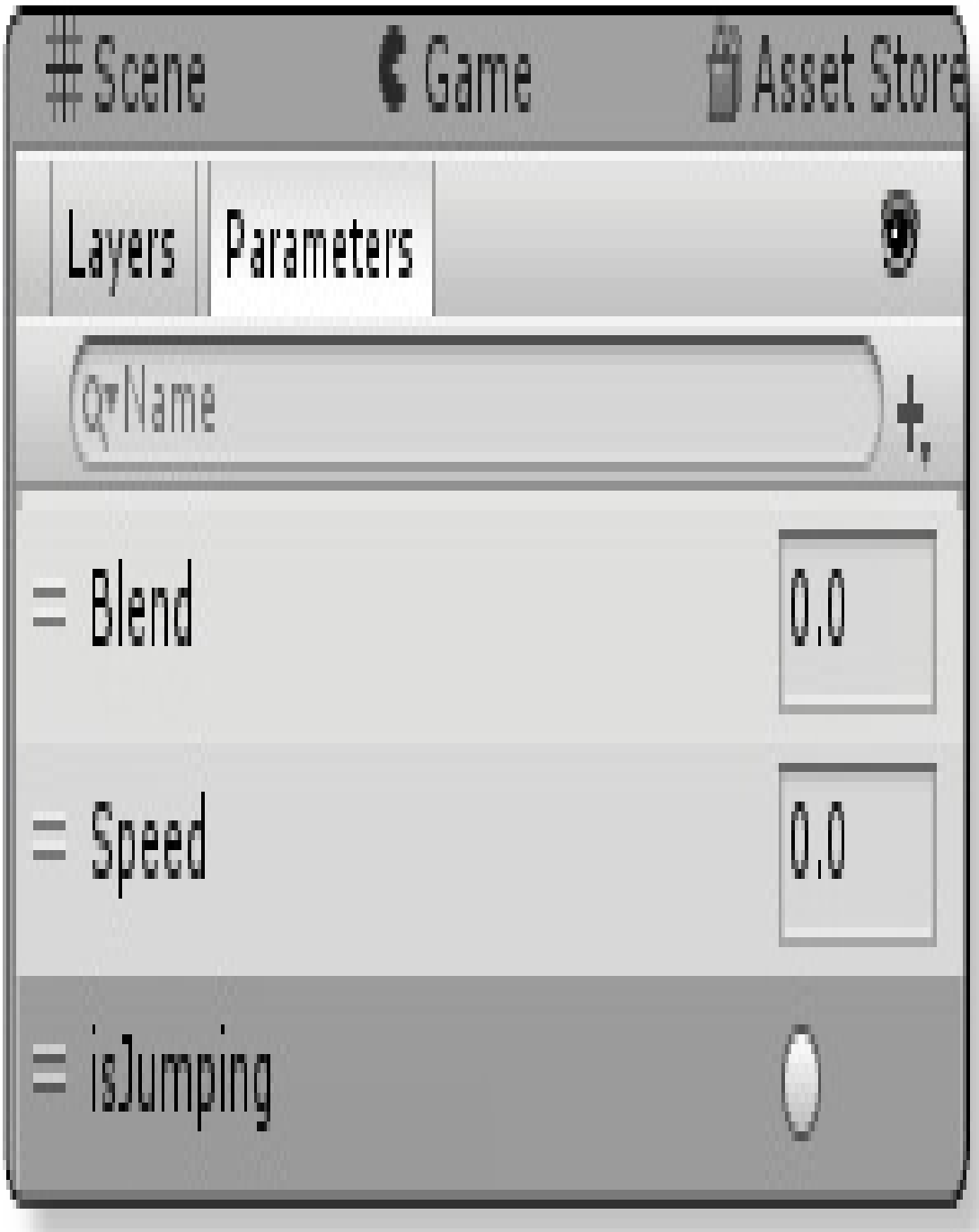
Transición creada

Repita el procedimiento para hacer la transición contraria.

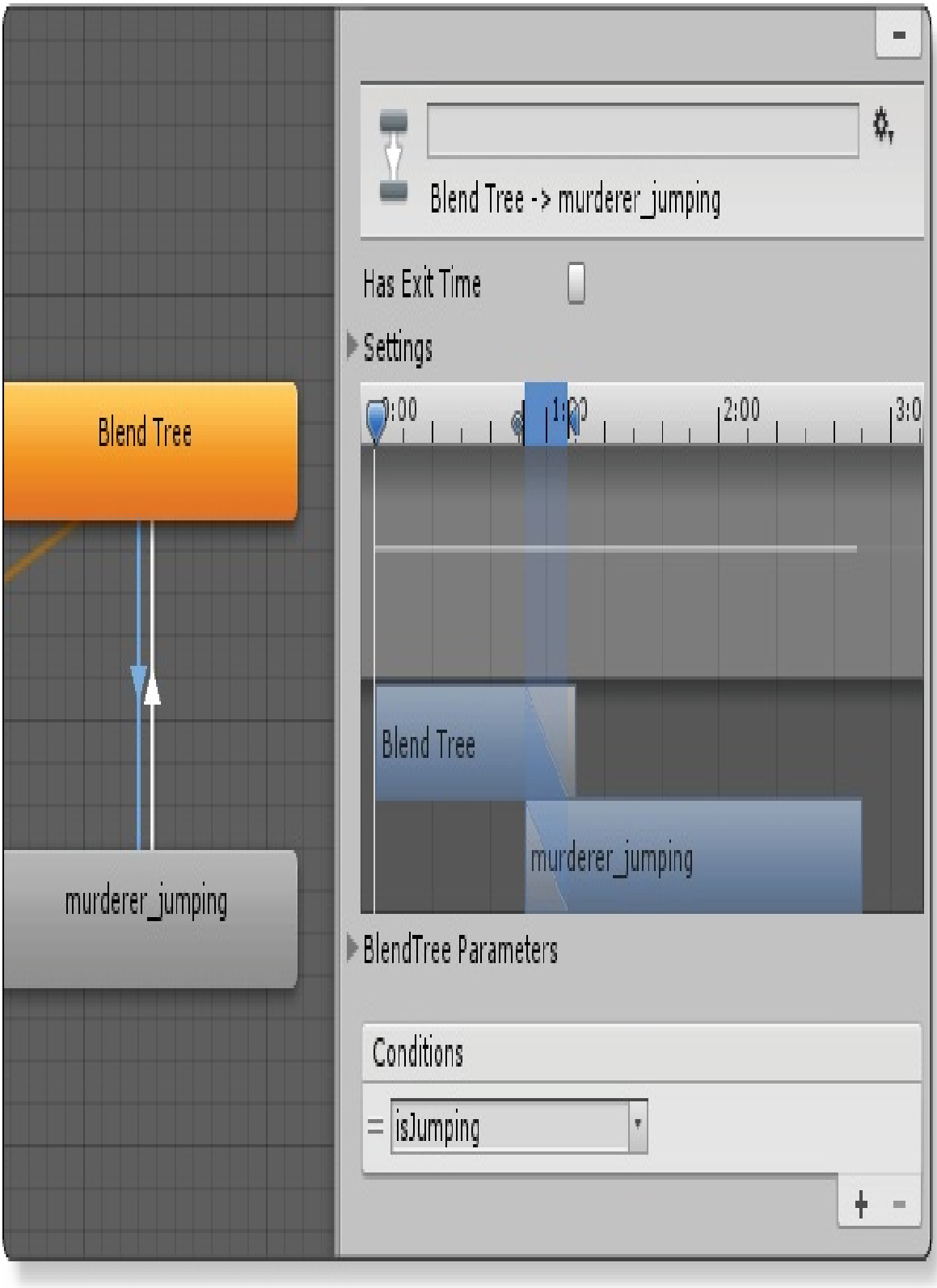

Any State
Blend Tree
Entry
murderer_jumping

Creación de la segunda transición

Haga un nuevo parámetro tipo “Trigger”, nómbrelo “isJumping” y asígnelo en la transición de entrada sin “Exit time”.


Scene
Game
Asset Store
Layers
Parameters
Name
Blend
0.0
Speed
0.0
isJumping

Creación de parámetro “isJumping”

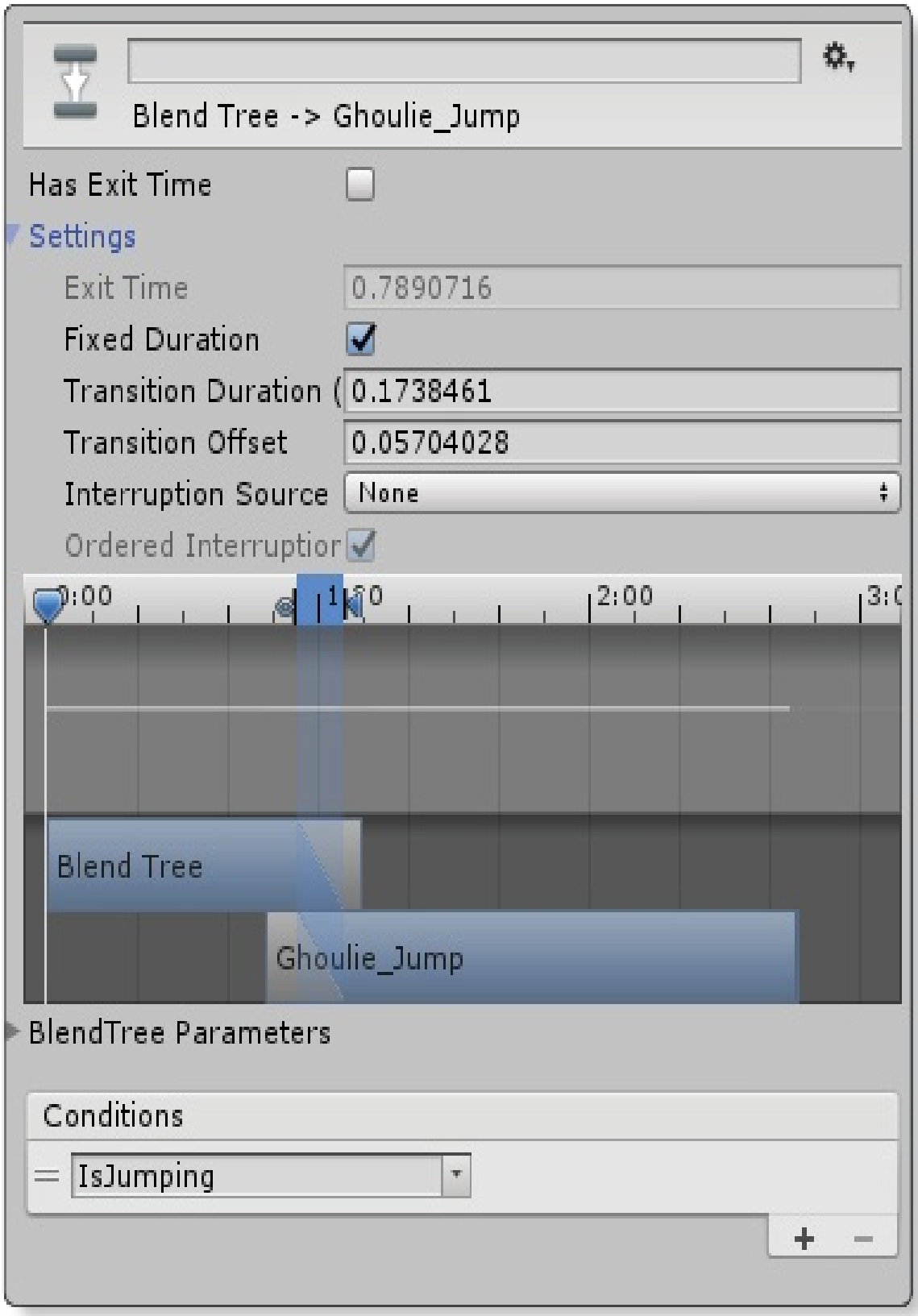

Blend Tree
murderer_jumping
Blend Tree -> murderer_jumping
Has Exit Time
Settings
2:00
3:0
Blend Tree
murderer_jumping
BlendTree Parameters
Conditions
isJumping

Asignación de parámetro

Los parámetros tipo “Trigger” los utilizamos dentro del scripting para correr una animación.

La propiedad “Has Exit Time” permite que la animación siguiente corra sin importar que la animación anterior no haya finalizado.

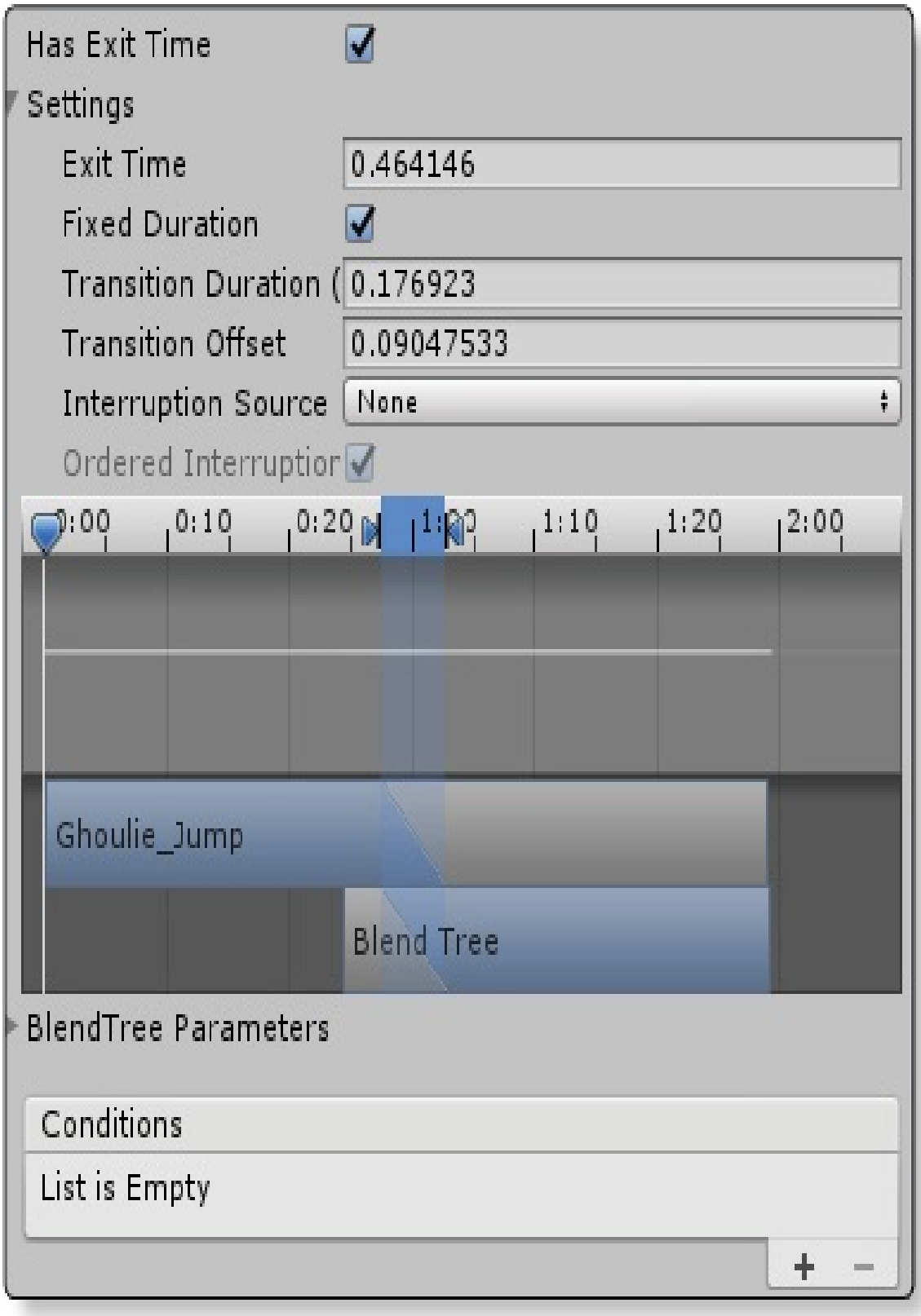
Con la transición de entrada seleccionada, expanda “Settings” y aplique los siguientes parámetros.


Blend Tree -> Ghoulie_Jump
Has Exit Time
Settings
Exit Time
0.7890716
Fixed Duration
Transition Duration (
0.1738461
Transition Offset
0.05704028
Interruption Source
None
Ordered Interruptior
2:00
Blend Tree
Ghoulie_Jump
BlendTree Parameters
Conditions
IsJumping

Parámetros de transición de ida

Dichos parámetros nos indican en que parte de la animación va a iniciar la transición y en que parte va a finalizarla.

Seleccione la transición de vuelta y asigne los parámetros que se muestra.


Has Exit Time
Settings
Exit Time
0.464146
Fixed Duration
Transition Duration (
0.176923
Transition Offset
0.09047533
Interruption Source
None
Ordered Interruptior
0:00
0:10
0:20
1:10
1:20
2:00
Ghoulie_Jump
Blend Tree
BlendTree Parameters
Conditions
List is Empty

Parámetros de transición de vuelta

Vaya al script de “CharacterMovement” y añada la siguiente línea de código en la función “Update” para arrancar la animación cuando nuestro “Player” brinque.

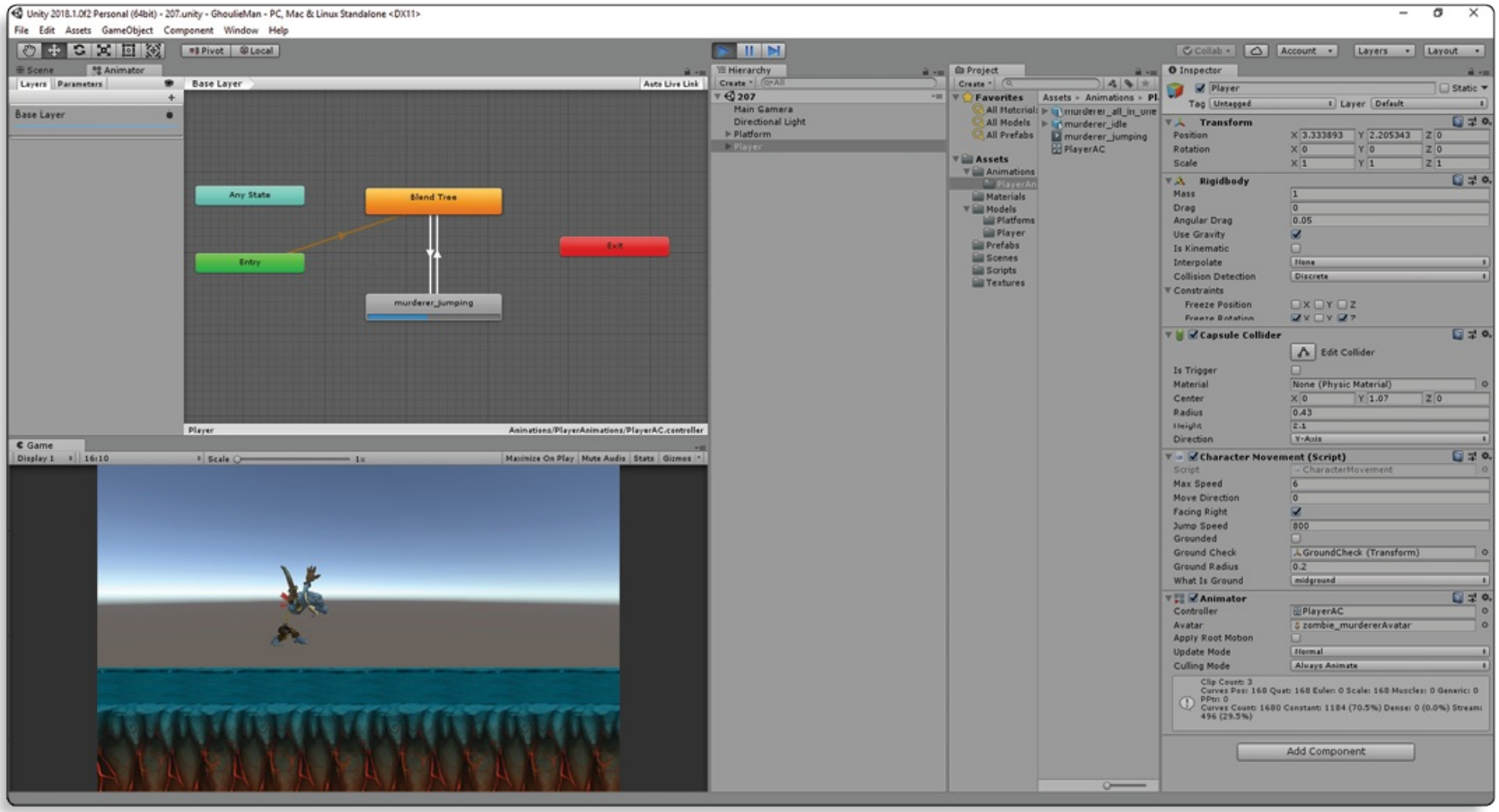
```
void Update () {
    moveDirection = Input.GetAxis("Horizontal");

    if (grounded && Input.GetButtonDown ("Jump")){
        anim.SetTrigger("isJumping");
        rigidbody.AddForce (new Vector2 (0, jumpSpeed));
    }
}
```

Arranque de animación

Salve el script y Previsualice el juego para observar la animación de salto.

Layout 2 by 3

Cuando necesitemos previsualizar nuestro videojuego en modo “Play” y a su vez tengamos la necesidad de ver otra pantalla activa llámese la de “Scene” o “Animator” podemos configurar el “layout 2 by 3” asignando en el menú general “Window/ Layouts/ 2 by 3”.


Unity 2018.1.0f2 Personal (64bit) - 207.unity - GhoulieMan - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Window Help
Pivot Local
Scene Animator
Layers Parameters
Base Layer
Auto Live Link
Any State
Blend Tree
Entry
Exit
murderer_jumping
Player
Animations/PlayerAnimations/PlayerAC.controller
Game
Display 1 16:10 Scale 1x
Maximize On Play Mute Audio Stats Gizmos
Hierarchy
Create
207
Main Camera
Directional Light
Platform
Player
Project
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platforms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Animations ▸ Pl
murderer_all_in_one
murderer_idle
murderer_jumping
PlayerAC
Collab
Account
Layers
Layout
Inspector
Player
Static
Tag Untagged
Layer Default
Transform
Position X 3.333893 Y 2.205343 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position X Y Z
Capsule Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 1.07 Z 0
Radius 0.43
Height 2.1
Direction Y-Axis
Character Movement (Script)
Script CharacterMovement
Max Speed 6
Move Direction 0
Facing Right
Jump Speed 800
Grounded
Ground Check GroundCheck (Transform)
Ground Radius 0.2
What Is Ground midground
Animator
Controller PlayerAC
Avatar zombie_murdererAvatar
Apply Root Motion
Update Mode Normal
Culling Mode Always Animate
Clip Count: 3
Curves Pos: 168 Quat: 168 Euler: 0 Scale: 168 Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 1680 Constant: 1184 (70.5%) Dense: 0 (0.0%) Stream: 496 (29.5%)
Add Component

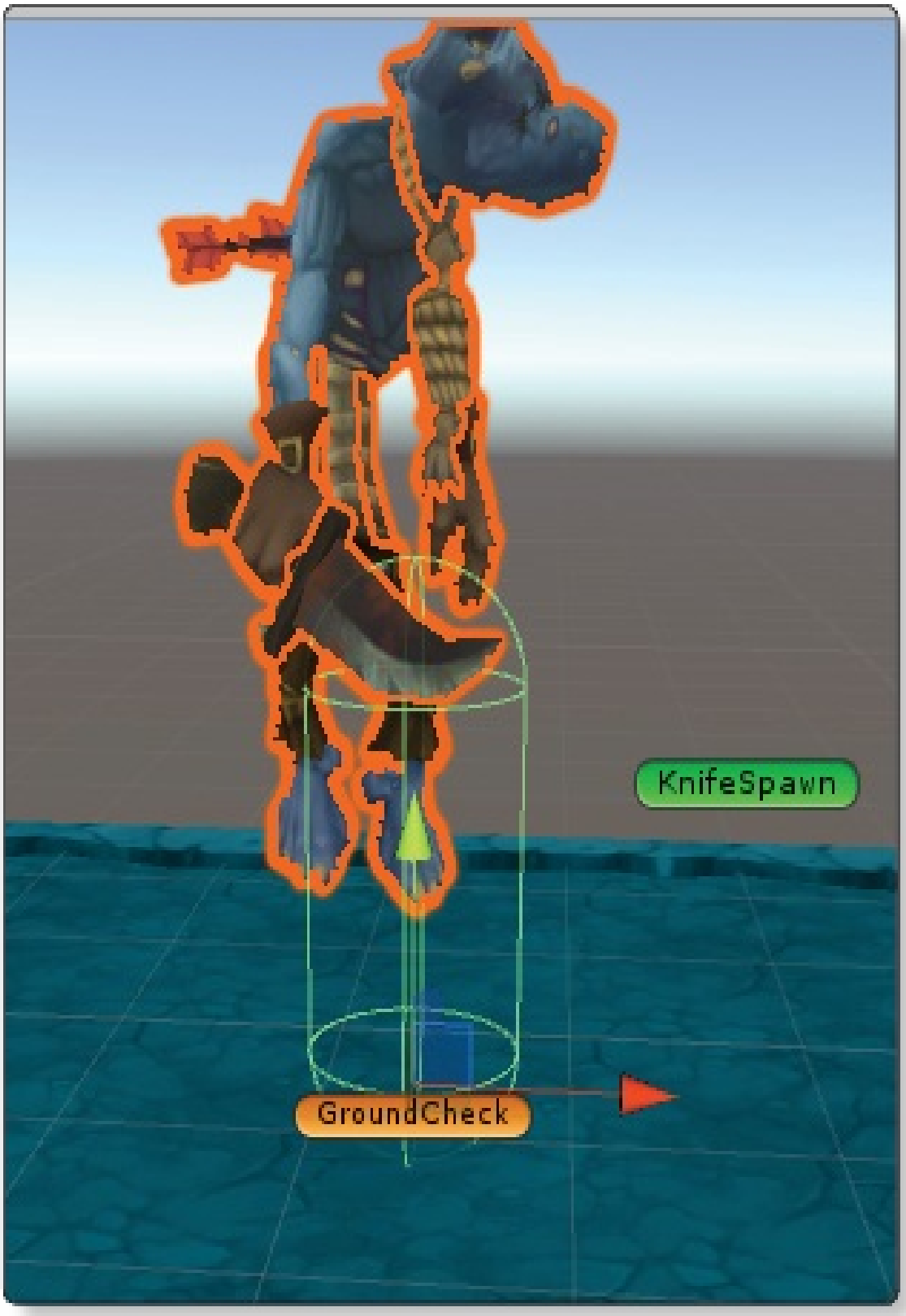
Layout 2 by 3

Para regresar a la vista “Default” asignar en el menú principal “Window/ Layouts/ Default”.

Vaya a la vista 2 by 3 y configure en la parte de arriba la ventana “Scene” y abajo “Game”

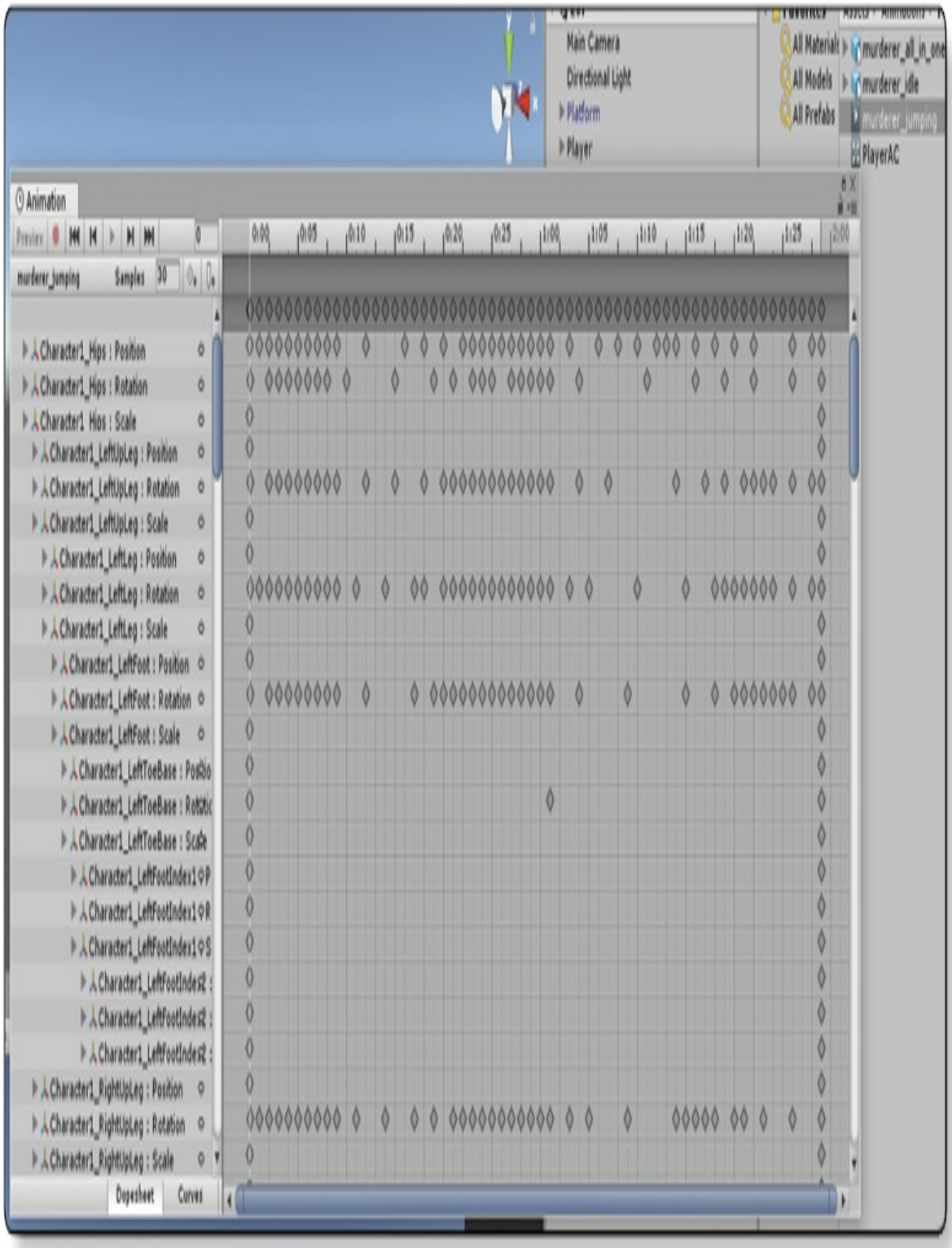
Seleccione el “Player” y haga un brinco

PROBLEMA: La animación salta físicamente más alto y el Collider no la sigue.


KnifeSpawn
GroundCheck

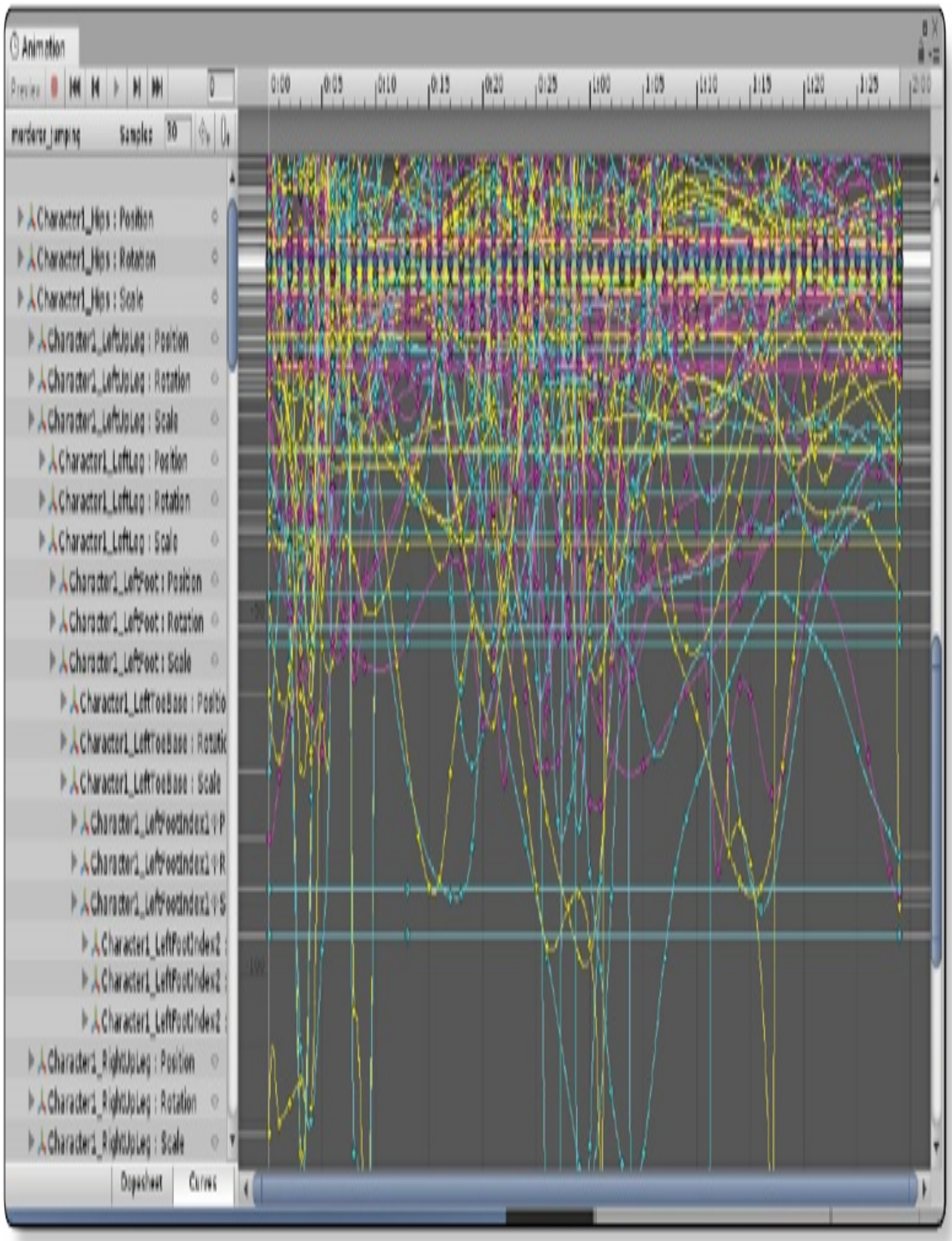
Problema con el salto

SOLUCIÓN: Vaya al folder de “PlayerAnimations”, seleccione el clip de animación relativo al salto y abra la ventana “Animation” (Window / Animation).


Main Camera
Directional Light
Platform
Player
All Materials
All Models
All Prefabs
murderer_all_in_one
murderer_idle
murderer_jumping
PlayerAC
Animation
Preview
murderer_jumping
Samples
30
Character1_Hips : Position
Character1_Hips : Rotation
Character1_Hips : Scale
Character1_LeftUpLeg : Position
Character1_LeftUpLeg : Rotation
Character1_LeftUpLeg : Scale
Character1_LeftLeg : Position
Character1_LeftLeg : Rotation
Character1_LeftLeg : Scale
Character1_LeftFoot : Position
Character1_LeftFoot : Rotation
Character1_LeftFoot : Scale
Character1_LeftToeBase : Position
Character1_LeftToeBase : Rotation
Character1_LeftToeBase : Scale
Character1_RightUpLeg : Position
Character1_RightUpLeg : Rotation
Character1_RightUpLeg : Scale
Dopesheet
Curves

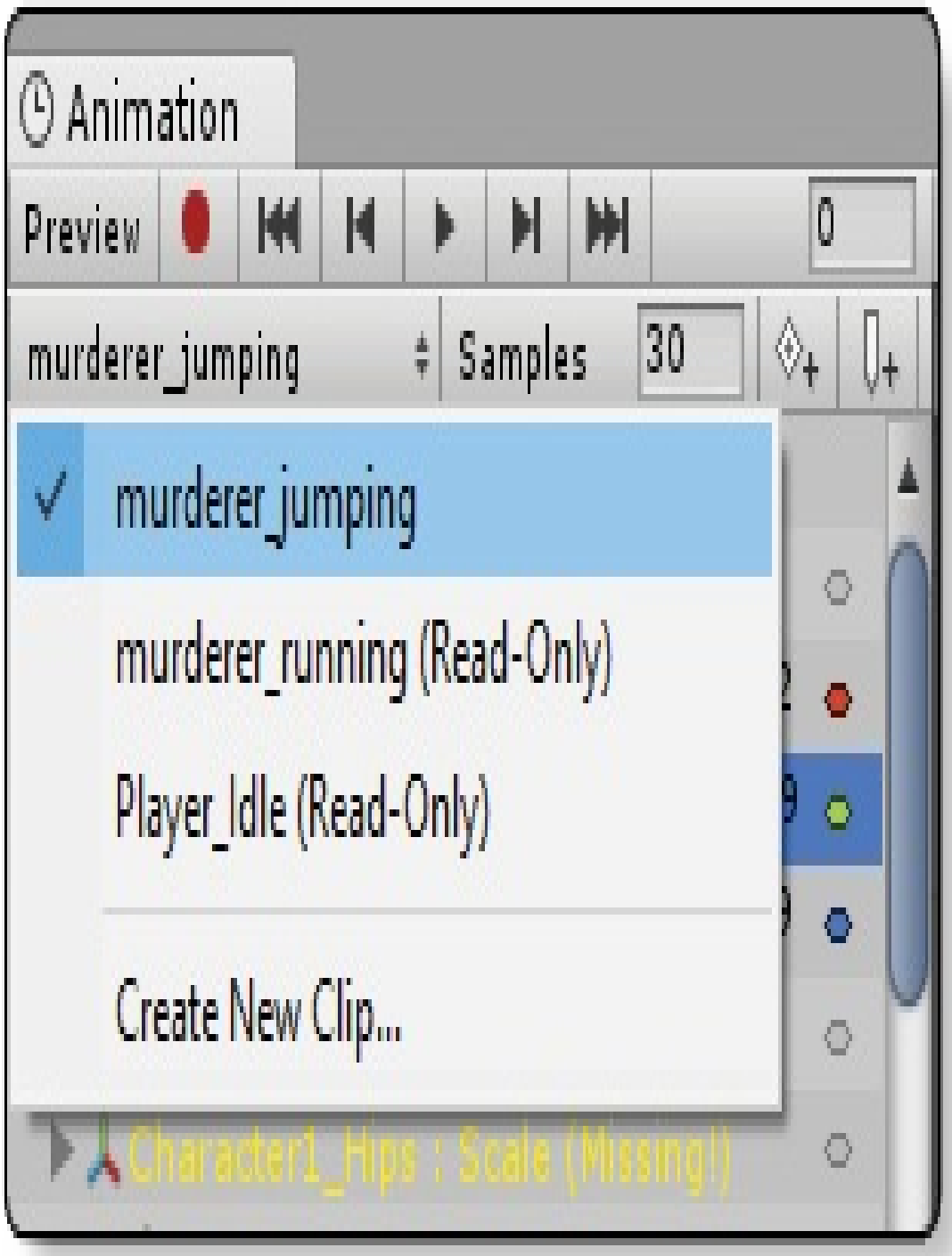
Ventana Animation

Clic en el botón inferior izquierdo “Curves” para visualizar sus curvas de animación.


Animation
Preview
Samples
Character1_Hips : Position
Character1_Hips : Rotation
Character1_Hips : Scale
Character2_LeftUpLeg : Position
Character2_LeftUpLeg : Rotation
Character2_LeftUpLeg : Scale
Character1_LeftLeg : Position
Character1_LeftLeg : Rotation
Character1_LeftLeg : Scale
Character2_LeftFoot : Position
Character2_LeftFoot : Rotation
Character2_LeftFoot : Scale
Character1_LeftToeBase : Position
Character1_LeftToeBase : Rotation
Character1_LeftToeBase : Scale
Character2_RightUpLeg : Position
Character2_RightUpLeg : Rotation
Character2_RightUpLeg : Scale
Dopesheet
Curves

Modo de curvas de Animation

En la esquina superior izquierda expanda y seleccione “murderer_jumping”.


Animation
Preview
0
murderer_jumping
Samples
30
murderer_jumping
murderer_running (Read-Only)
Player_Idle (Read-Only)
Create New Clip...
Character1_Hips : Scale (Missing!)

Selección de clip de animación

NOTA:

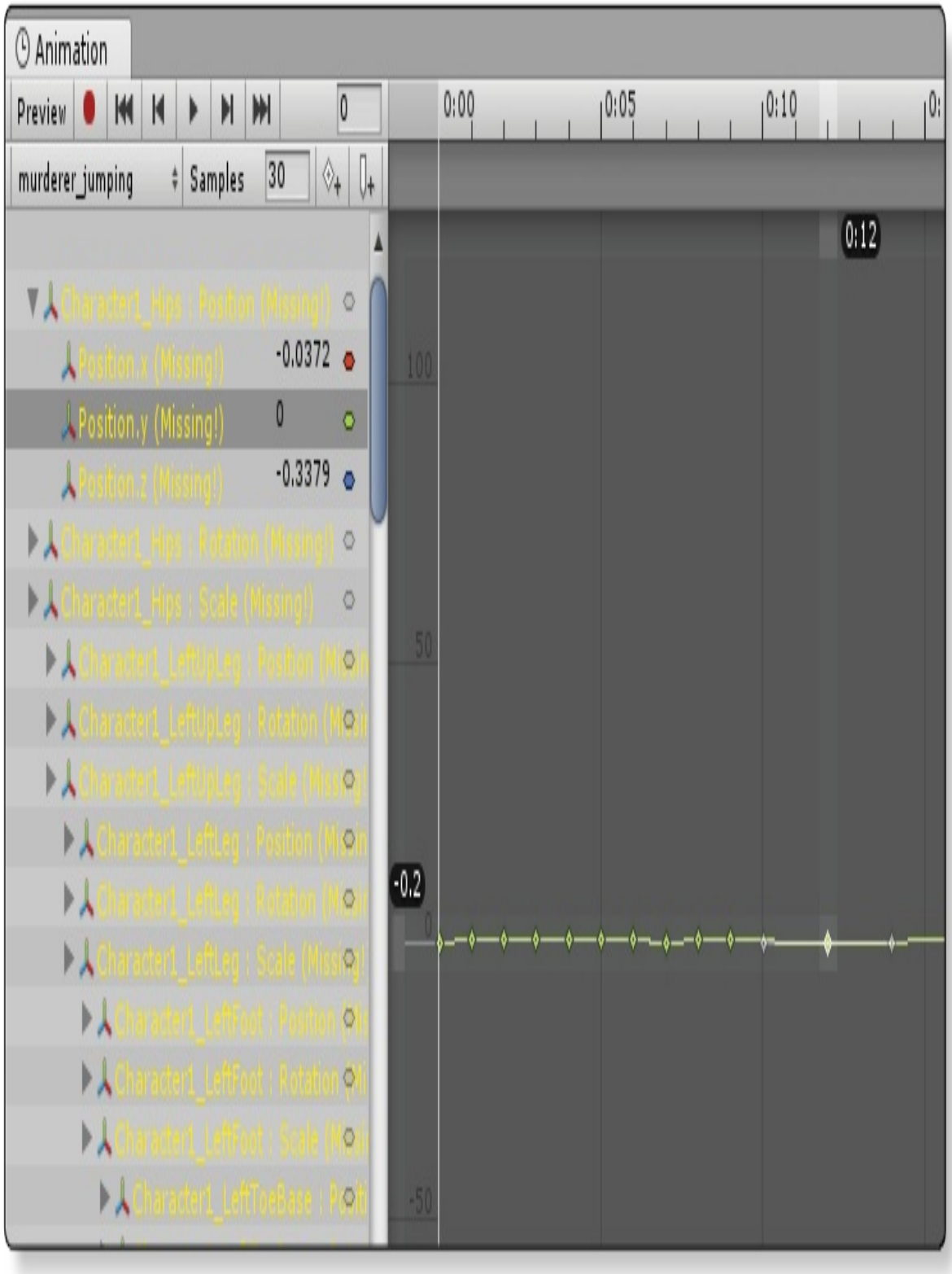
De no haber hecho el duplicado del clip de la animación anteriormente, no hubiéramos podido editar las curvas ya que únicamente nos aparecería la opción (Read-Only).

Seleccione el nodo relativo a “Hips Position”.

Animation
Preview
0
murderer_jumping
Samples
30
Character1_Hips : Position (Missing!)
Position.x (Missing!)
-0.0372
Position.y (Missing!)
0.49599
Position.z (Missing!)
-0.3379
Character1_Hips : Rotation (Missing!)
Character1_Hips : Scale (Missing!)

Selección de nodo de animación

Seleccione los nodos altos y bájelos manualmente a cero uno por uno.


Animation
Preview
0
0:00
0:05
0:10
murderer_jumping
Samples
30
0:12
Character1_Hips : Position (Missing!)
Position.x (Missing!)
-0.0372
Position.y (Missing!)
0
Position.z (Missing!)
-0.3379
Character1_Hips : Rotation (Missing!)
Character1_Hips : Scale (Missing!)
Character1_LeftUpLeg : Position (Missin
Character1_LeftUpLeg : Rotation (Missi
Character1_LeftUpLeg : Scale (Missing!
Character1_LeftLeg : Position (Missin
Character1_LeftLeg : Rotation (Missi
Character1_LeftLeg : Scale (Missing!
Character1_LeftFoot : Position (Mis
Character1_LeftFoot : Rotation (Mi
Character1_LeftFoot : Scale (Missi
Character1_LeftToeBase : Positi
100
50
-0.2
0
-50

Edición de nodos de animación

Esta edición la hacemos para que en la animación nuestro personaje se quede en su lugar y el movimiento en “Y” lo haga únicamente el elemento “Player”.

Previsualice el juego para observar que el CapsulleCollider ya sigue a la animación.

GroundCheck

Problema solucionado

PROGRAMACIÓN DEL ATAQUE DEL PLAYER

En este apartado programaremos la funcionalidad de ataque del Player, importaremos su proyectil y se programará la clonación de una instancia como el proyectil.

Haga un nuevo “EmptyGameObject”, reset sus transformaciones, renómbrelo “KnifeSpawn”, añada un “Label” color verde y colóquelo donde se muestra.

KnifeSpawn
GroundCheck

Creación de “KnifeSpawn”

Este objeto va a ser el generador de los proyectiles.

Empareje “KnifeSpawn” con el “Player”.

Hierarchy
Create
All
208*
Main Camera
Directional Light
Platform
Player
zombie_murderer
CameraTarget
GroundCheck
KnifeSpawn

Emparentamiento del KnifeSpawn

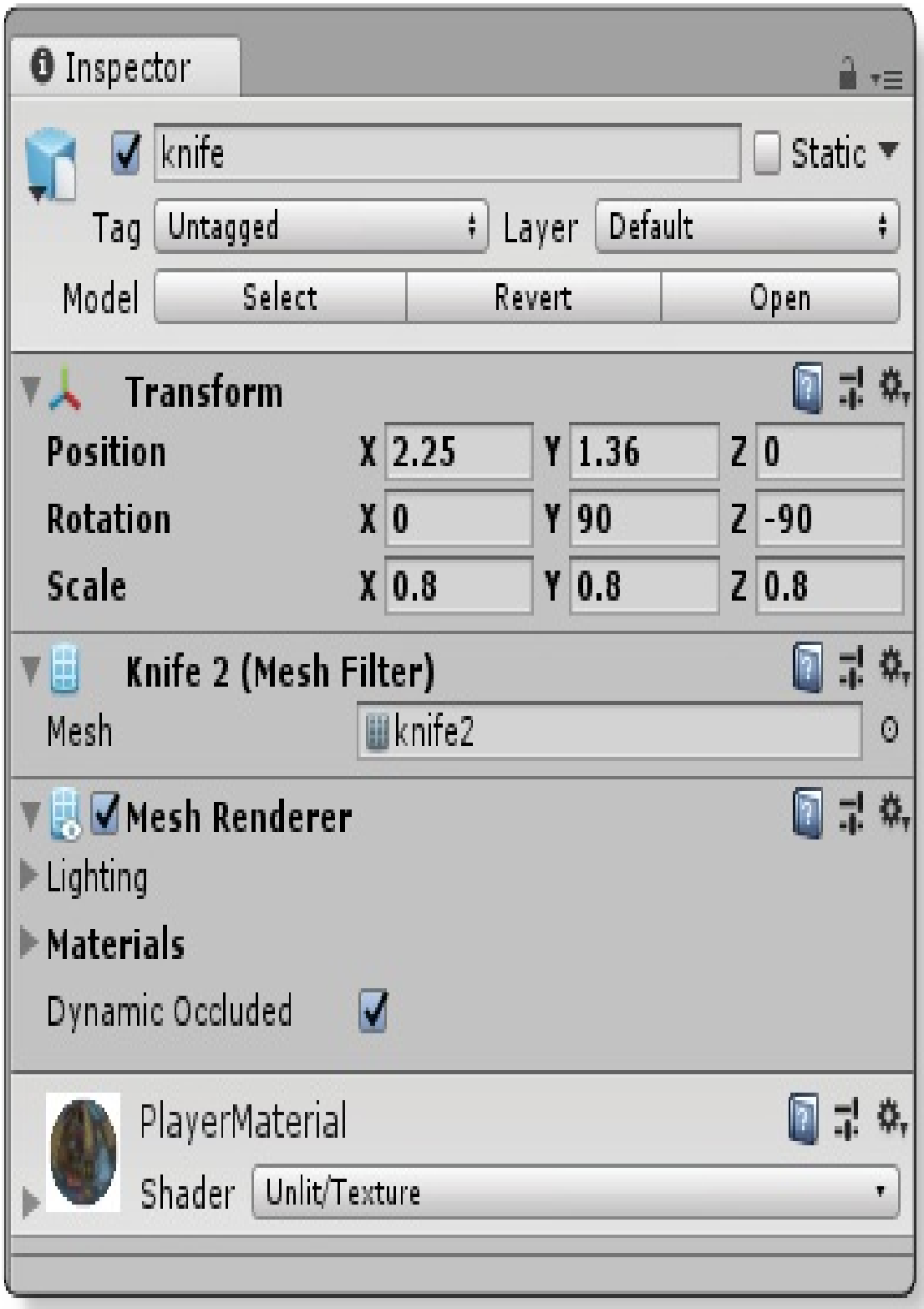
En seguida queremos crear el “Prefab” del proyectil de nuestro “Player”.

Vaya al folder de “Models” de la ventana “Project” e importe el modelo “knife” de la carpeta “Models” dentro de “resources”

Con los conocimientos adquiridos coloque y texturice el objeto “knife” en la escena.

KnifeSpawn
CameraTarget
GroundCheck

Importación y texturización de Knife


Inspector
knife
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 2.25
Y 1.36
Z 0
Rotation
X 0
Y 90
Z -90
Scale
X 0.8
Y 0.8
Z 0.8
Knife 2 (Mesh Filter)
Mesh
knife2
Mesh Renderer
Lighting
Materials
Dynamic Occluded
PlayerMaterial
Shader
Unlit/Texture

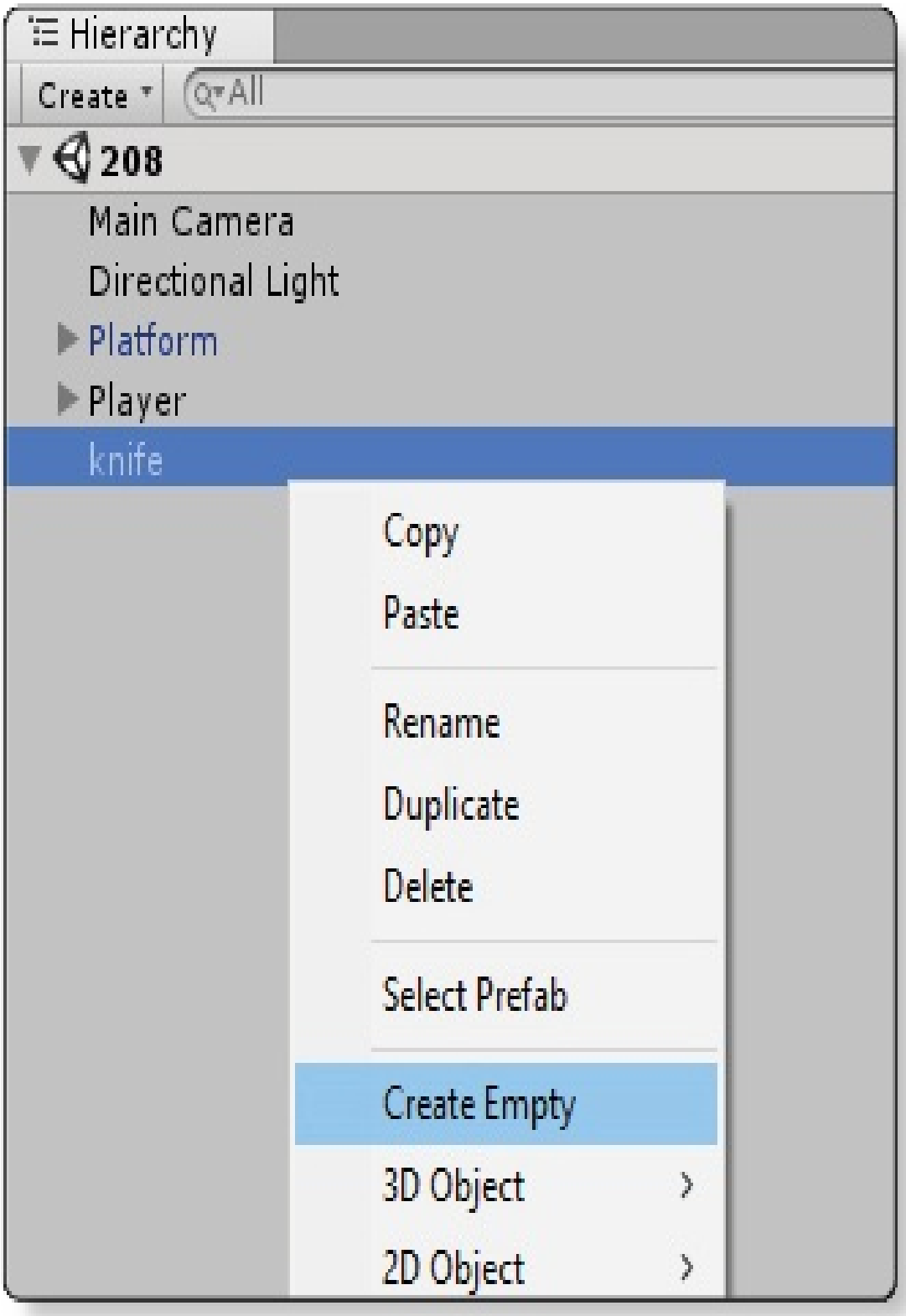
Material para Knife

NOTA:

Se le asignará el mismo material que al “Player”, no necesita hacer un material extra para el “knife”.

Queremos crear un nuevo “EmptyGameObject” que se encuentre posicionado exactamente donde se encuentra nuestro objeto “Knife”, para ello utilizaremos una técnica que se describe en los pasos siguientes.

Seleccione en “Hierarchy” el objeto “knife”, asigne clic derecho y elija “Create Empty”.


Hierarchy
Create
All
208
Main Camera
Directional Light
Platform
Player
knife
Copy
Paste
Rename
Duplicate
Delete
Select Prefab
Create Empty
3D Object
2D Object

Creación de EmptyObject vía Hierarchy

Se creará el nuevo “EmptyGameObject” como hijo del objeto “knife” y heredando sus transformaciones.

Renombre el nuevo objeto como “Knife_PFB”.

Hierarchy
Create
All
208*
Main Camera
Directional Light
Platform
Player
knife
Knife_PFB

Renombrar objeto

Dado que no deseamos que el nuevo objeto sea hijo de “knife” sino al contrario, que sea padre de “knife” pasamos a la siguiente instrucción.

Seleccione el objeto “Knife_PFB” y lo arrastramos fuera de la jerarquía.

Hierarchy
Create
All
208*
Main Camera
Directional Light
Platform
Player
knife
Knife_PFB

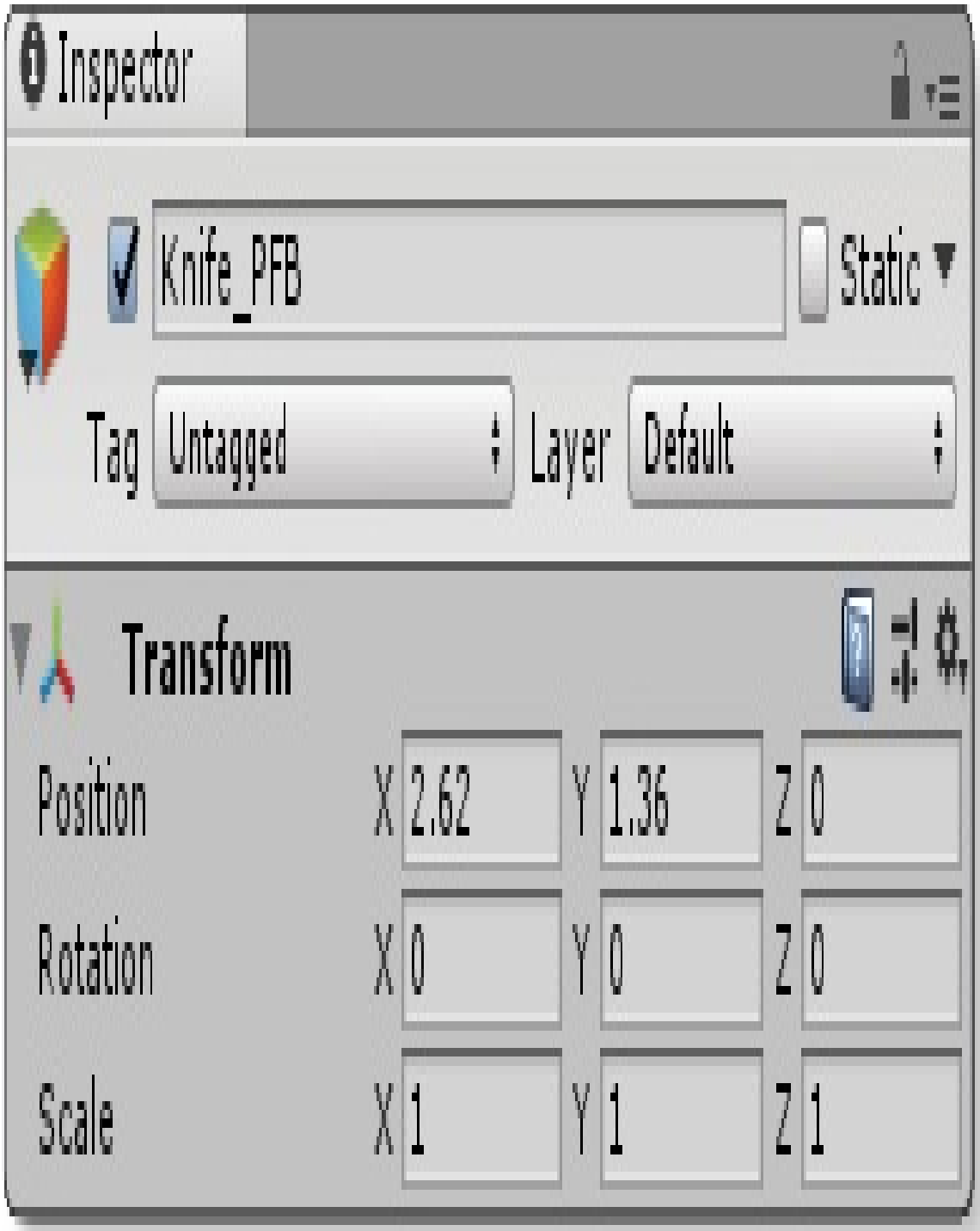
Desemparejar Knife_PFB

Como se comentaba anteriormente no deseamos que conserve la herencia de transformaciones del objeto “knife” adquiridas, así que procedemos a resetear sus transformaciones en cuanto a rotación y escala se refieren.

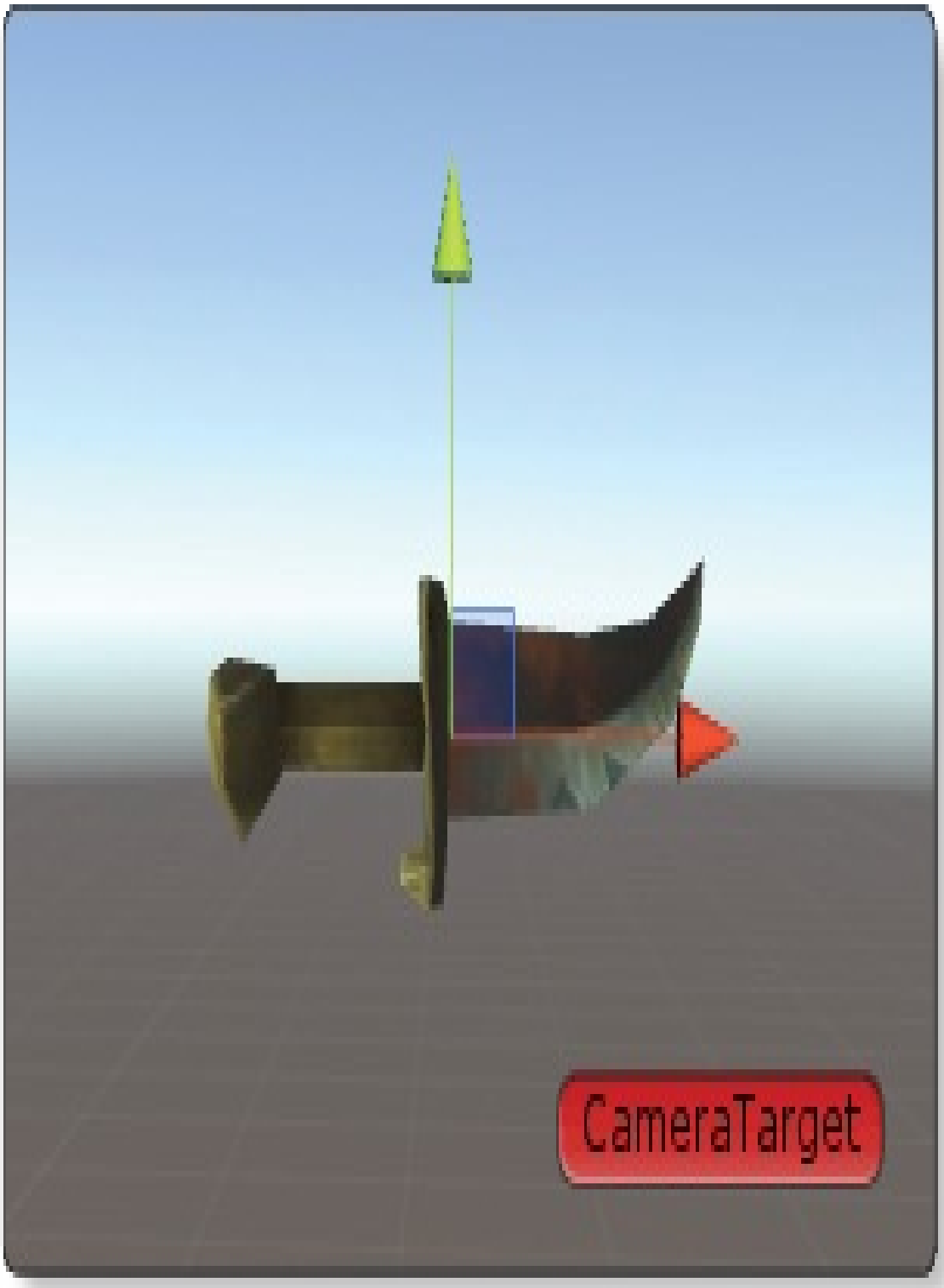
NOTA:

No es recomendable que los objetos que provienen de un “EmptyObject” como “Player” o como “Knife_PFB” contentan en sus atributos de escala otro valor que no sea uno.

Haga reset en las transformaciones de rotación y escala de “Knife_PFB”.


Inspector
Knife_PFB
Static
Tag
Untagged
Layer
Default
Transform
Position
X 2.62
Y 1.36
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1

Transformaciones deKnife_PFB


CameraTarget

Knife_PFB correctamente colocado

Emparentamos “knife” con “Knife_PFB”.

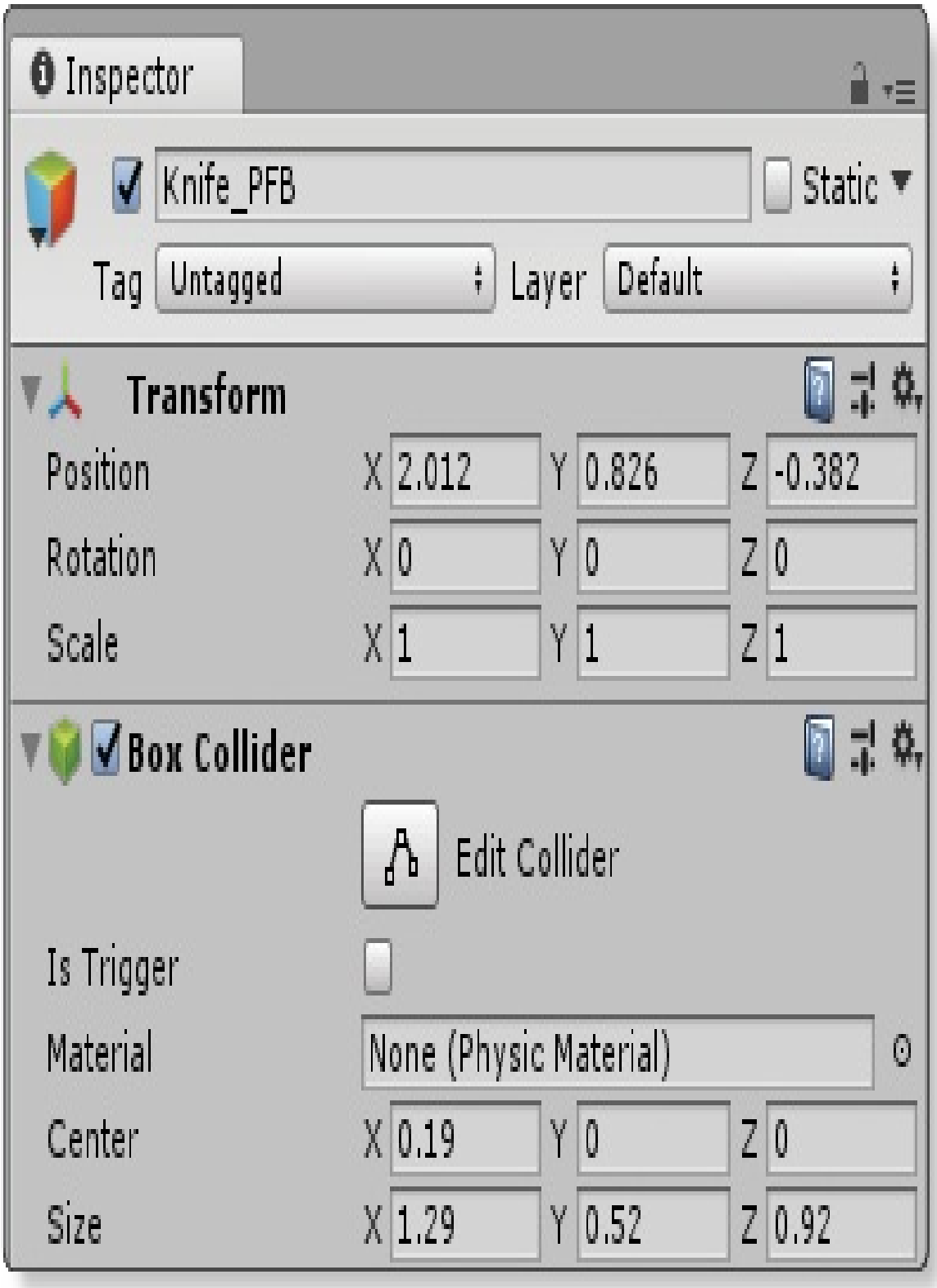
Hierarchy
Create
All
208*
Main Camera
Directional Light
Platform
Player
Knife_PFB
knife

Emparentamiento correcto

Asigne un componente “BoxCollider” al objeto “Knife_PFB” y edite sus parámetros.

CameraTarget
pawn

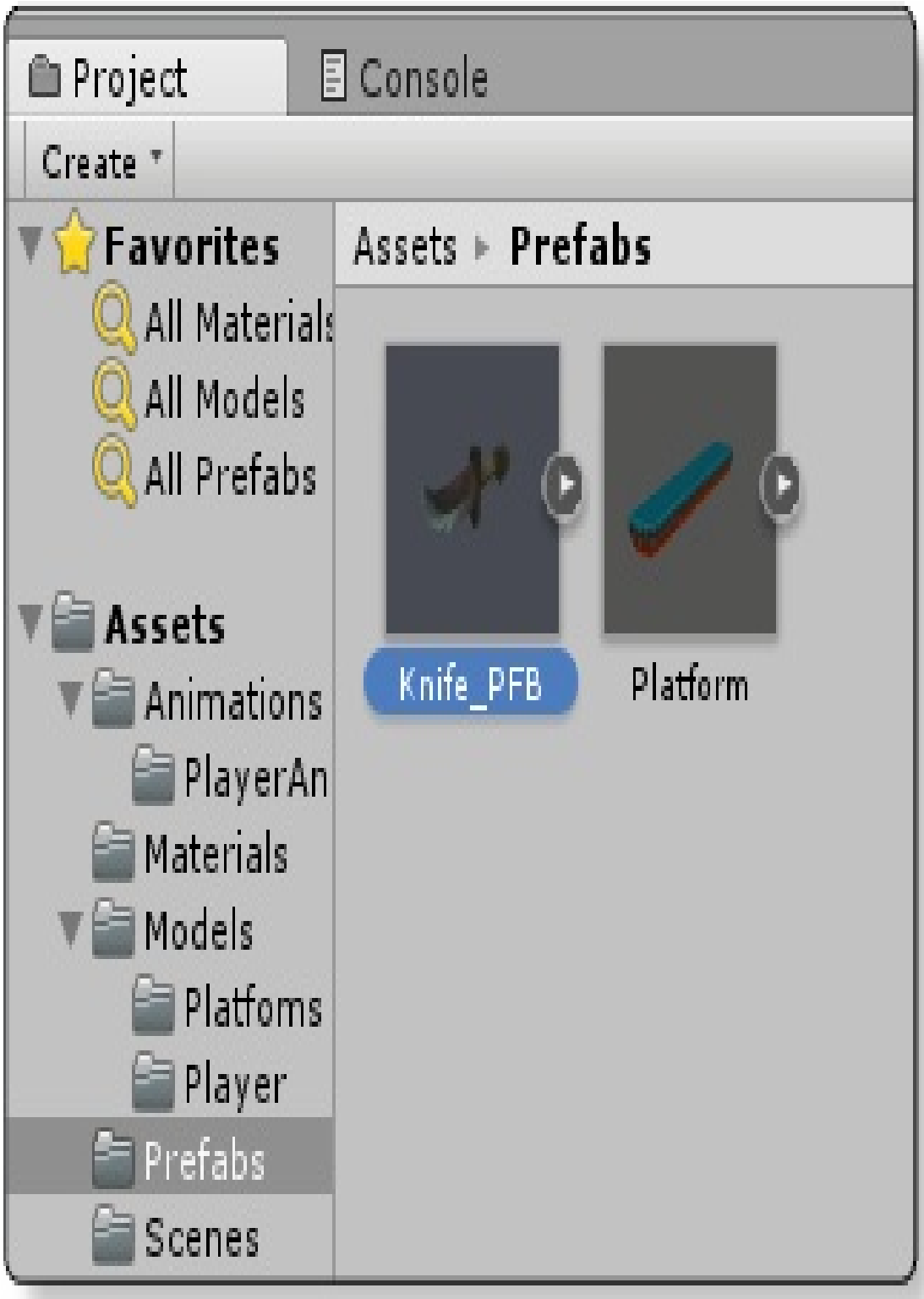
Box Collider


Inspector
Knife_PFB
Static
Tag
Untagged
Layer
Default
Transform
Position
X 2.012
Y 0.826
Z -0.382
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0.19
Y 0
Z 0
Size
X 1.29
Y 0.52
Z 0.92

Parámetros de BoxCollider

Asigne también un componente “RigidBody” y desactive su casilla “use Gravity”

Arrastre “Knife_PFB” de la ventana de “Hierarchy” al folder “Prefabs”.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platforms
Player
Prefabs
Scenes
Assets
Prefabs
Knife_PFB
Platform

Creación del Prefab

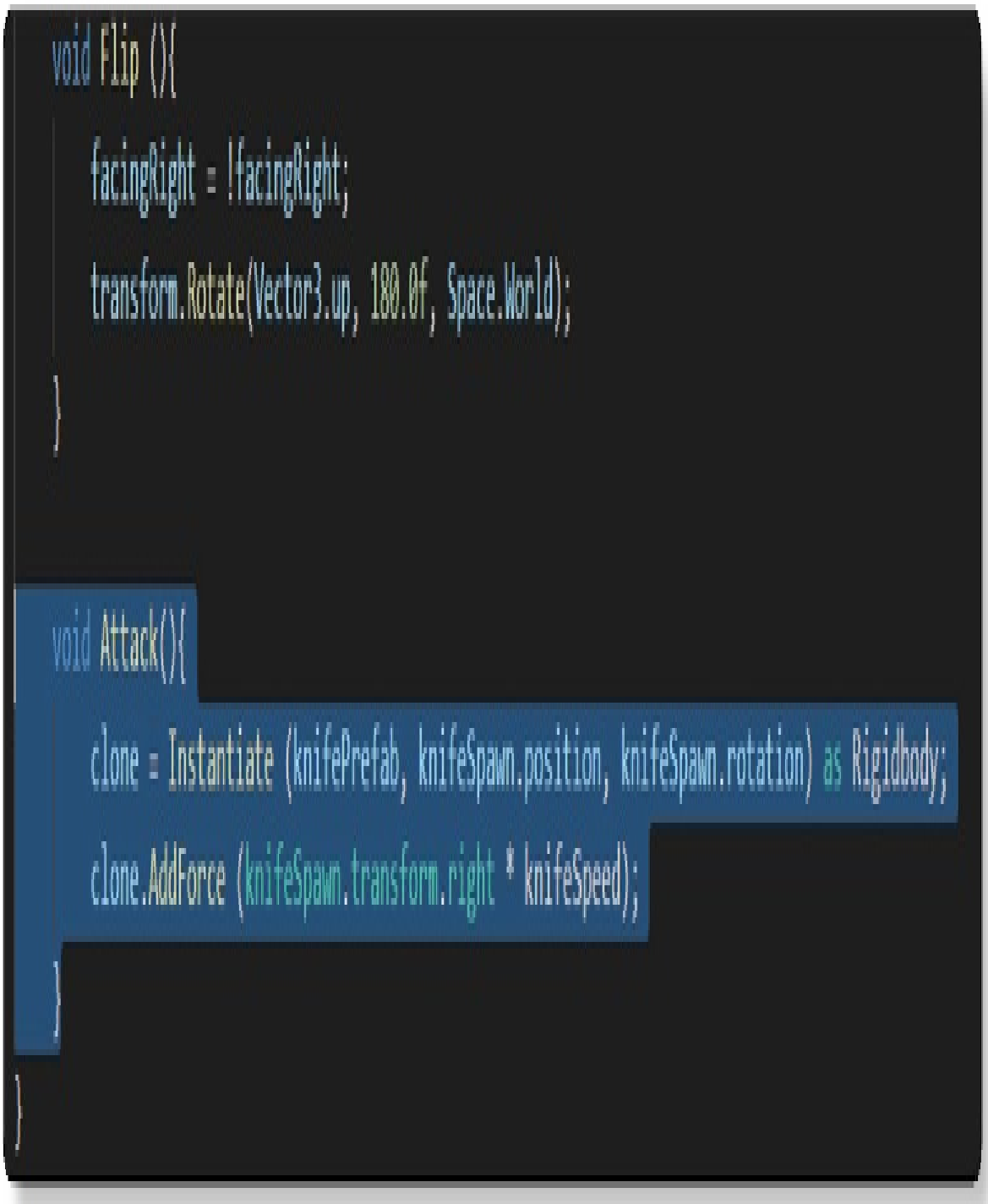
Elimine el elemento “Knife_PFB” de la ventana de “Hierarchy”.

Vaya al script “CharacterMovement”, haga las siguientes variables e inicialice.

```
public LayerMask whatIsGround;

public float knifeSpeed = 600.0f;
public Transform knifeSpawn;
public Rigidbody knifePrefab;
Rigidbody clone;

// Use this for initialization
void Start () {
    rigidbody = GetComponent<Rigidbody> ();
    anim = GetComponent<Animator>();
    groundCheck = GameObject.Find ("GroundCheck").transform;
    knifeSpawn = GameObject.Find ("KnifeSpawn").transform;
}
```

Creación e inicialización de variables

Escriba la siguiente función “Attack” debajo de la función “Flip”.



```
    void Flip (){
        facingRight = !facingRight;
        transform.Rotate(Vector3.up, 180.0f, Space.World);
    }


    void Attack(){
        clone = Instantiate (knifePrefab, knifeSpawn.position, knifeSpawn.rotation) as Rigidbody;
        clone.AddForce (knifeSpawn.transform.right * knifeSpeed);
    }
}
```

Creación de función “Attack”

En esta parte del código estamos asignando un valor a la variable “clone”.

Este valor será una instancia (copia con herencia) en base al “rigidBody” de la variable “knifePrefab” conforme a la posición y rotación del objeto “KnifeSpawn” y será tratado como un RigidBody.

Seguido a asignarle una fuerza hacia la derecha multiplicada por el valor de la variable “knifeSpeed”.

Vaya a la función “FixedUpdate” y escriba el siguiente código para mandar llamar la función creada cuando se oprima el botón izquierdo del ratón.

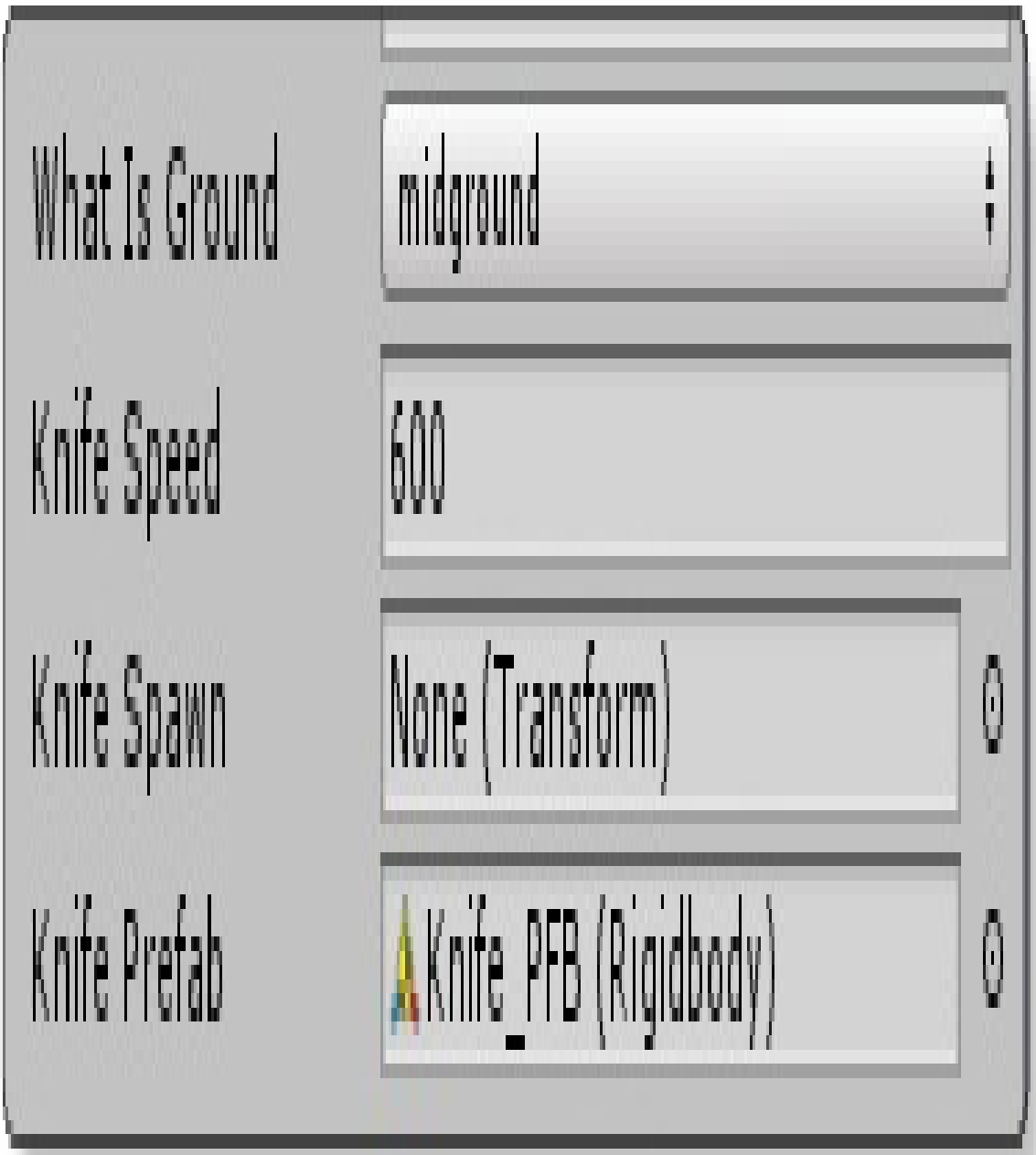
```
void FixedUpdate (){
    rigidbody.velocity = new Vector2 (moveDirection * ma
    grounded = Physics2D.OverlapCircle (groundCheck.posit

    if (moveDirection > 0.0f && !facingRight){
        Flip ();
    } else if (moveDirection < 0.0f && facingRight) {
        Flip ();
    }
    anim.SetFloat ("Speed", Mathf.Abs (moveDirection));

    if (Input.GetButtonDown ("Fire1")){
        Attack();
    }
```

Programación del ataque

El comando “Fire1” ya es una acción preestablecida que está ligada al botón izquierdo del ratón.

Salve el script

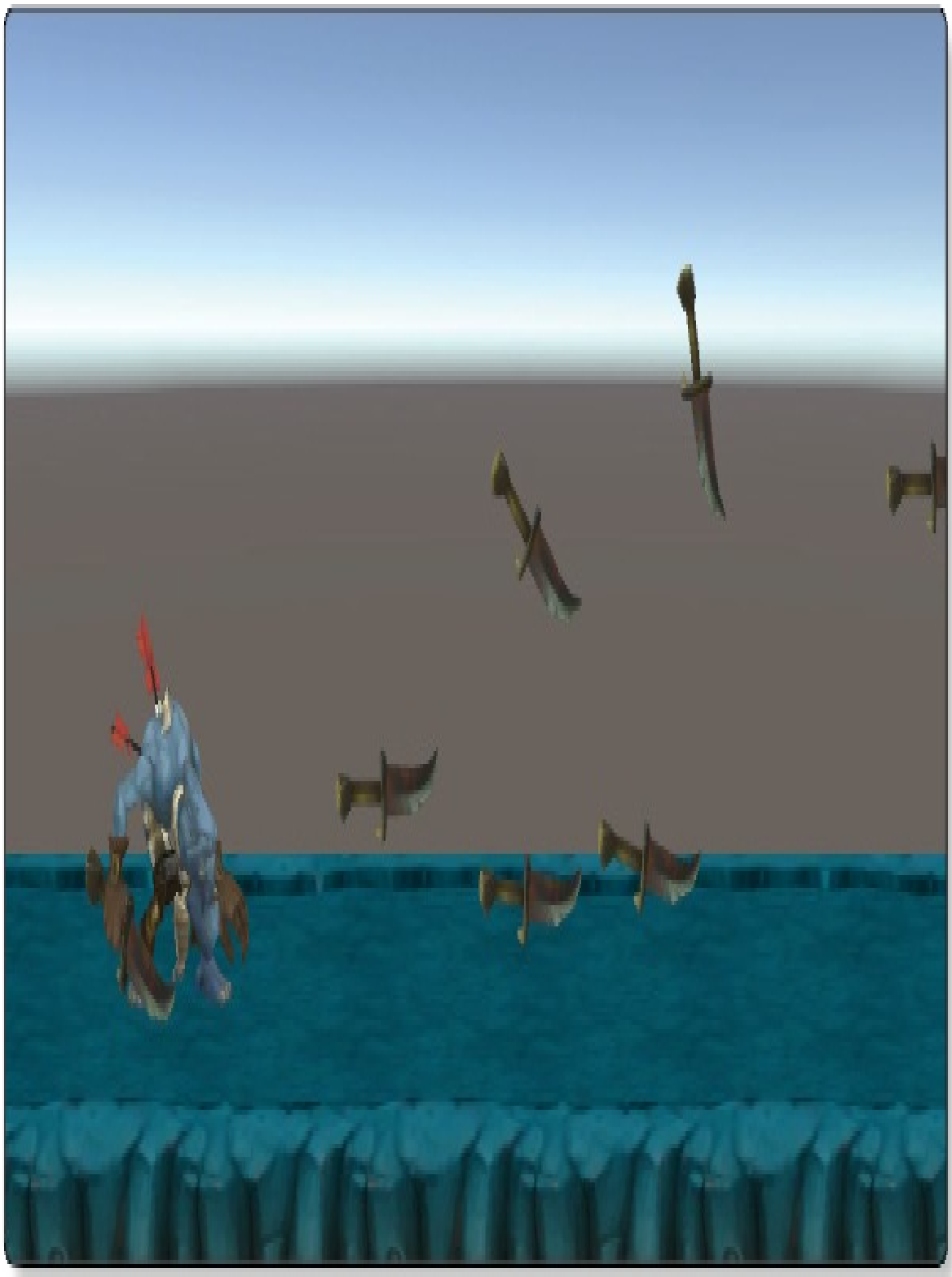
En Unity seleccione “Player” y arrastre el prefab “Knife_PFB” del folder “Prefabs” a la ventana de “Inspector” en la variable “Knife Prefab”


What Is Ground
midground
Knife Speed
600
Knife Spawn
None (Transform)
Knife Prefab
Knife_PFB (Rigidbody)

Asignación del Prefab

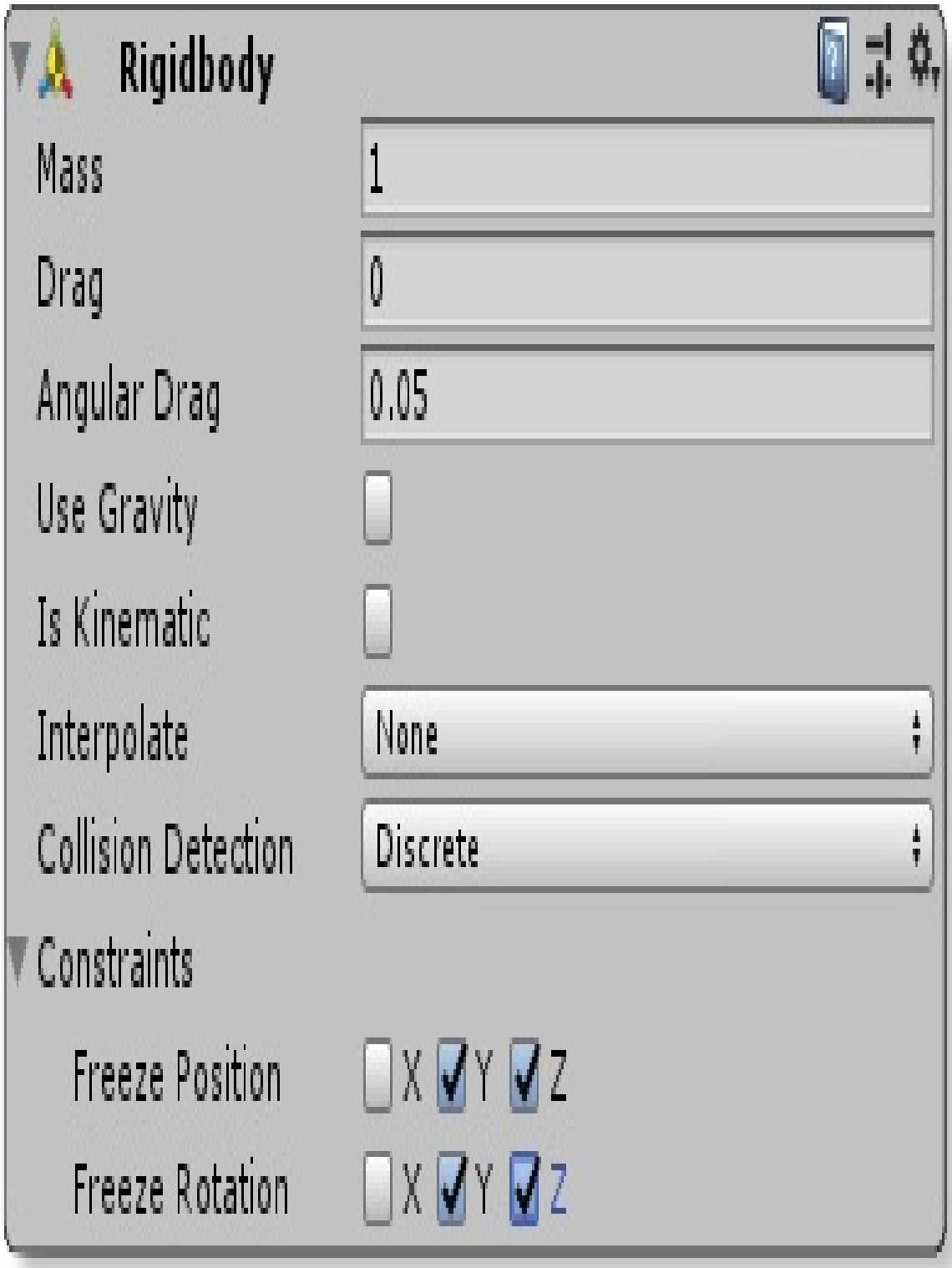
Previsualice el juego para observar como dispara los proyectiles el “Player”

PROBLEMA 1: Al brincar y disparar, algunos proyectiles cambian su rotación



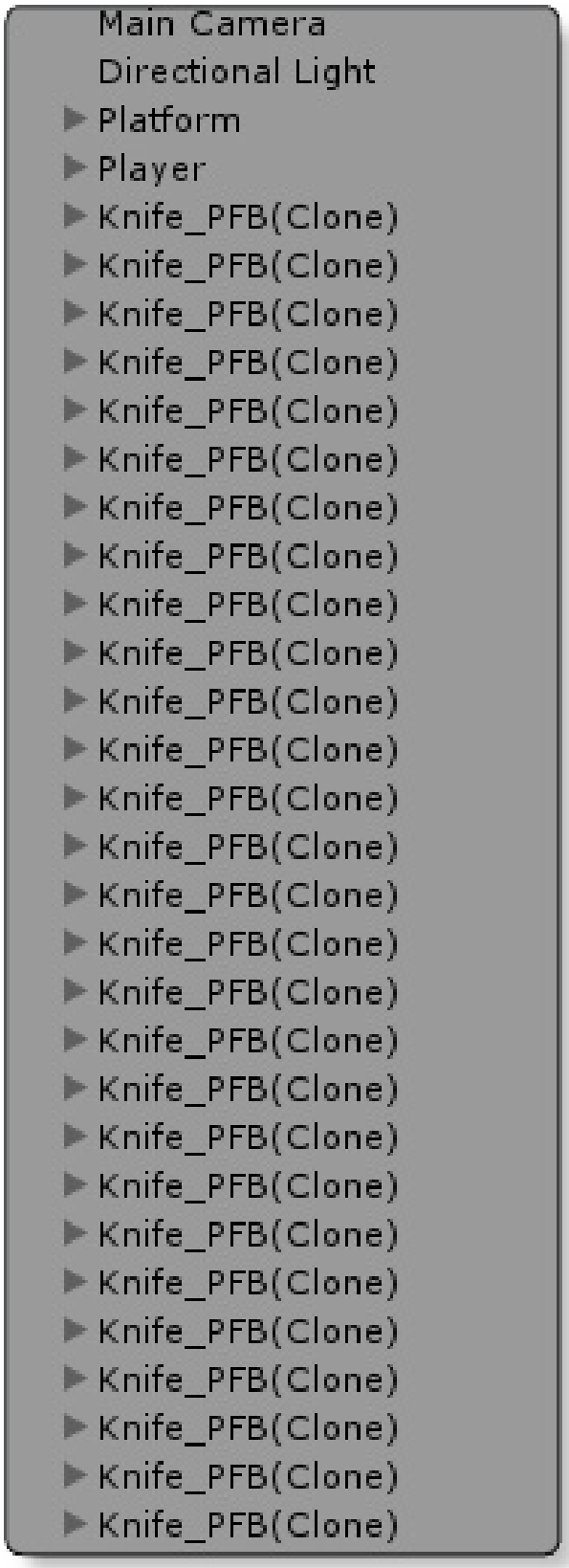
Problema 1 de disparo

SOLUCIÓN: Seleccione el “Prefab” del proyectil y en su componente del “RigidBody” congele sus rotaciones y posiciones.


Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Z
Freeze Rotation
X
Y
Z

Congelamiento de transformaciones

PROBLEMA 2: A medida que se disparan los proyectiles se van creando instancias del mismo en nuestro juego y las podemos visualizar en la ventana “Hierarchy”.


Main Camera
Directional Light
Platform
Player
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)
Knife_PFB(Clone)

Problema 2 de disparo

SOLUCIÓN: Debemos de crear un script para destruir los proyectiles después de determinado tiempo

Vaya a la carpeta de “Scripts”, haga un nuevo script C# y renómbrelo “KnifeDestruction”

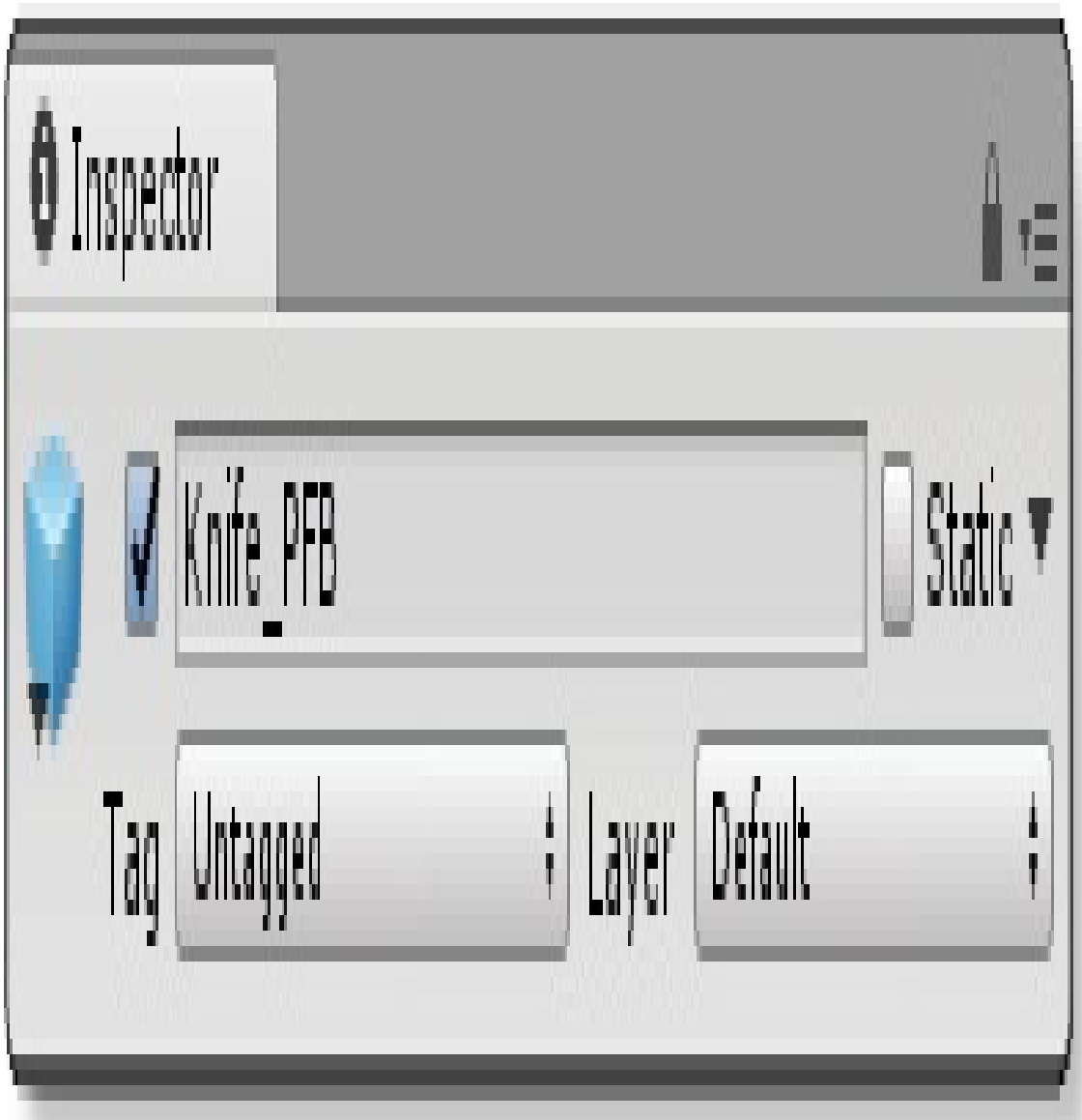
Abra el script y escriba el siguiente código:

public class KnifeDestruction : MonoBehaviour {
public float lifeSpan = 2.0f;
// Use this for initialization
void Start () {
Destroy (gameObject, lifeSpan);
}

Script para destrucción

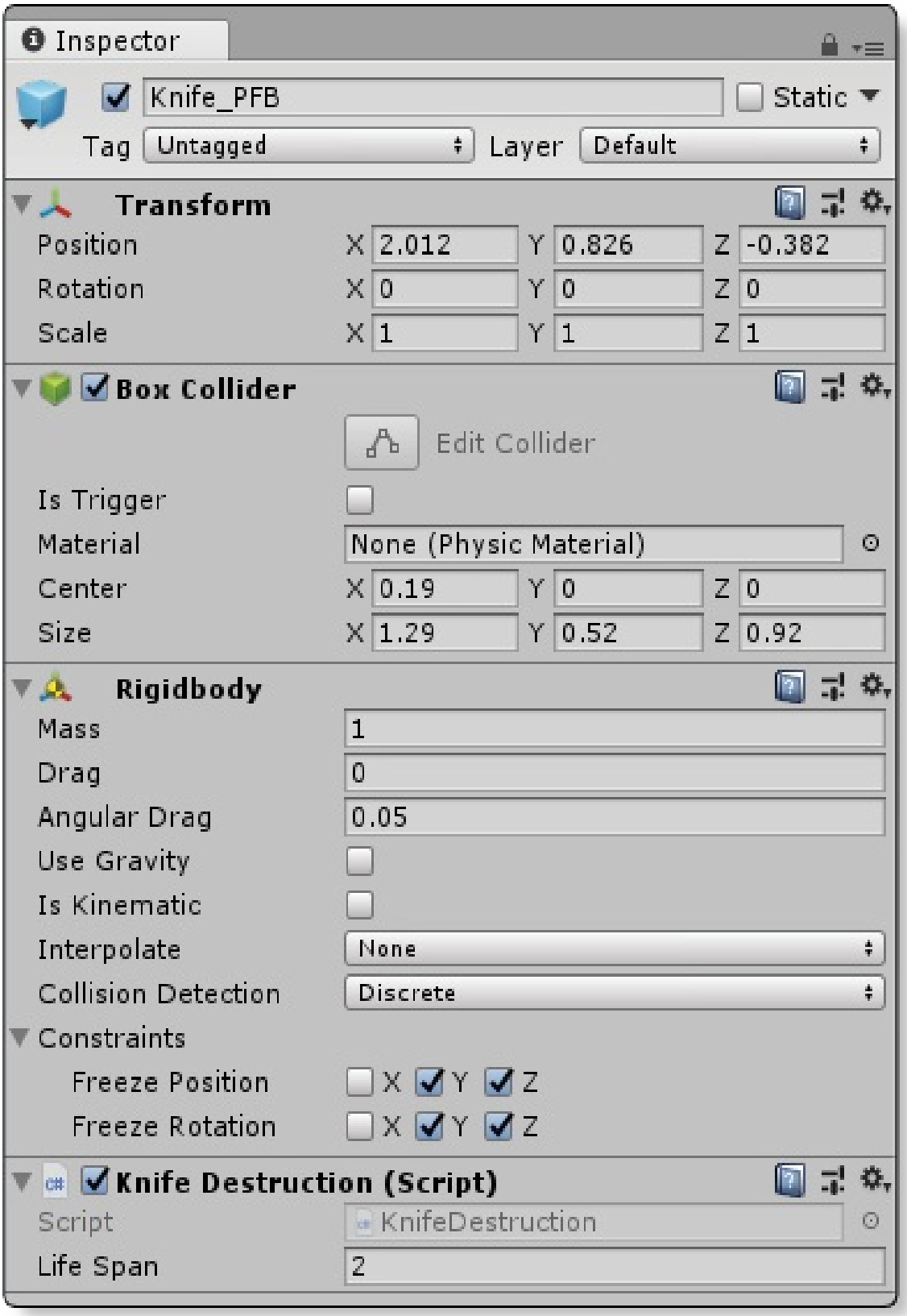
NOTA:

Cuando queremos mantener la ventana de “Hierarchy” bloqueada para que no cambie a momento de seleccionar otro elemento, hacemos clic en el icono del candado en la parte superior derecha de la ventana “Hierarchy”.


Inspector
Knife_PFB
Static
Tag
Untagged
Layer
Default

Bloqueo de Inspector

Asigne el script creado al objeto “Knife_PFB” seleccionando “Knife_PFB”, bloquear la ventana de “Inspector”, seleccionar el script y arrastrarlo como componente.


Inspector
Knife_PFB
Static
Tag
Untagged
Layer
Default
Transform
Position
X 2.012
Y 0.826
Z -0.382
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0.19
Y 0
Z 0
Size
X 1.29
Y 0.52
Z 0.92
Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Z
Freeze Rotation
X
Y
Z
Knife Destruction (Script)
Script
KnifeDestruction
Life Span
2

Asignación de script

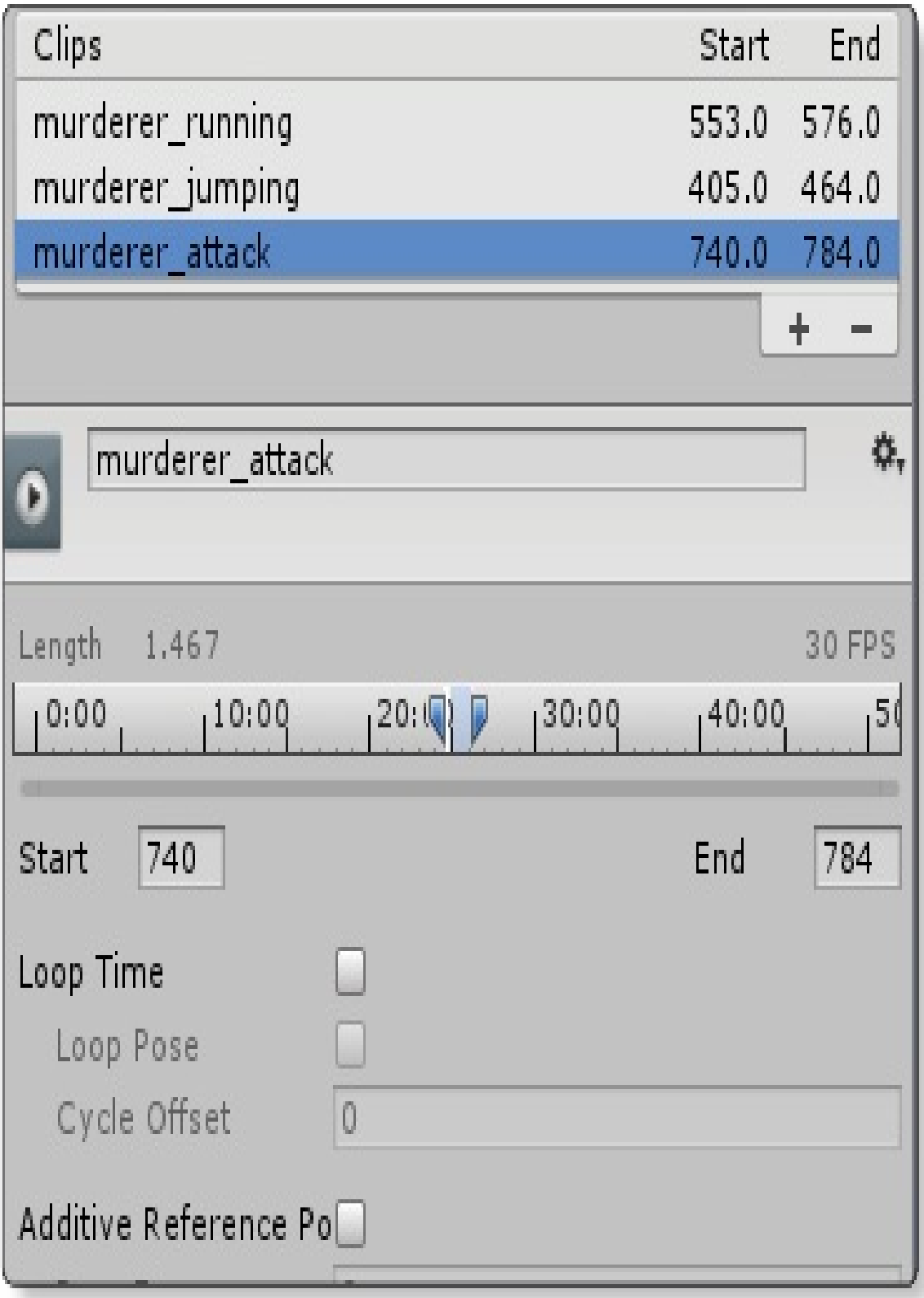
Previsualice el juego para comprobar que las instancias se destruyen después de 2 segundos.

PROGRAMACIÓN DE LA ANIMACIÓN DE ATAQUE DEL PLAYER

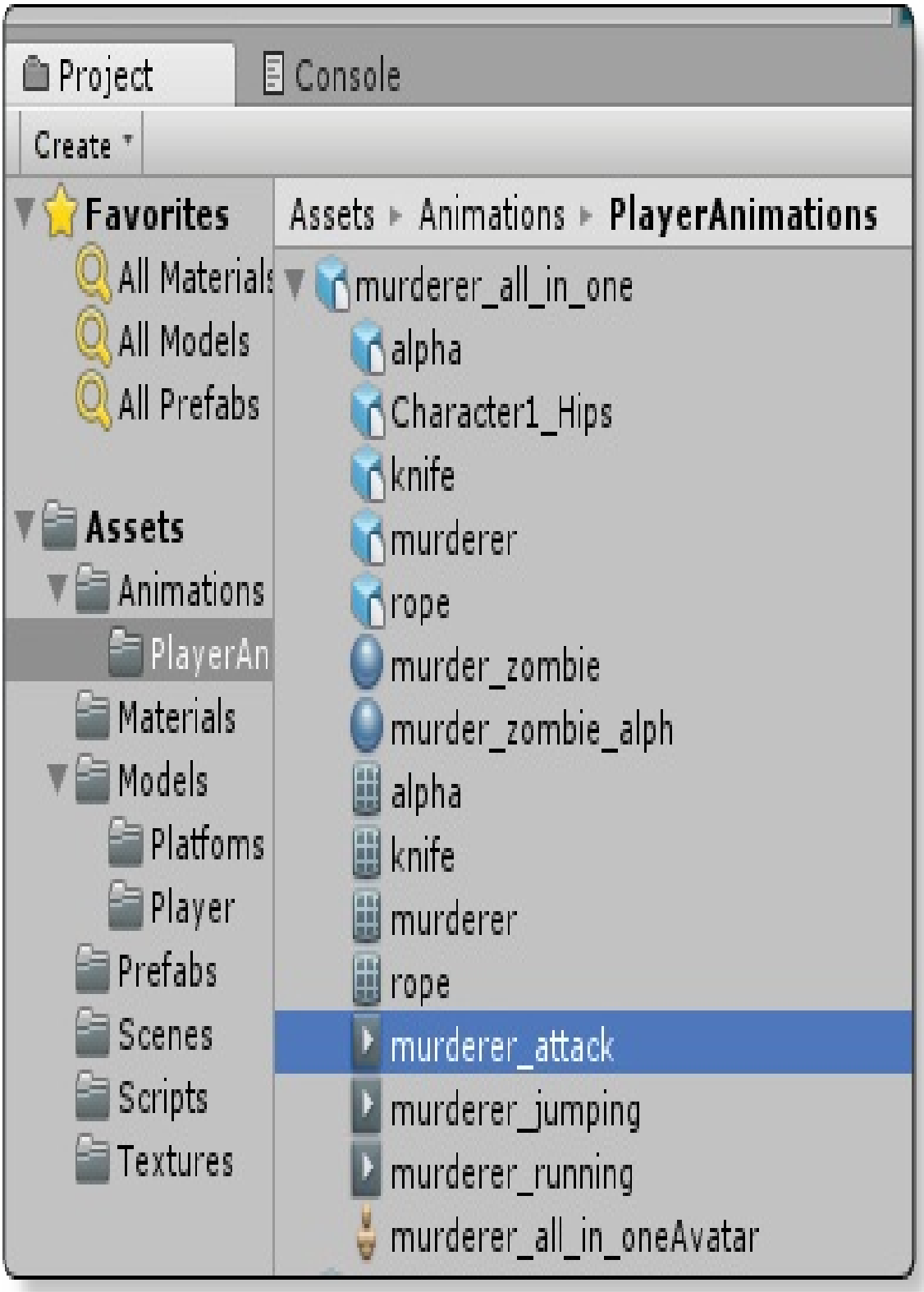
En este apartado programaremos la animación de ataque de nuestro Player y reprogramaremos su función de ataque a través de un evento de animación (Animation Event) y revisaremos un concepto nuevo: “Avatar Mask” para combinar animaciones.

Con los conceptos aprendidos haga el clip de animación para el ataque del Player en base a:

frame: 0740-0784 slash.

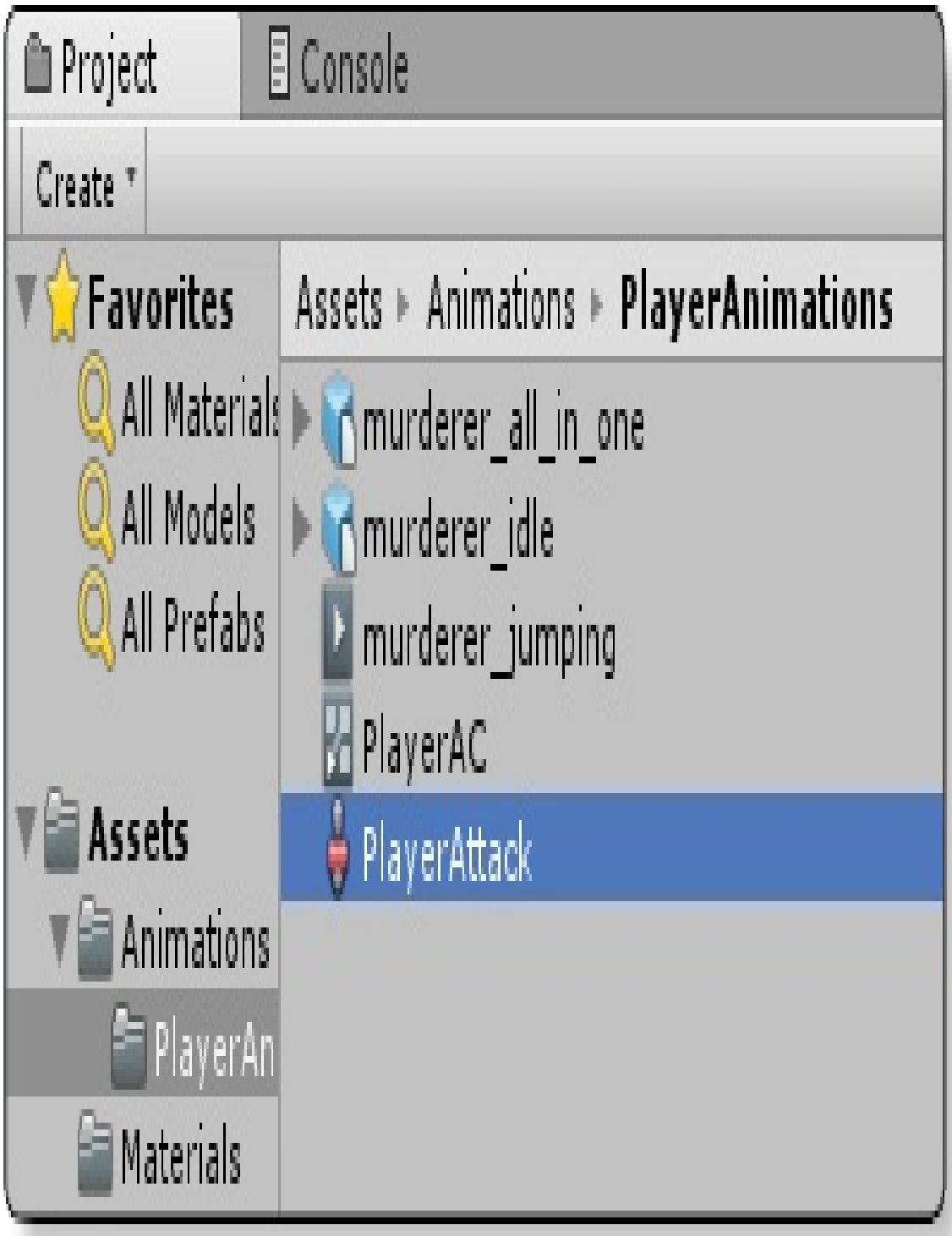

Clips
Start
End
murderer_running
553.0
576.0
murderer_jumping
405.0
464.0
murderer_attack
740.0
784.0
+
-
murderer_attack
Length
1.467
30 FPS
0:00
10:00
30:00
40:00
Start
740
End
784
Loop Time
Loop Pose
Cycle Offset
0
Additive Reference Po

Creación de clip de animación de ataque


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platfoms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Animations ▸ PlayerAnimations
murderer_all_in_one
alpha
Character1_Hips
knife
murderer
rope
murder_zombie
murder_zombie_alph
alpha
knife
murderer
rope
murderer_attack
murderer_jumping
murderer_running
murderer_all_in_oneAvatar

Clip de ataque creado

En la ventana del proyecto vaya a la carpeta “PlayerAnimations”, clic derecho y asigne “create/ Avatar Mask” y renómbrela “PlayerAttack”

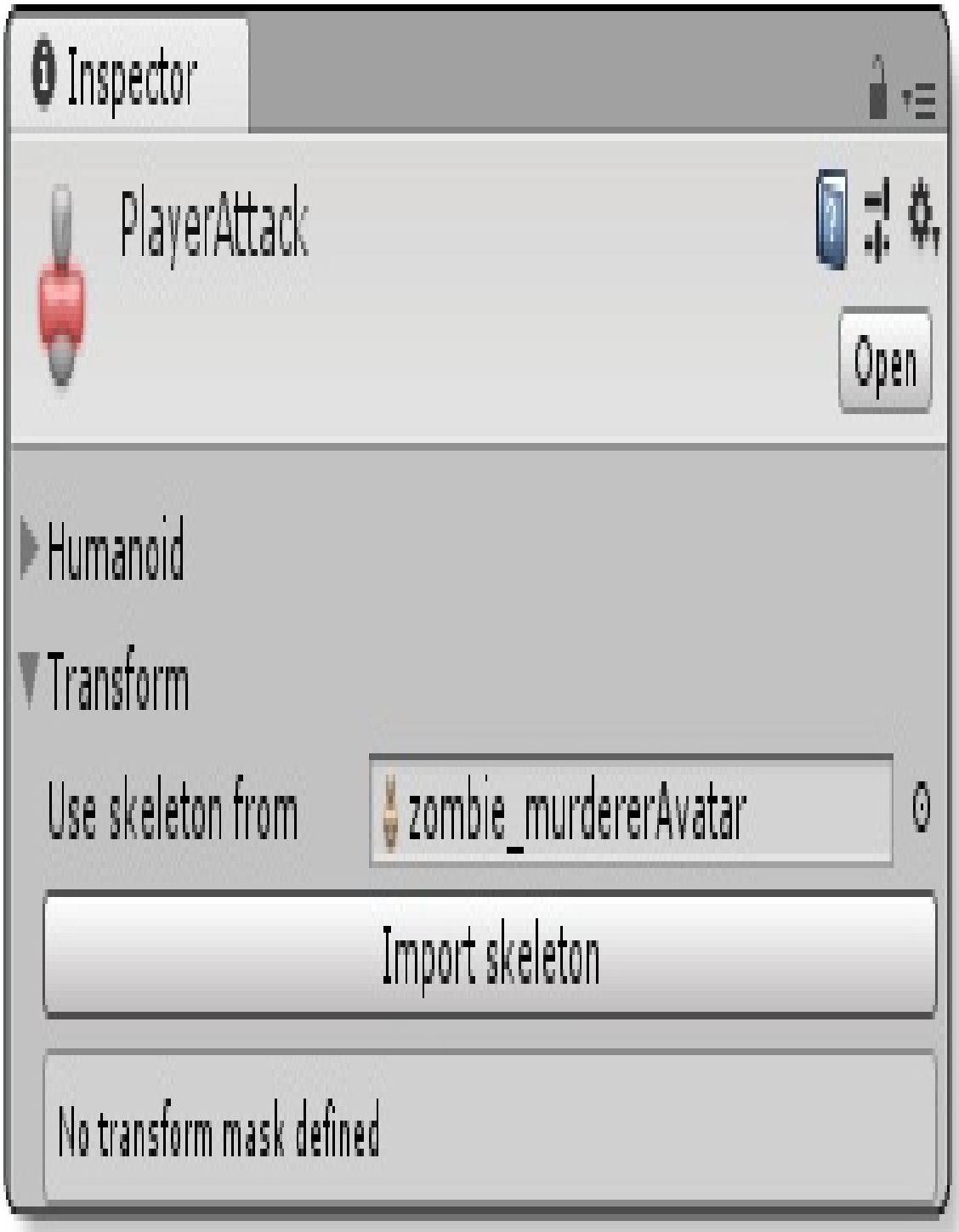

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Assets ▸ Animations ▸ PlayerAnimations
murderer_all_in_one
murderer_idle
murderer_jumping
PlayerAC
PlayerAttack

Avatar Mask

Un “Avatar Mask” nos permite deseleccionar algunos huesos o “joints” del Rig de nuestro “Player” para poder enmascararlos y que solo corra la animación nueva en los “joints” seleccionados.

En este caso queremos aplicar la animación de ataque únicamente del tronco hacia arriba de nuestro “Player” para que se pueda visualizar la animación de ataque cuando se encuentre en estado “Idle” o cuando se encuentre corriendo o saltando.

En Inspector expanda “Transform” y asigne el “Avatar” del personaje en “Use skeleton from”.

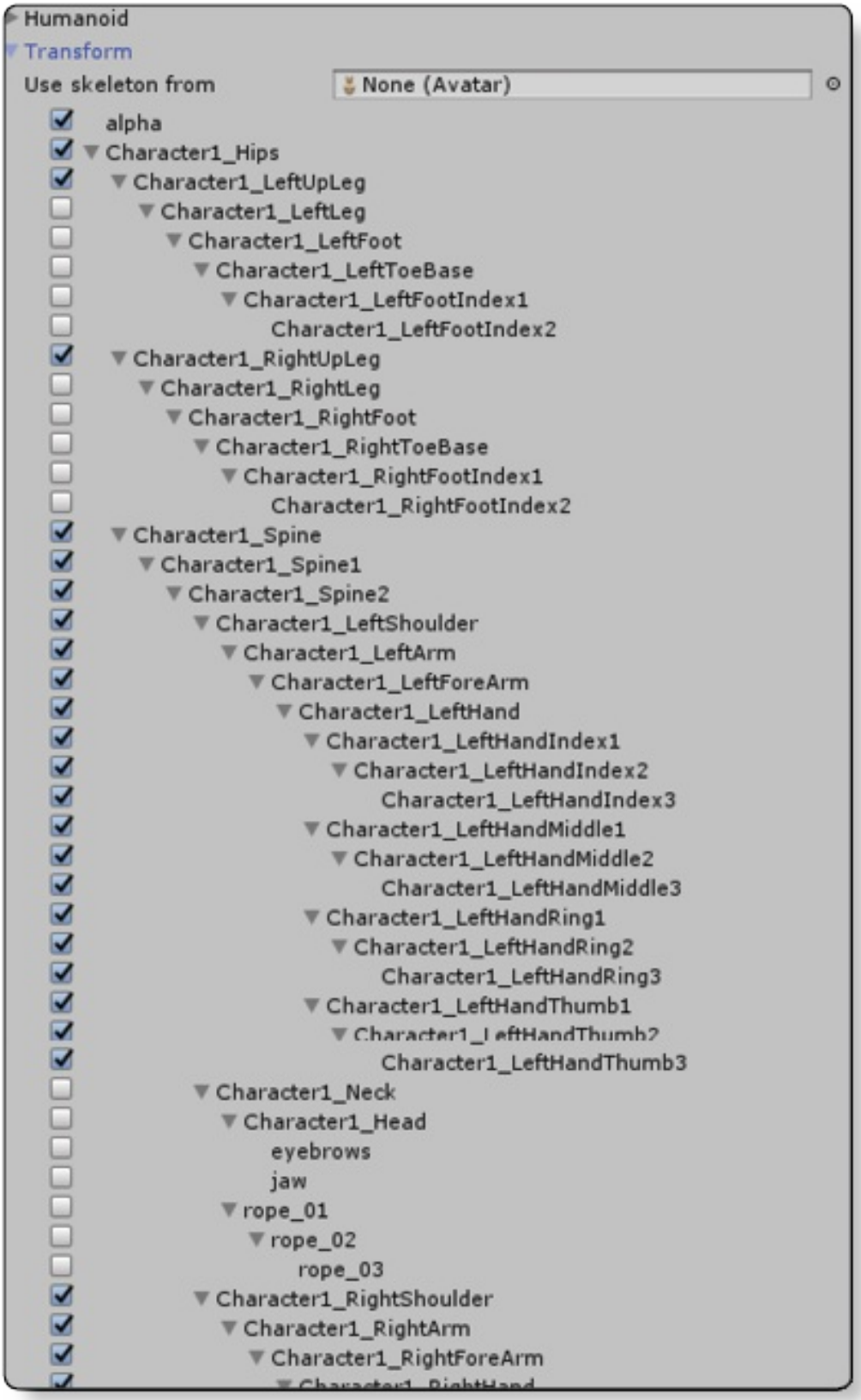

Inspector
PlayerAttack
Open
Humanoid
Transform
Use skeleton from
zombie_murdererAvatar
Import skeleton
No transform mask defined

Asignación de Rig

Nuestro modelo de “zombie_murderer” contiene un “Rig” de animación creado externamente. Es necesario asignar como uso de esqueleto su propio Avatar

Clic en “Import skeleton”

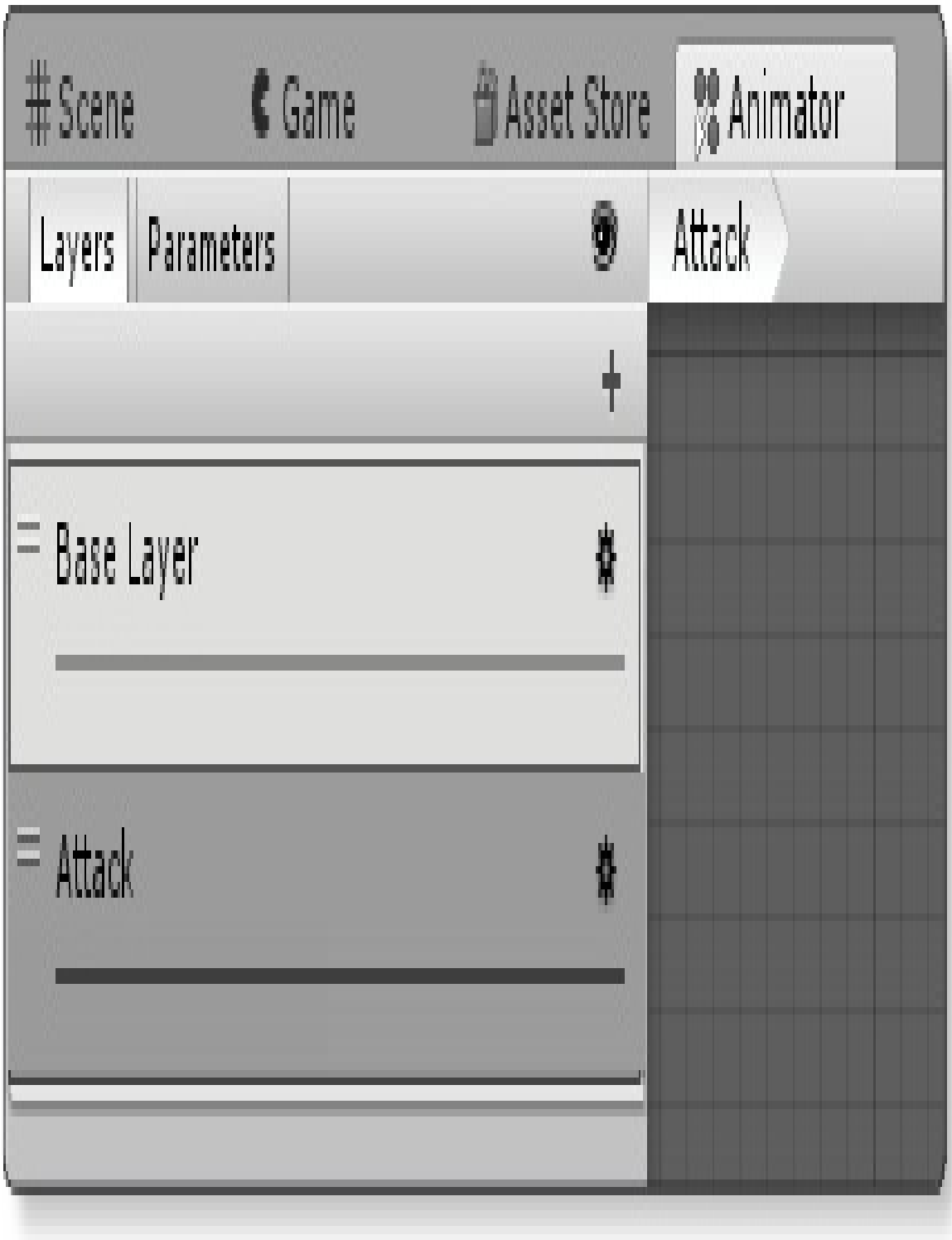
De la lista que se extiende, desmarque las opciones que se muestran..


Humanoid
Transform
Use skeleton from
None (Avatar)
alpha
Character1_Hips
Character1_LeftUpLeg
Character1_LeftLeg
Character1_LeftFoot
Character1_LeftToeBase
Character1_LeftFootIndex1
Character1_LeftFootIndex2
Character1_RightUpLeg
Character1_RightLeg
Character1_RightFoot
Character1_RightToeBase
Character1_RightFootIndex1
Character1_RightFootIndex2
Character1_Spine
Character1_Spine1
Character1_Spine2
Character1_LeftShoulder
Character1_LeftArm
Character1_LeftForeArm
Character1_LeftHand
Character1_LeftHandIndex1
Character1_LeftHandIndex2
Character1_LeftHandIndex3
Character1_LeftHandMiddle1
Character1_LeftHandMiddle2
Character1_LeftHandMiddle3
Character1_LeftHandRing1
Character1_LeftHandRing2
Character1_LeftHandRing3
Character1_LeftHandThumb1
Character1_LeftHandThumb3
Character1_Neck
Character1_Head
eyebrows
jaw
rope_01
rope_02
rope_03
Character1_RightShoulder
Character1_RightArm
Character1_RightForeArm

Asignación de Joints para Avatar Mask

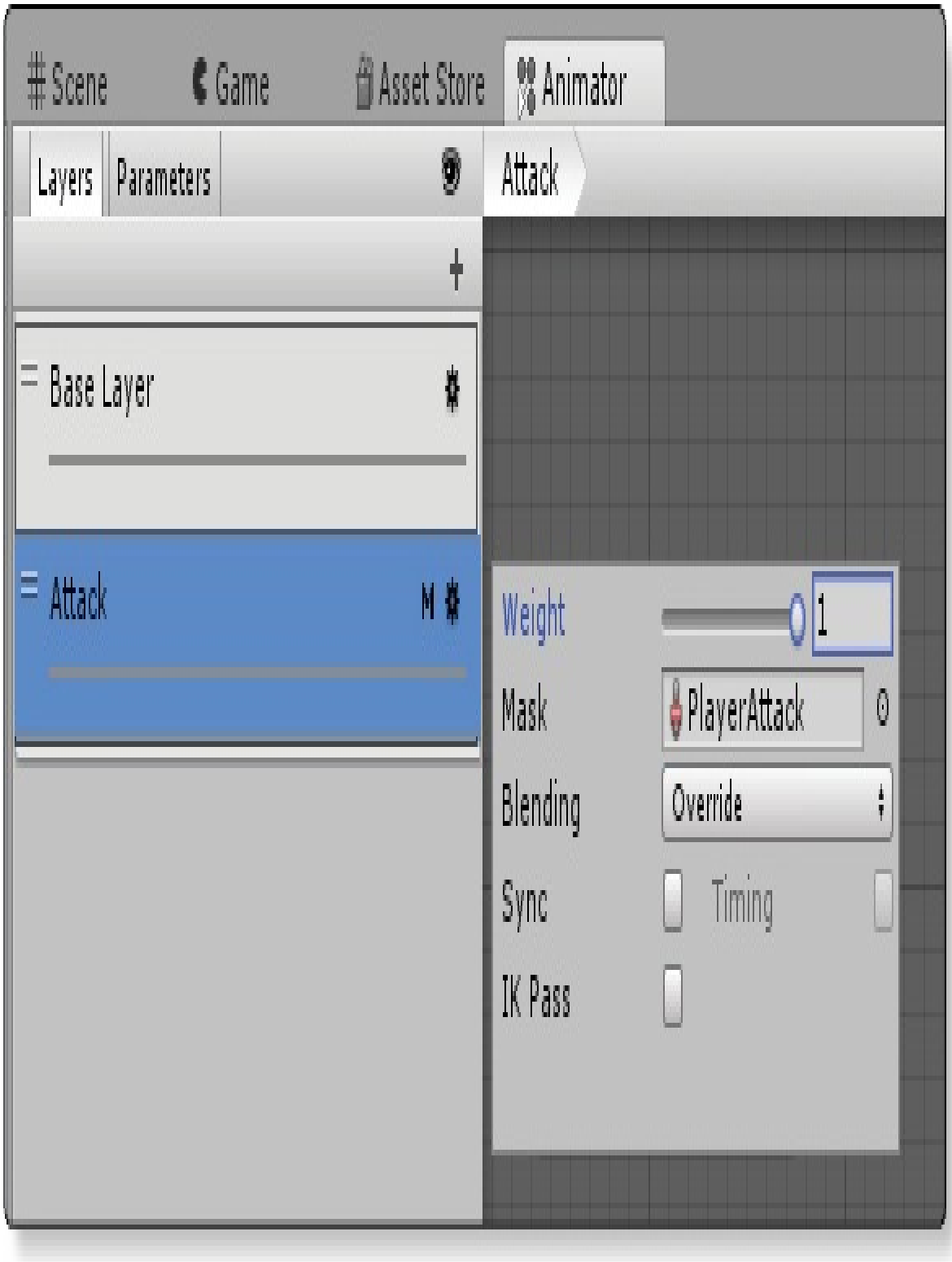
Seleccione “Player” y abra la ventana “Animator”

Haga un nuevo layer en la ventana “Animator” (clic en el símbolo +) y nómbrelo “Attack”


Scene
Game
Asset Store
Animator
Layers
Parameters
Attack
Base Layer
Attack

Creación de un nuevo layer de Animator

Clic en su icono de opciones (el icono de engrane) y asigne los siguientes valores.

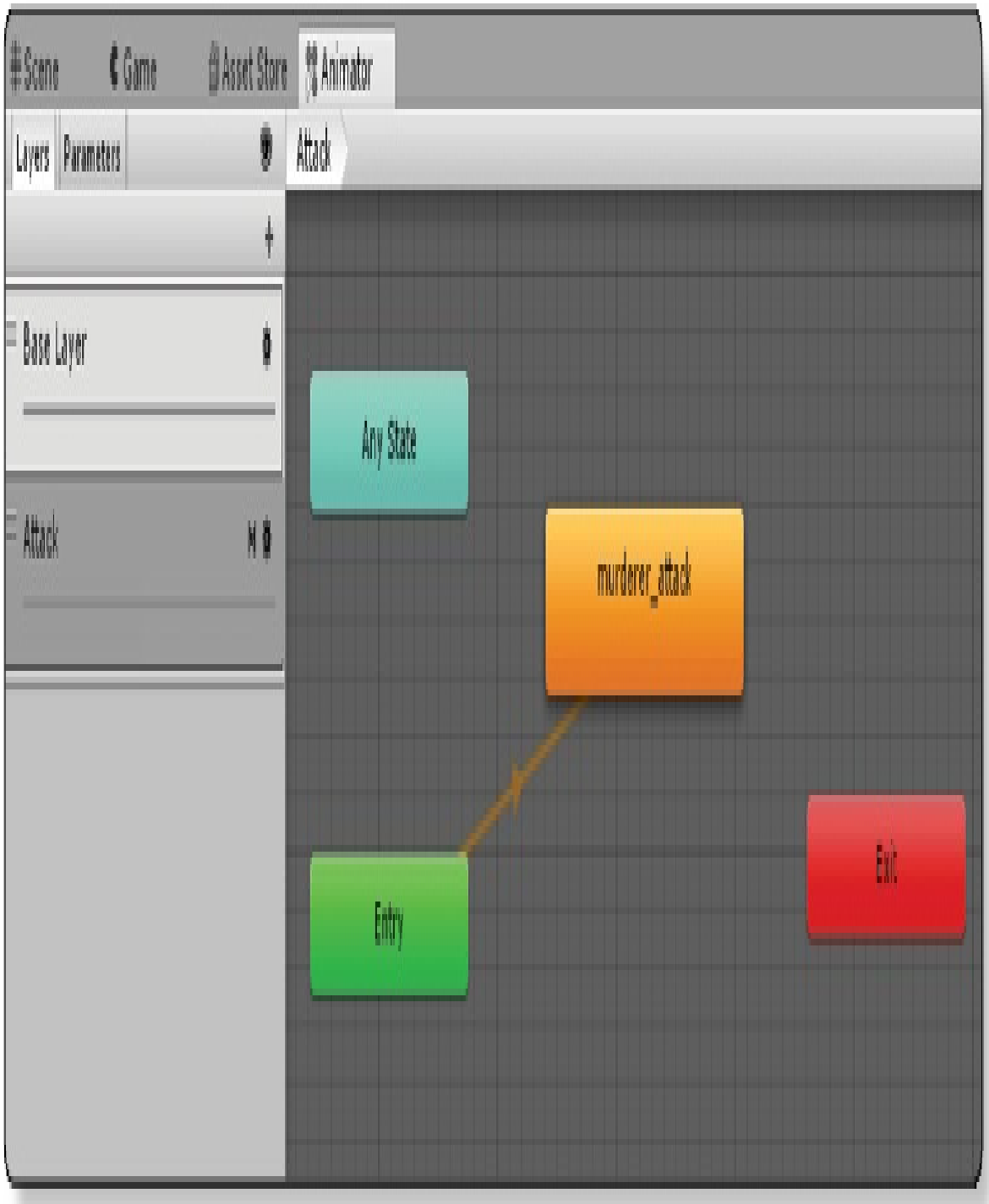

Scene
Game
Asset Store
Animator
Layers
Parameters
Attack
Base Layer
Attack
M
Weight
1
Mask
PlayerAttack
Blending
Override
Sync
Timing
IK Pass

Parámetros del nuevo layer de Animator

En “Mask” asignamos el “Layer Mask” creado anteriormente

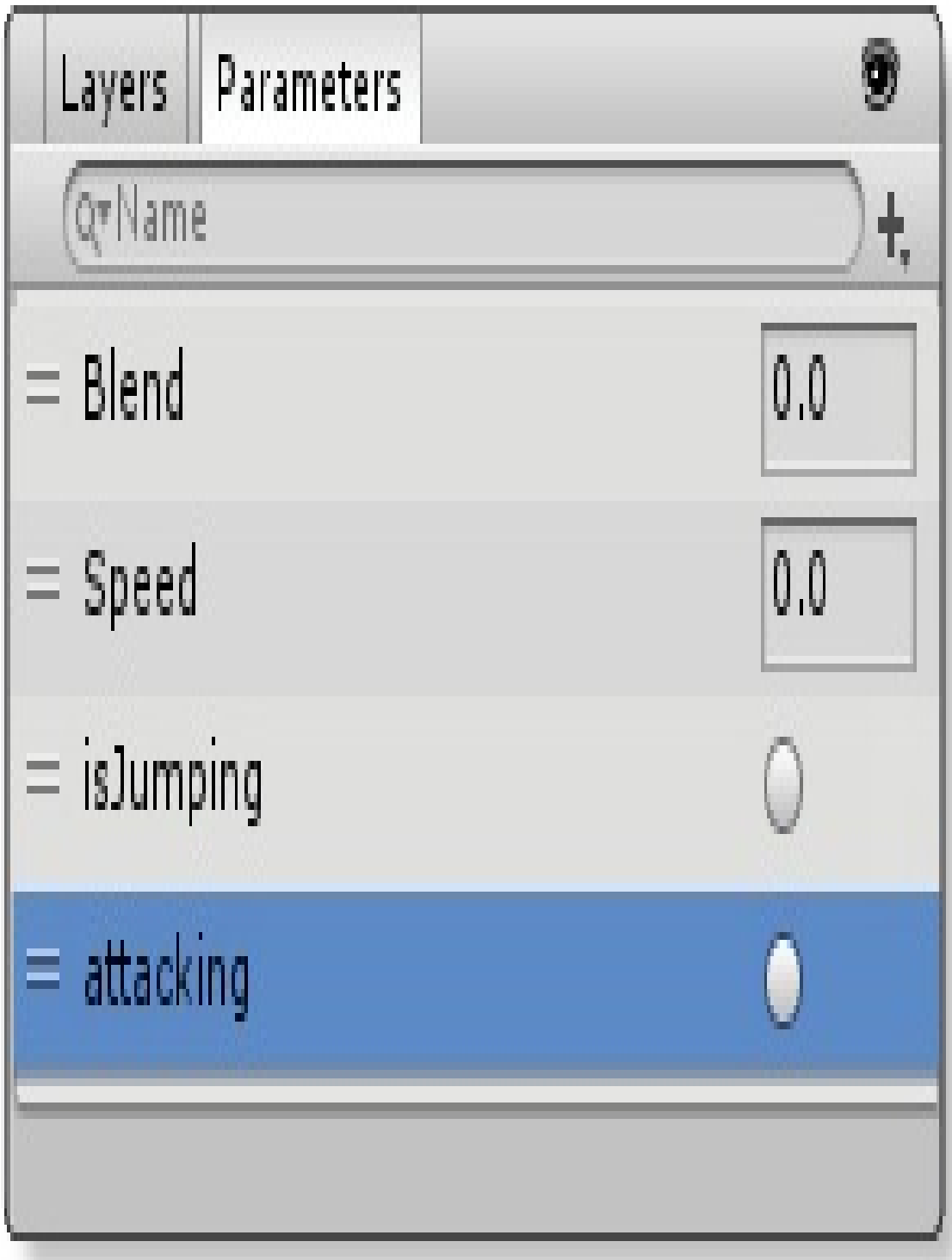
El parámetro de “Weight” nos indica del cero al 1 el nivel de la combinación de nuestra animación.

En el nuevo layer “Attack” coloque la animación del ataque.


Scene
Game
Asset Store
Animator
Layers
Parameters
Attack
Base Layer
Attack
Any State
murderer_attack
Entry
Exit

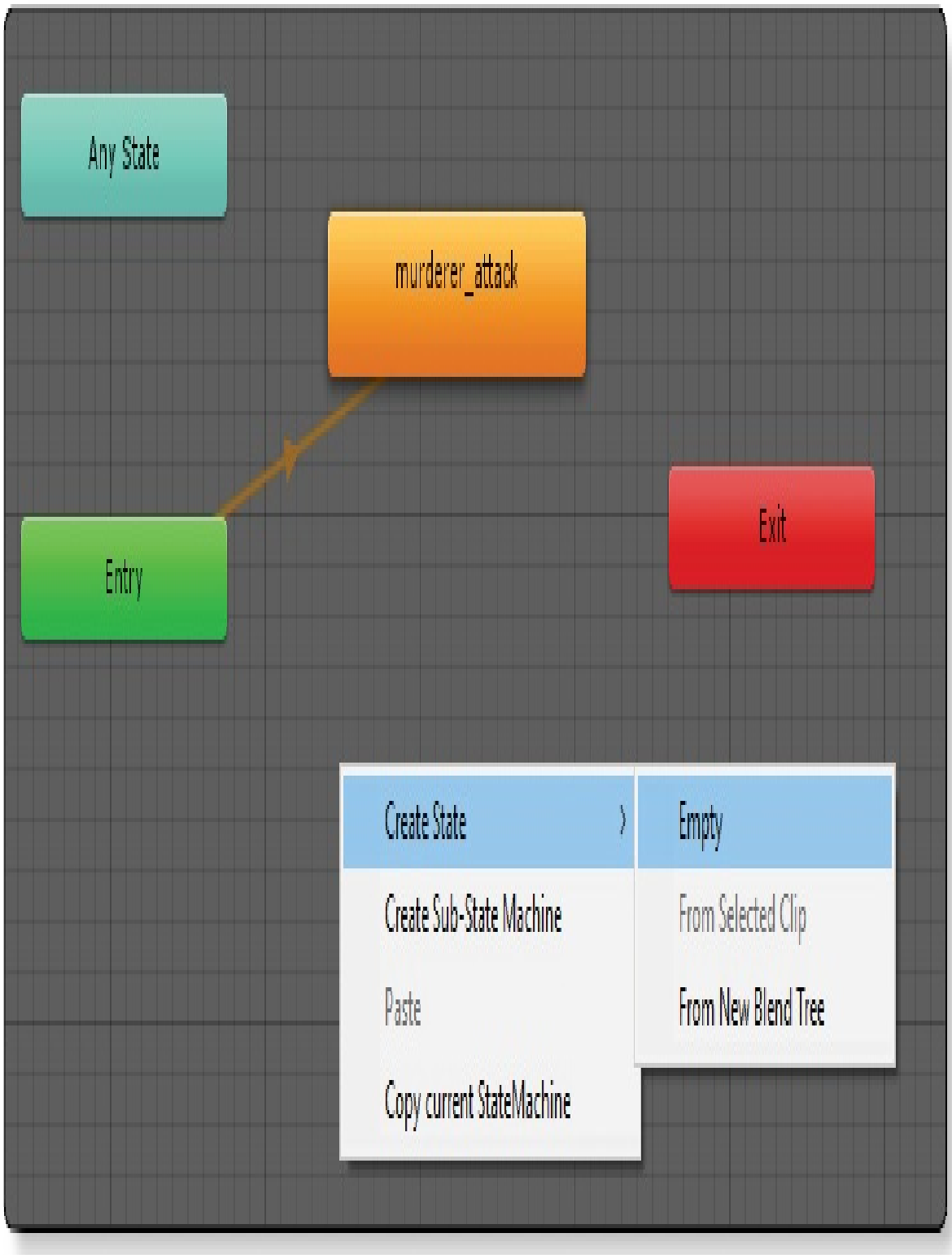
Asignación de animación de ataque

Haga un nuevo parámetro tipo “Trigger y nómbrelo “attacking”.


Layers
Parameters
Q▾Name
Blend
0.0
Speed
0.0
isJumping
attacking

Parámetro tipo Trigger

Haga un nuevo “EmptyState” y asigne su estado “default” con clic derecho.


Any State
murderer_attack
Entry
Exit
Create State
Empty
Create Sub-State Machine
From Selected Clip
Paste
From New Blend Tree
Copy current StateMachine

Creación de un nuevo “EmptyState”

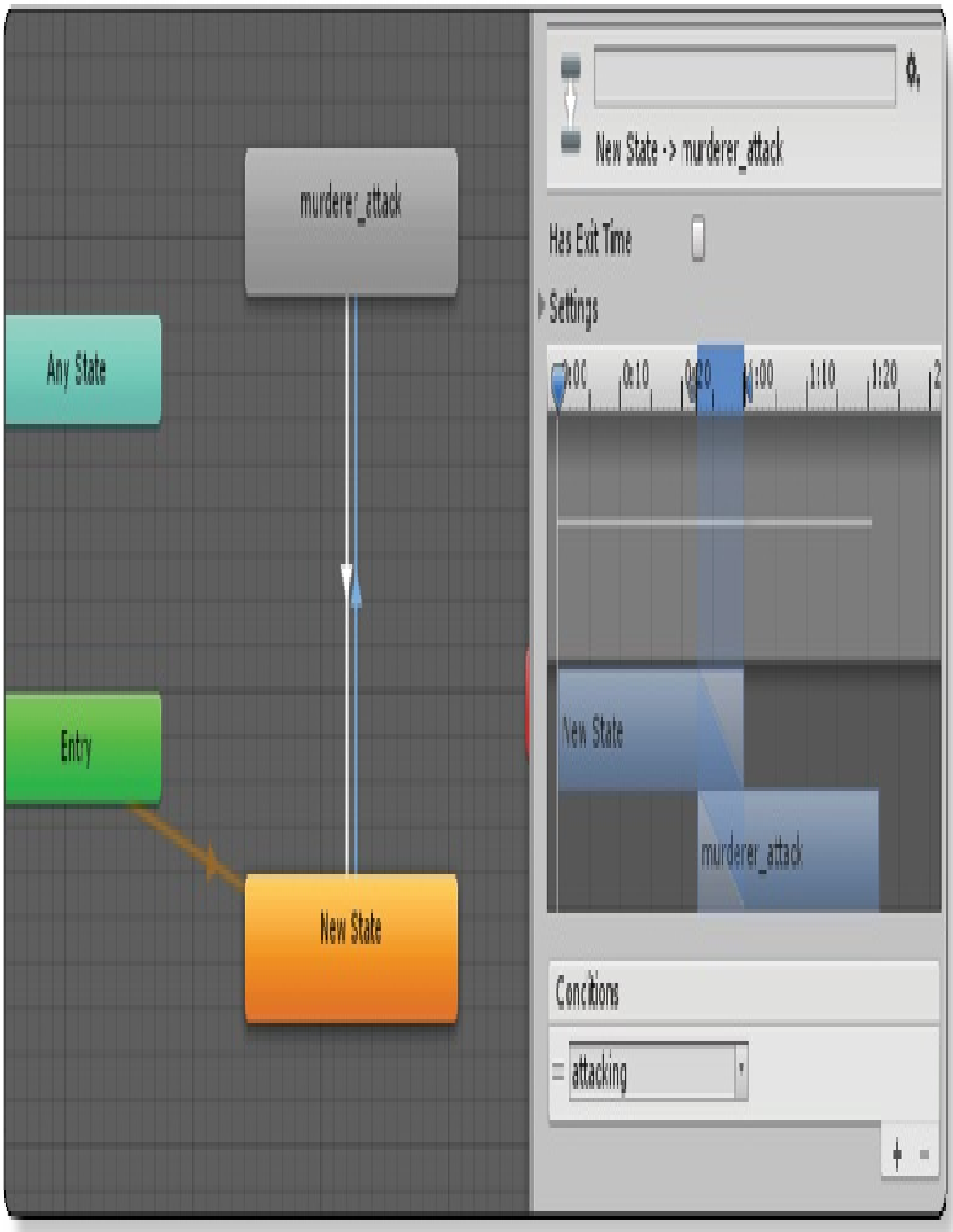
Any State
murderer_attack
Exit
Entry
New State
Make Transition
Set as Layer Default State
Copy
Create new BlendTree in State
Delete

Estado “Default”

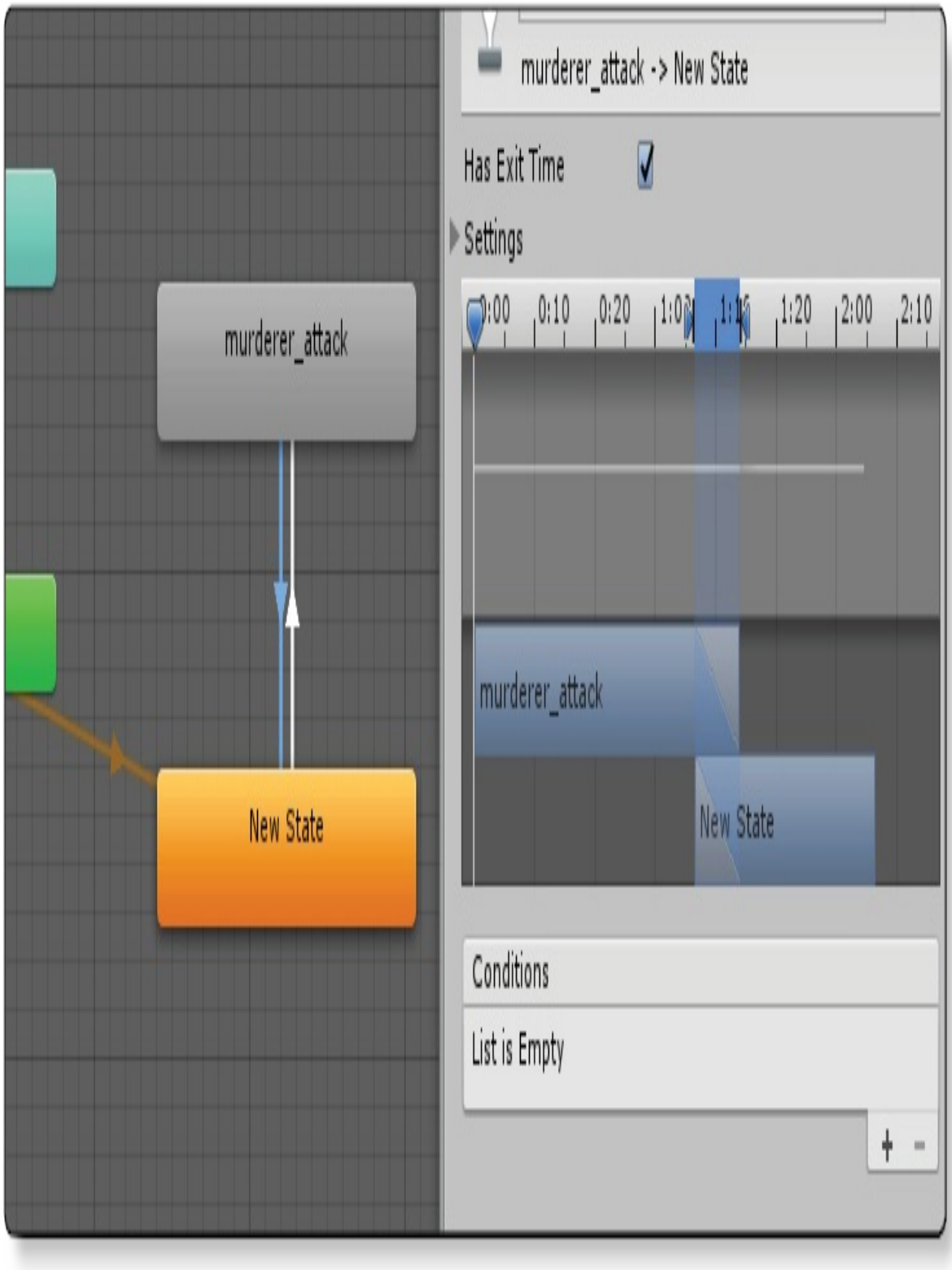
Haga las transiciones que se muestran y en la transición de ida asigne el parámetro “attacking” como condición

NOTA:

En la de vuelta no asignar condición.

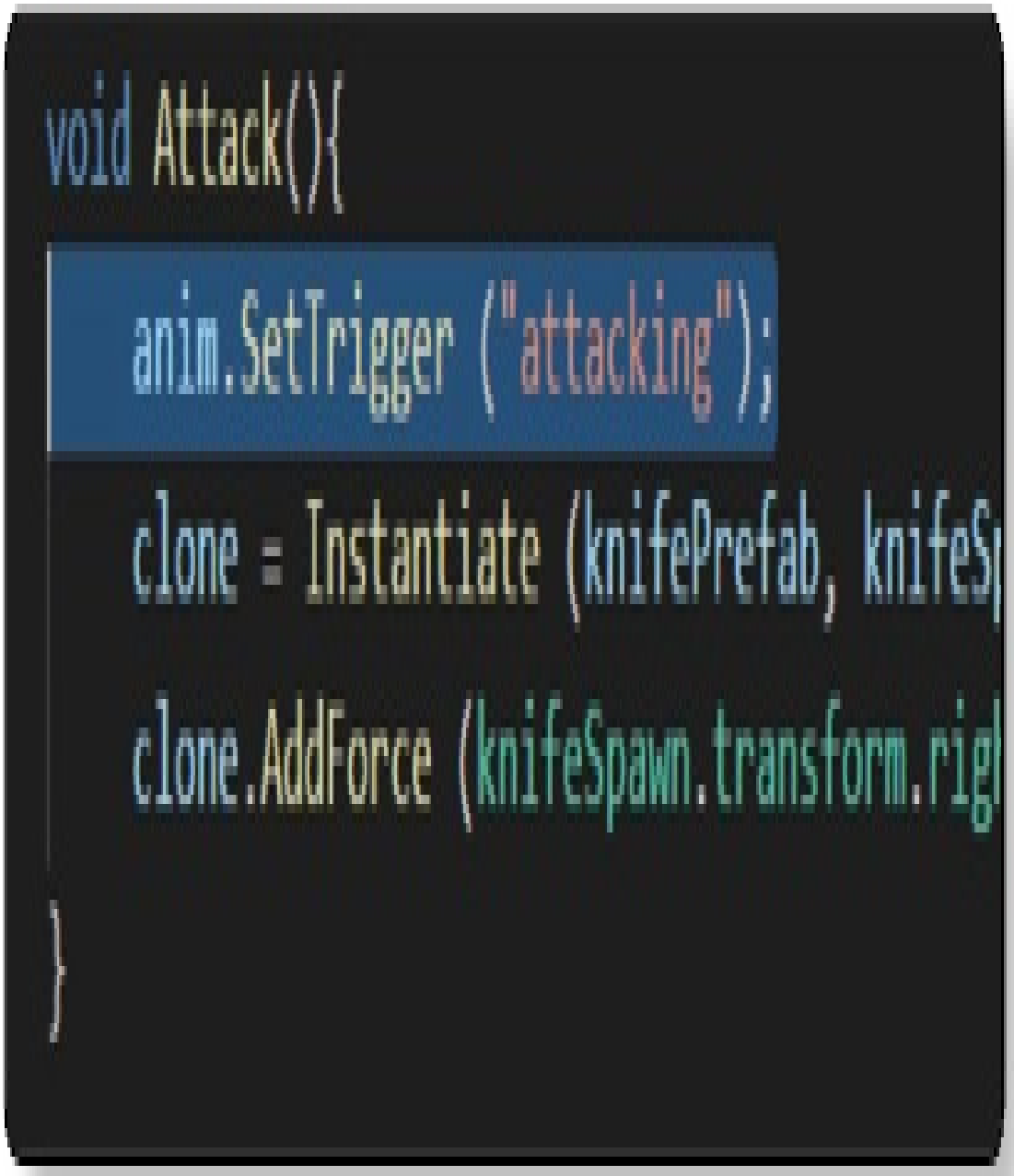

murderer_attack
Any State
Entry
New State
New State -> murderer_attack
Has Exit Time
Settings
0:10
1:10
1:20
New State
murderer_attack
Conditions
attacking

Transición de ida


murderer_attack -> New State
Has Exit Time
Settings
0:00
0:10
0:20
1:00
1:10
1:20
2:00
2:10
murderer_attack
New State
Conditions
List is Empty
murderer_attack
New State

Transición de vuelta

Vaya al script “CharacterMovement” y añada la siguiente línea de código en la función “Attack” para arrancar la animación de ataque.


void Attack(){
anim.SetTrigger ("attacking");
clone = Instantiate (knifePrefab, knifeS
clone.AddForce (knifeSpawn.transform.rig
}

Arranque de animación de ataque

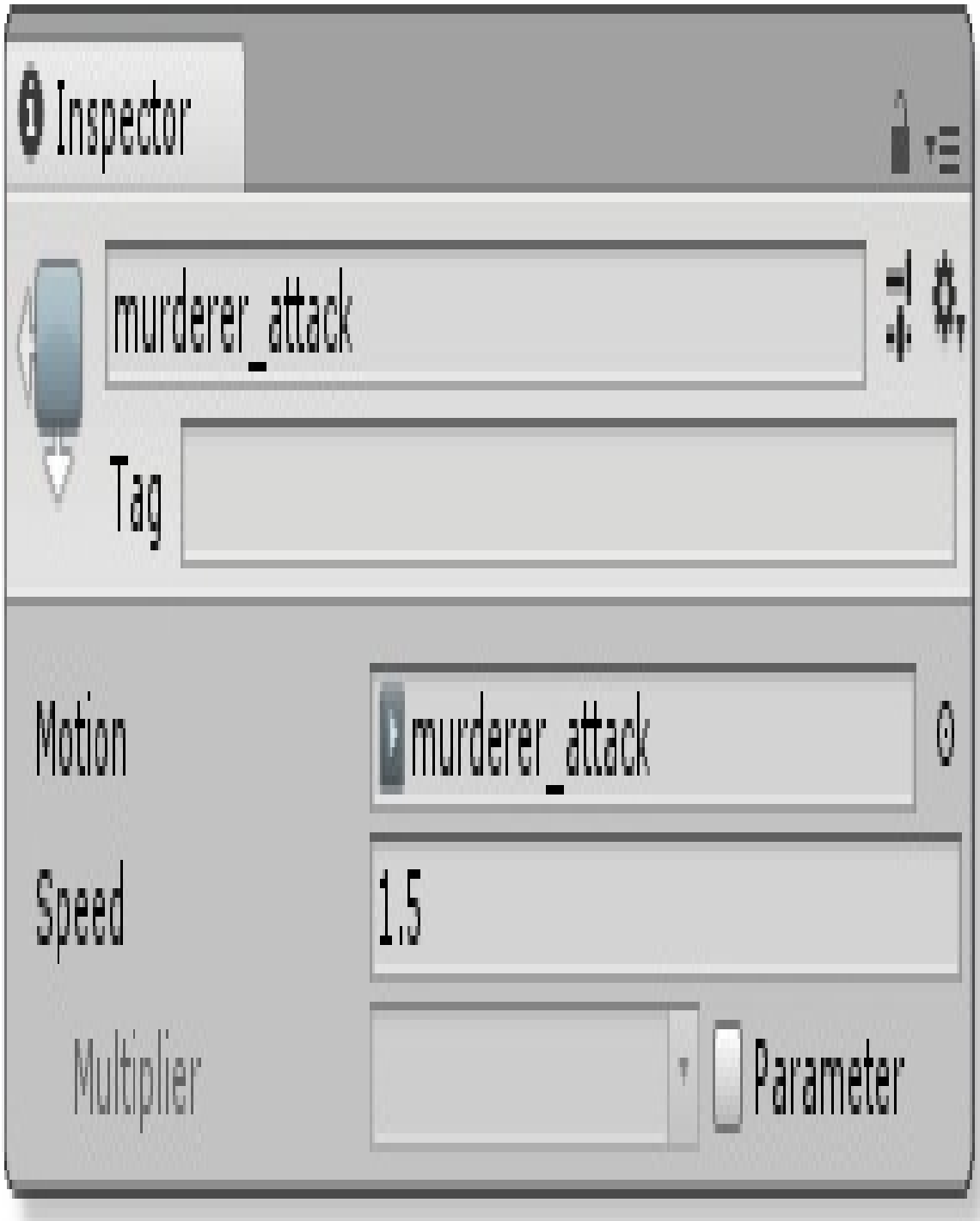
Salve el script.

Previsualice el juego para observar la animación de ataque del “Player”.

Previsualización de ataque

PROBLEMA 1: Queremos que la animación de ataque corra más rápido.

SOLUCIÓN: En la ventana “Animator” seleccione el bloque de animación de ataque y en “Inspector” cambie la velocidad.

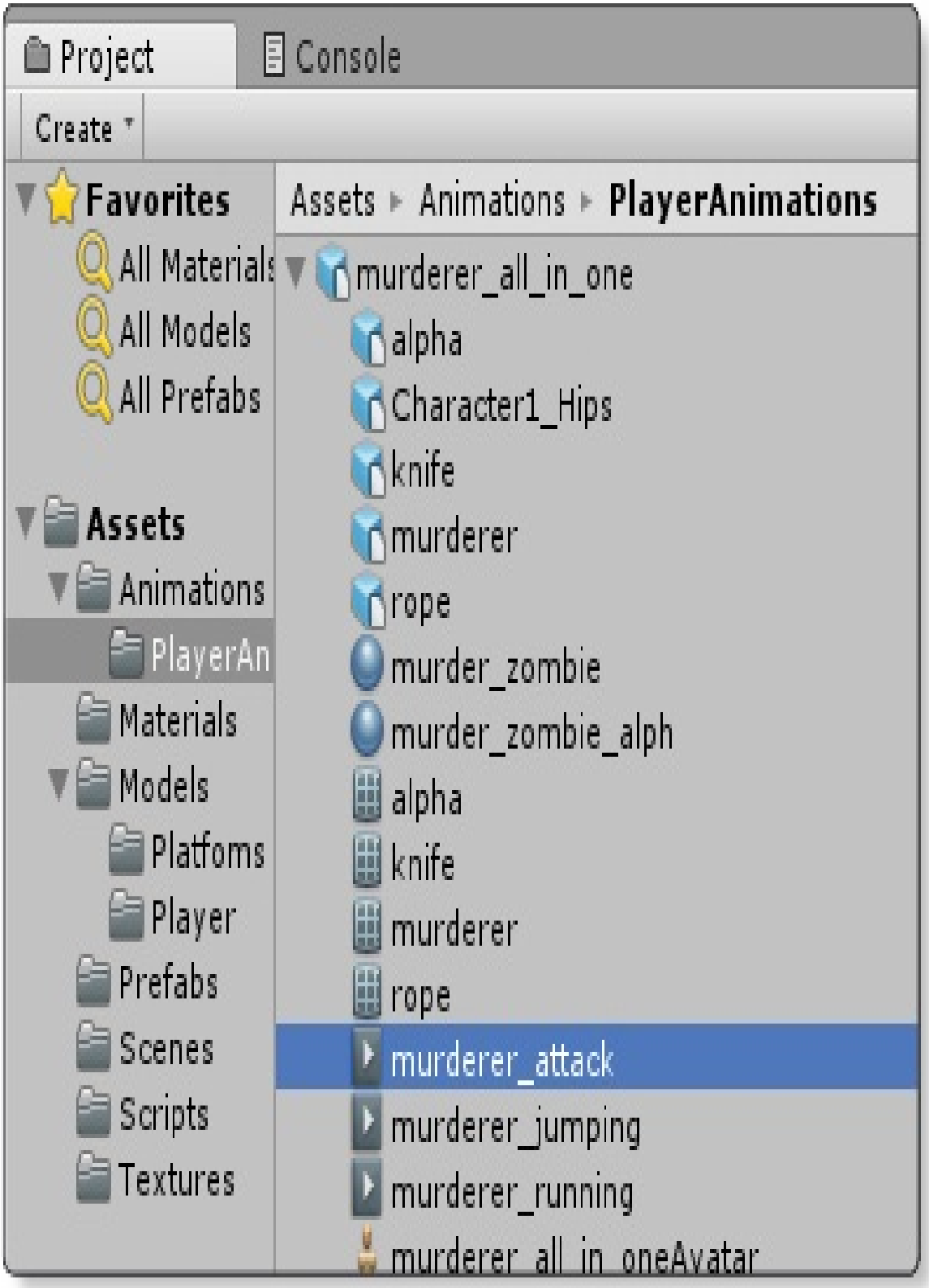

Inspector
murderer_attack
Tag
Motion
murderer_attack
Speed
1.5
Multiplier
Parameter

Aceleración de la animación

PROBLEMA 2: El lanzamiento del proyectil no corresponde al tiempo en el que la animación hace el “slash”.

SOLUCIÓN: Para sincronizar la animación de ataque con el lanzamiento del proyectil, crearemos un “Animation Event”.

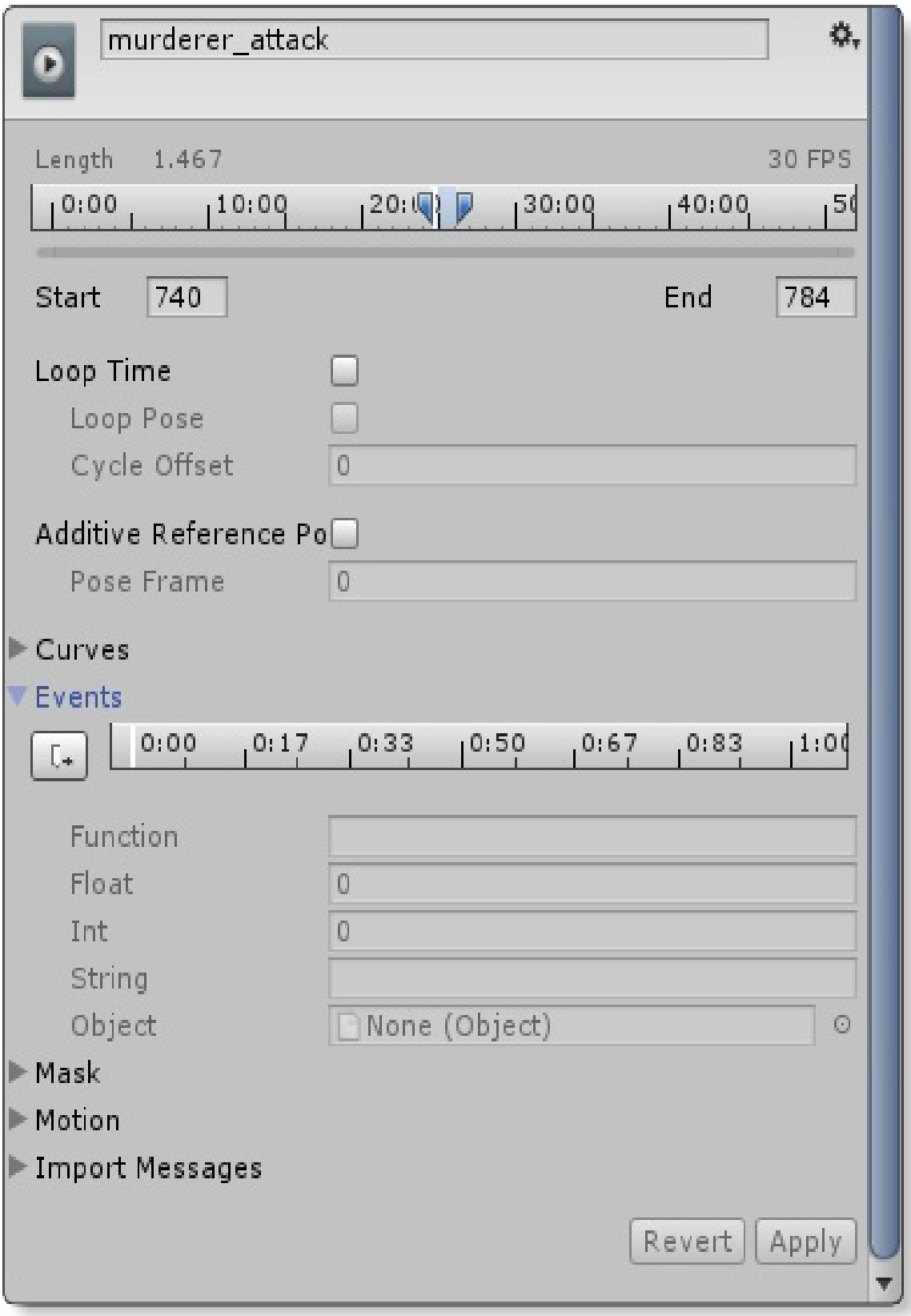
En el folder de “PlayerAnimations” seleccione la animación de ataque.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
PlayerAn
Materials
Models
Platfoms
Player
Prefabs
Scenes
Scripts
Textures
Assets ▸ Animations ▸ PlayerAnimations
murderer_all_in_one
alpha
Character1_Hips
knife
murderer
rope
murder_zombie
murder_zombie_alph
alpha
knife
murderer
rope
murderer_attack
murderer_jumping
murderer_running
murderer_all_in_oneAvatar

Selección de animación de ataque

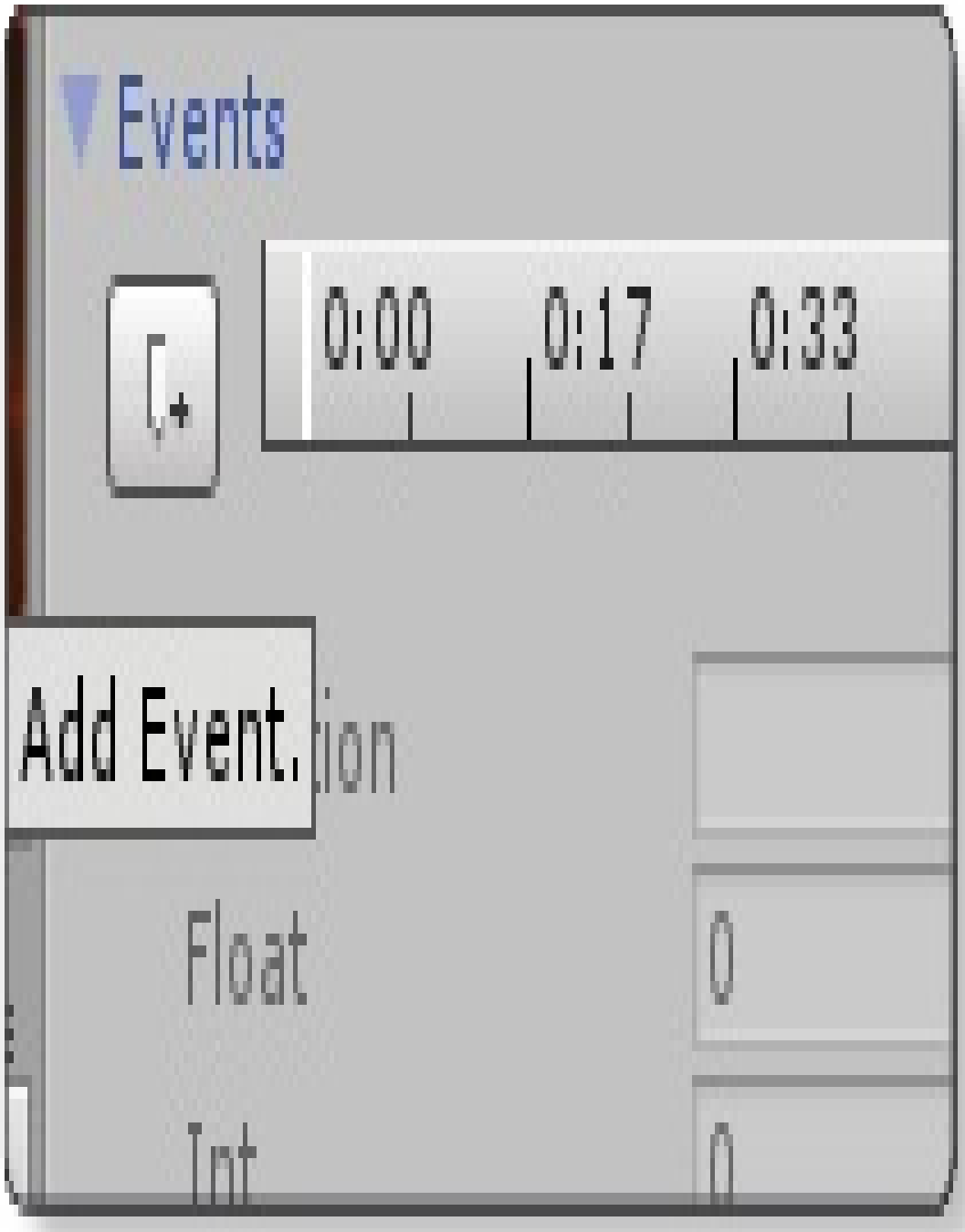
En “Inspector” clic en la opción “edit”.

En la pestaña “Animation” que aparece expanda la opción de “Events”.


murderer_attack
Length 1.467
30 FPS
0:00
10:00
30:00
40:00
Start 740
End 784
Loop Time
Loop Pose
Cycle Offset 0
Additive Reference Po
Pose Frame 0
Curves
Events
0:00
0:17
0:33
0:50
0:67
0:83
Function
Float 0
Int 0
String
Object None (Object)
Mask
Motion
Import Messages
Revert
Apply

Expansión de “Events”

Clic en el botón de “Add Event” para crear un nuevo evento.


Events
0:00
0:17
0:33
Add Event.
Float
0
Int
0

Creación de un evento de animación

Nombre el evento como “CallFireProjectile” y arrástrelo hasta el momento en que la animación del personaje va a realizar el “slash” y clic en “Apply”.

Curves
Events
0:00
0:33
0:50
0:67
0:83
Function
CallFireProjectile
Float
0
Int
0
String
Object
None (Object)
Mask
Motion
Import Messages
Revert
Apply
murderer_attack
2D
1.00
0:10 (024.9%) Frame 750
AssetBundle
None
None

Configuración del “Animation Event”

Cuando hacemos un “Animation Event” este aparece como un separador blanco, cuando lo seleccionamos se vuelve azul.

En la vista previa inferior podemos arrastrar el indicador hasta el momento que deseamos y colocar nuestro indicador superior azul (el evento) en el momentum.

NOTA:

Es de suma importancia hacer clic en “Apply”, si no, el evento no se registrará y se tendrá que repetir el procedimiento.

Vaya al script “CharacterMovement”, escriba la siguiente función modificando las líneas de código que se muestran de la función “Attack”.

```
    void Attack(){
        anim.SetTrigger ("attacking");
    }


    public void CallFireProjectile(){
        clone = Instantiate (knifePrefab, knifeSpawn.position, knifeSpawn.rotation) as Rigidbody;
        clone.AddForce (knifeSpawn.transform.right * knifeSpeed);
    }
}
```

Edición de script

Creamos la función “CallFireProjectile” cuyo nombre es igual al evento de animación.

Dentro de esta función colocamos el código de “Attack” relativo a la instancia de los proyectiles.

Previsualice el juego para observar que el proyectil se lanza en el momento correcto.

Con esto damos por terminado el capítulo “Trabajo con el Player”.

Hasta este momento debemos de tener listos los siguientes objetivos:

Movimiento horizontal del “Player”

Animación de “Idle” y de “Running”

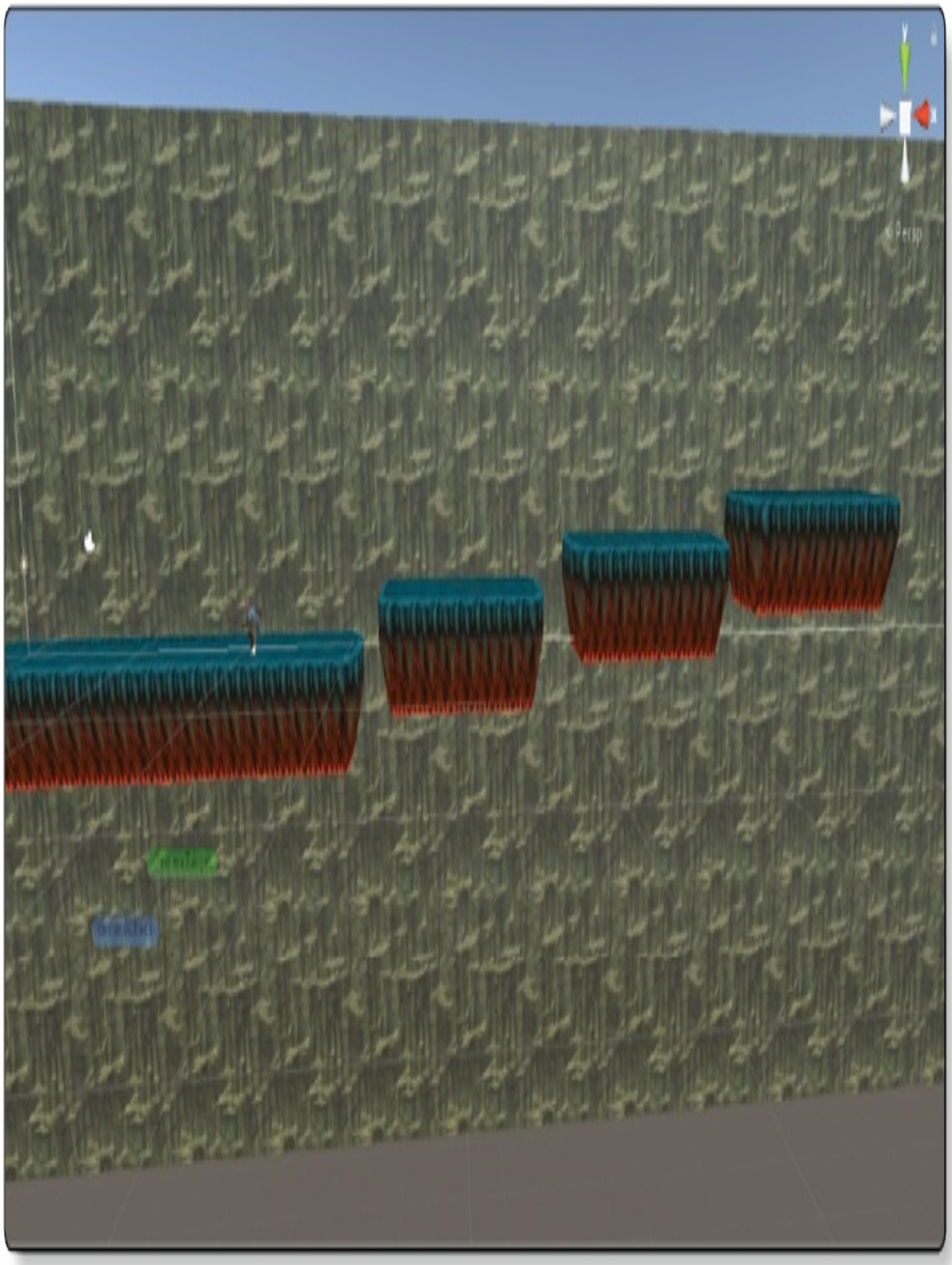
Brinco y animación del salto del “Player”

Funcionalidad y animación del ataque del “Player”

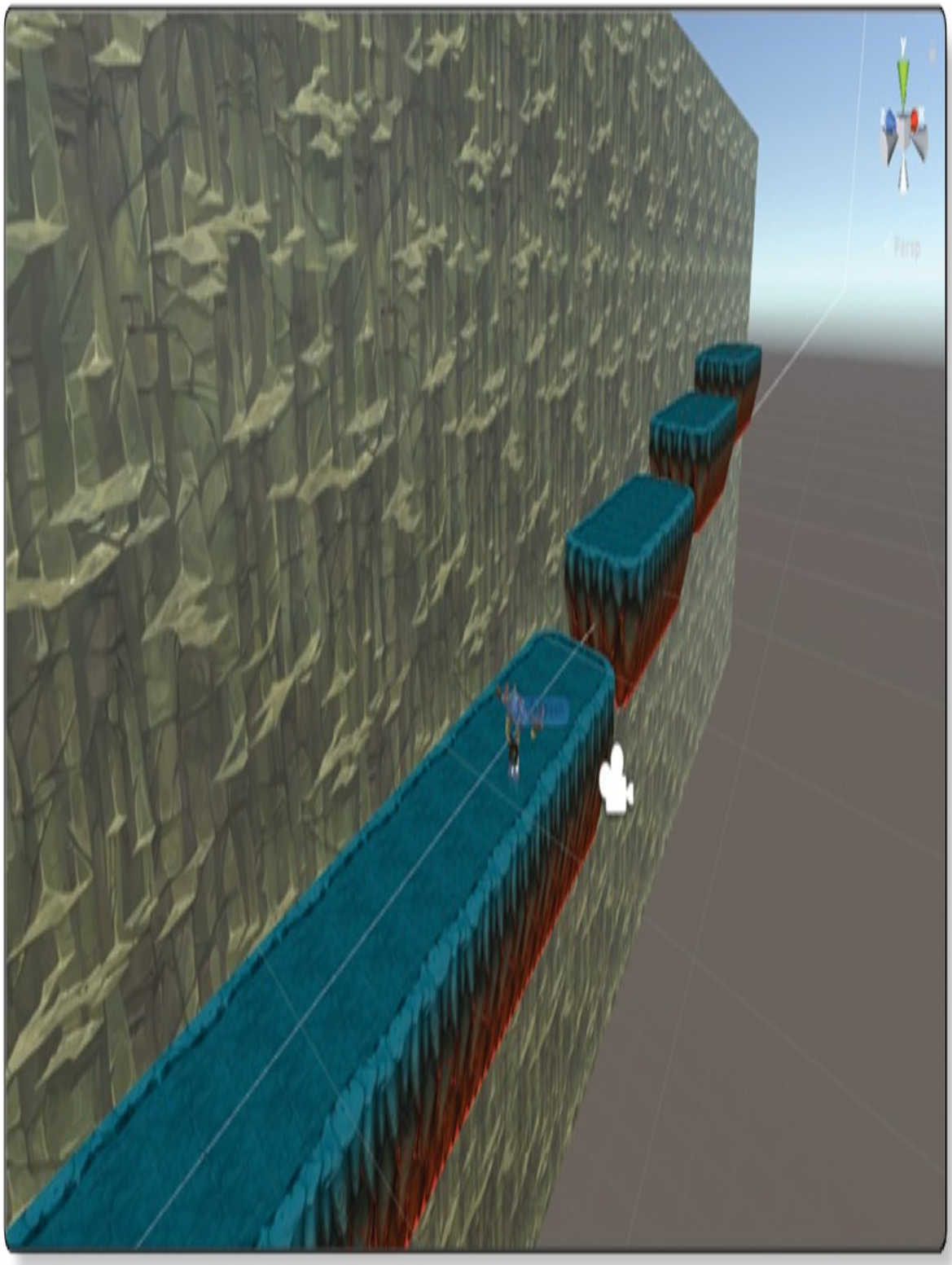
RETO AL LECTOR

Creará un “Plane” asignando en el menú principal “GameObject/ 3D Object/ Plane”, lo duplicará, acomodará y texturizará con los elementos dentro de “resources”.

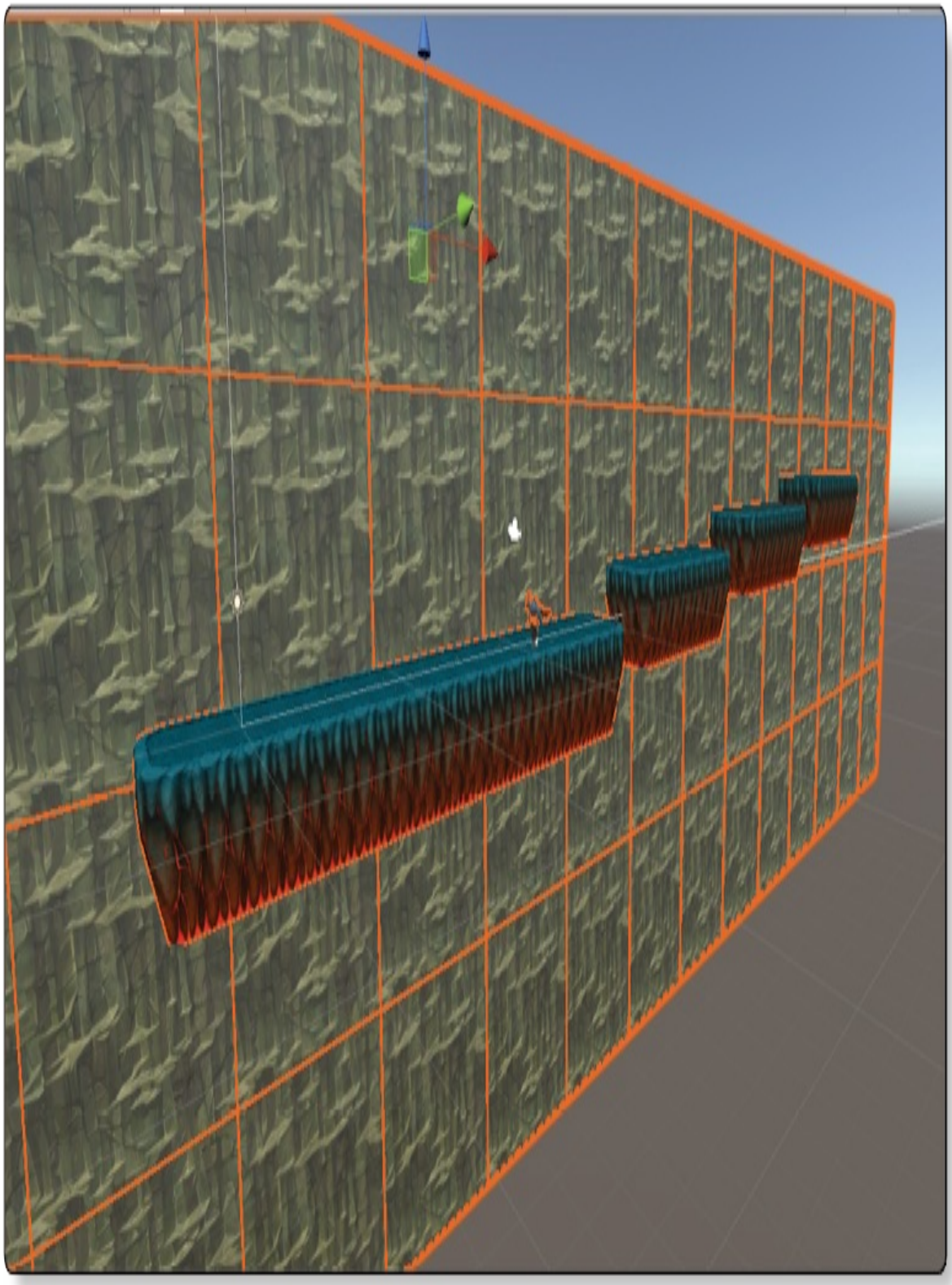
Creará tres plataformas funcionales donde pueda correr el “Player” y convirtiéndolas en “Prefabs”.



Creación de plataformas y background 1



Creación de plataformas y background 2



Creación de plataformas y background 3

3

TRABAJO CON ENEMIGO 01

INTRODUCCIÓN

En este capítulo importaremos nuestro primer enemigo, a través de los conocimientos adquiridos previamente lo prepararemos conforme a su colocación, texturización y animaciones.

Posteriormente abarcaremos el primer módulo de Inteligencia Artificial (AI) para nuestro primer enemigo tipo “caminante”.

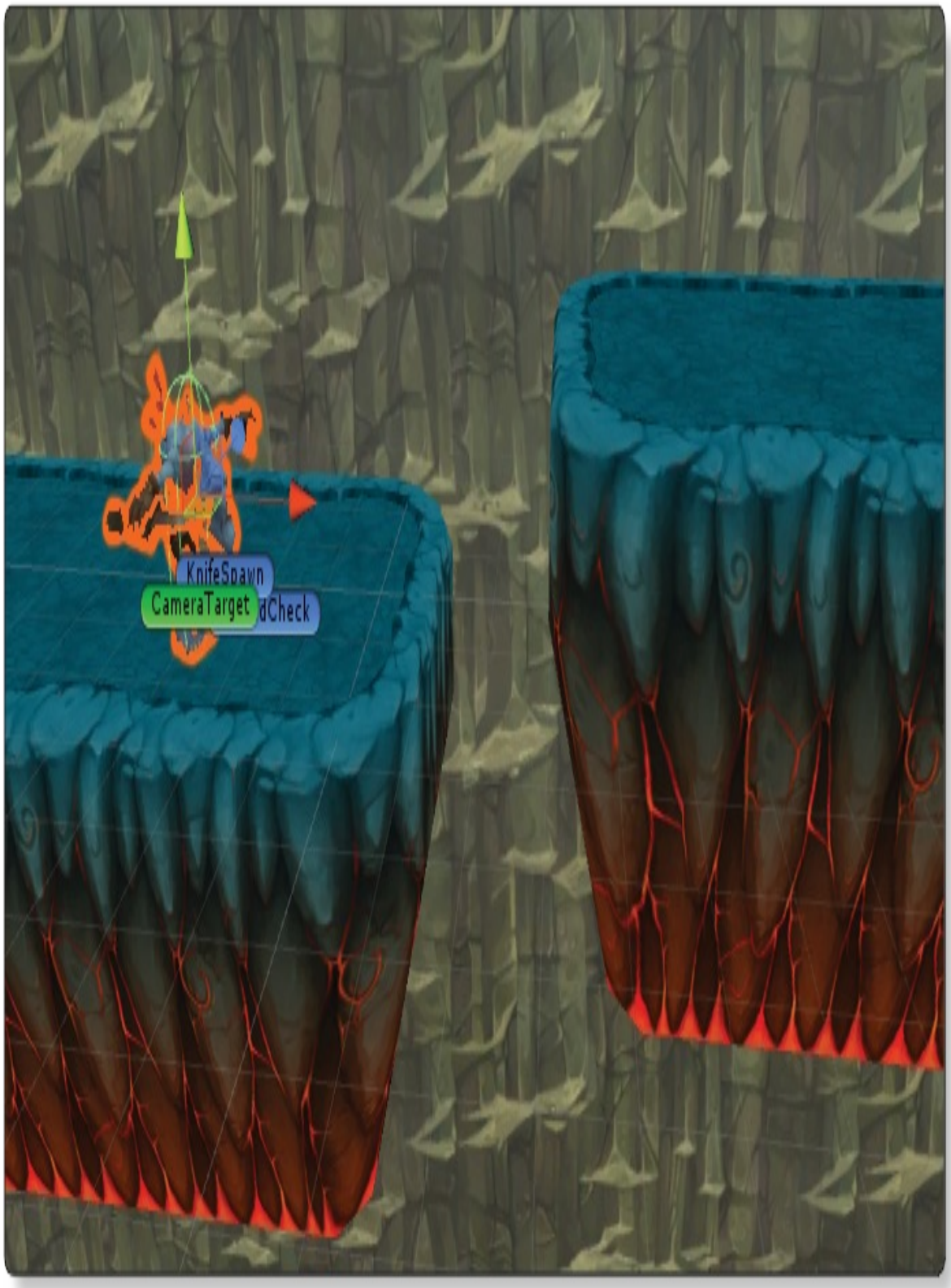
Utilizaremos la técnica con el componente “NavMeshAgent” para que el enemigo siga al Player. A su vez programaremos su inicio de actividad al momento que nuestro “Player” se encuentre cerca de él y que comience su funcionalidad de ataque cuando el “Player” se encuentre junto al enemigo.

Programaremos su estado de vida, su muerte y su desaparición al ser destruido.

PREPARACION DEL ENEMIGO 01

En este apartado nos ocuparemos en preparar a nuestro primer enemigo con todos sus elementos necesarios para pasar posteriormente a la programación de sus funcionalidades.

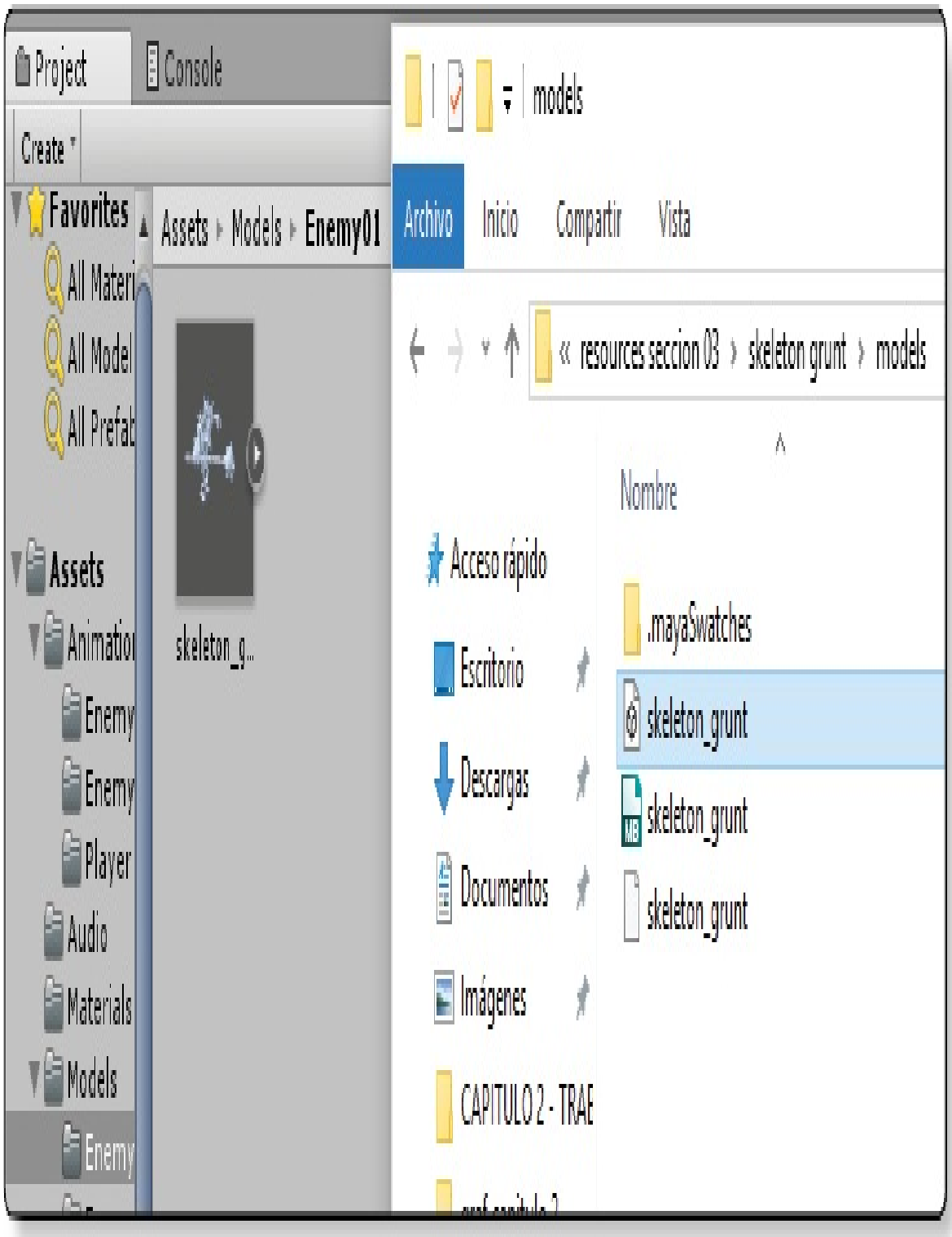
Coloque al “Player” donde se muestra, al borde de la primera plataforma.


KnifeSpawn
CameraTarget

Colocación del Player

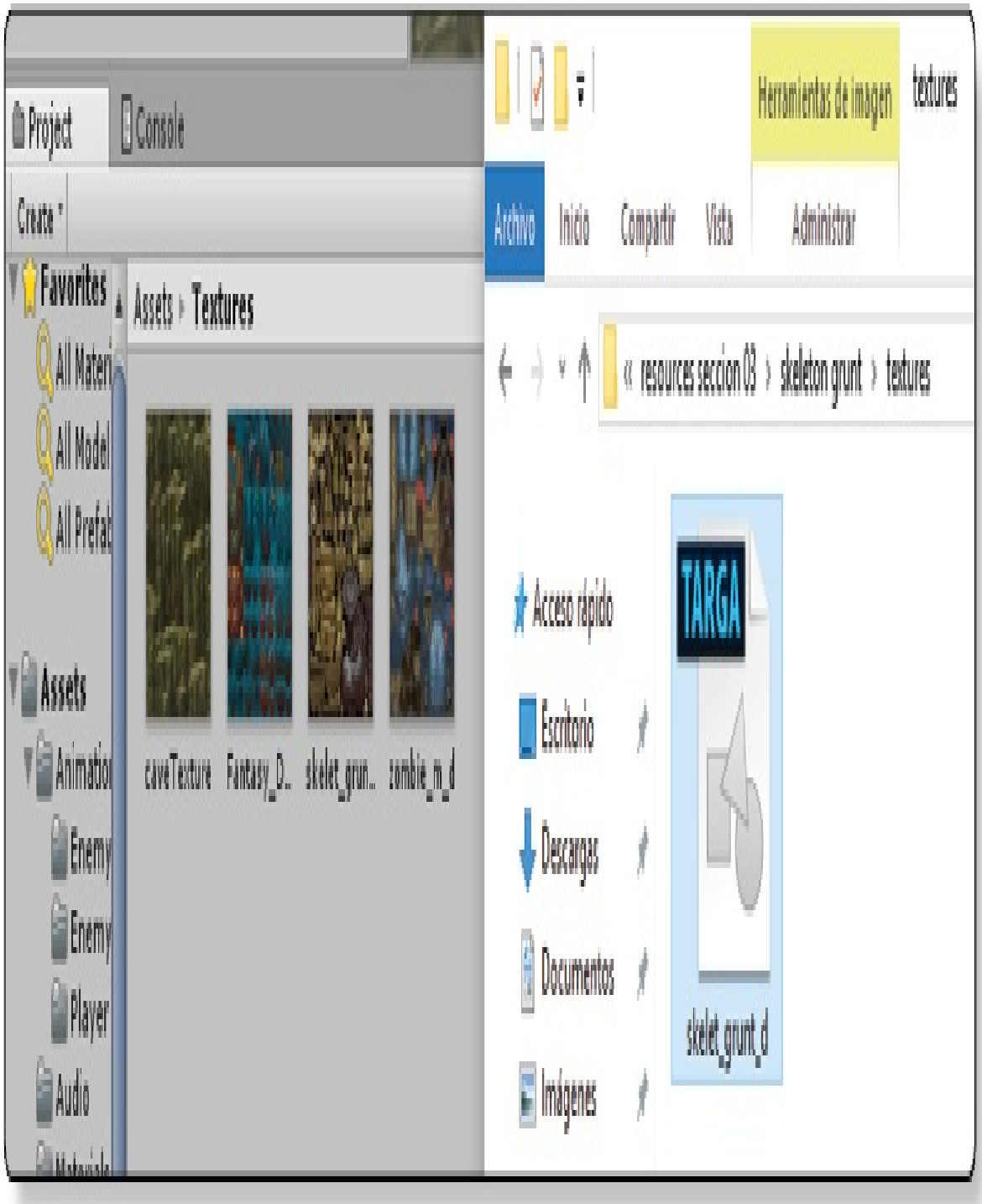
En la ventana “Project” vaya al folder “Models”, haga un nuevo folder y nómbrelo “Enemy01”.

Dentro de la nueva carpeta importe el modelo “Grunt” de la carpeta externa “resources sección 03”.


Project
Console
Create
Favorites
All Materi
All Model
All Prefat
Assets
Animation
Enemy
Enemy
Player
Audio
Materials
Models
Enemy
Assets
Models
Enemy01
skeleton_g...
models
Archivo
Inicio
Compartir
Vista
resources seccion 03
skeleton grunt
models
Nombre
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
CAPITULO 2 - TRAE
.mayaSwatches
skeleton_grunt
skeleton_grunt
skeleton_grunt

Importación del modelo

Dentro del folder “Textures” importe su textura correspondiente.


Project
Console
Create
Favorites
All Materi
All Model
All Prefab
Assets
Animation
Enemy
Enemy
Player
Audio
Assets › Textures
caveTexture
Fantasy_D..
skelet_grun..
zombie_m_d
Herramientas de imagen
textures
Archivo
Inicio
Compartir
Vista
Administrar
« resources seccion 03 › skeleton grunt › textures
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
TARGA
skelet_grunt_d

Importación de la textura

Haga un nuevo material tipo “Unlit/ Texture” para el enemigo.

Project
Console
Create
Favorites
All Materi
All Model
All Prefat
Assets
Animatio
Enemy
Enemy
Assets ▸ Materials
caveTexture
Fantasy_D...
Lava
PlayerMater...
skelet_grun...

Creación del material

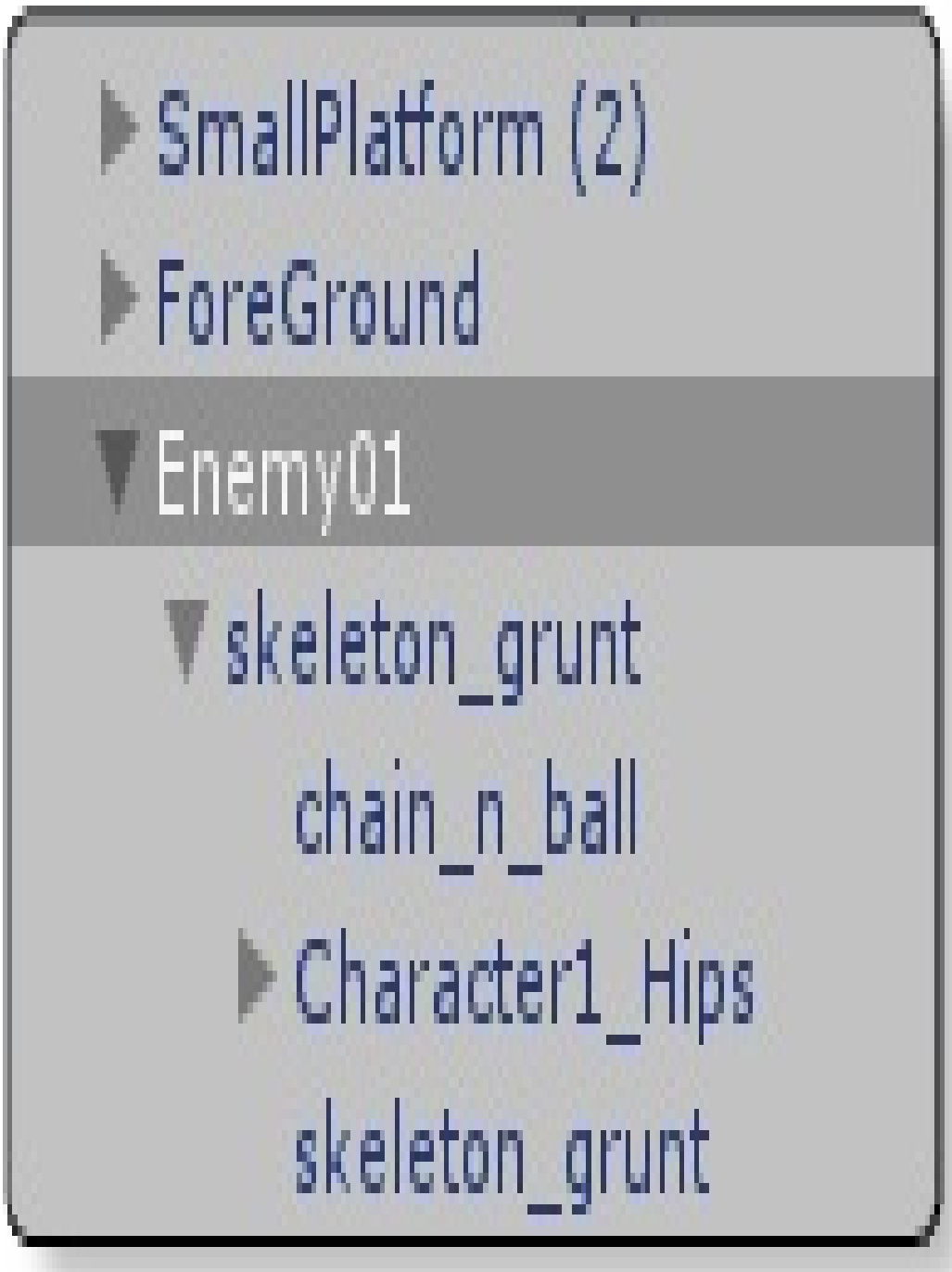
Añada el modelo a la escena como se muestra, texturícelo y colóquelo como se indica.

Persp

Colocación del modelo

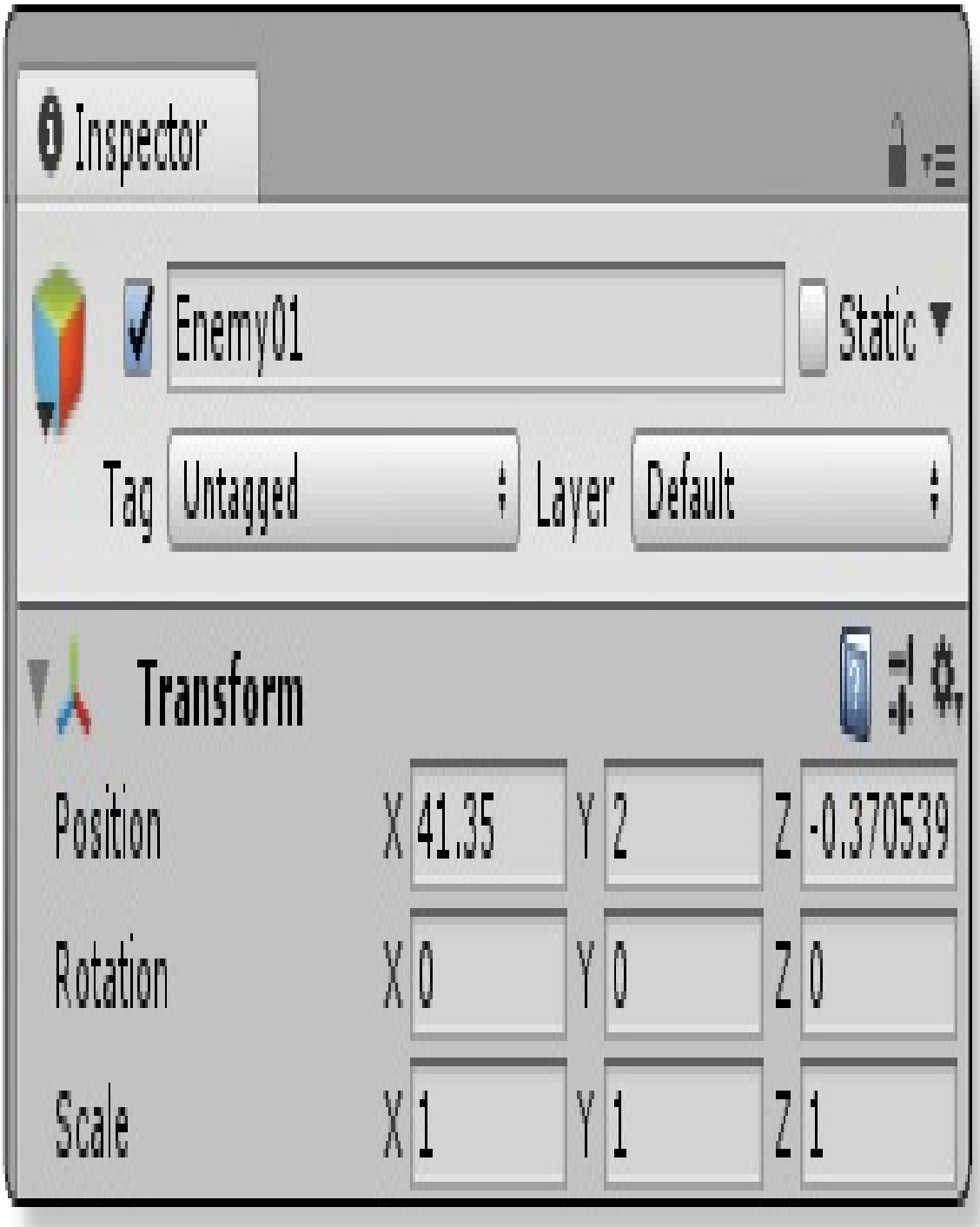
Elimine el componente “Animator” de “Skeleton_Grunt”.

Haga un nuevo “EmptyGameObject”, renómbrelo “Enemy01”, posiciónelo correctamente, Reset en sus transformaciones y empareje “Skeleton_Grunt” a “Enemy01”.


SmallPlatform (2)
ForeGround
Enemy01
skeleton_grunt
chain_n_ball
Character1_Hips
skeleton_grunt

Emparentamiento

Correcta orientación del “Enemy01”


Inspector
Enemy01
Static
Tag
Untagged
Layer
Default
Transform
Position
X 41.35
Y 2
Z -0.370539
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1

Correcta orientación del “Enemy01”

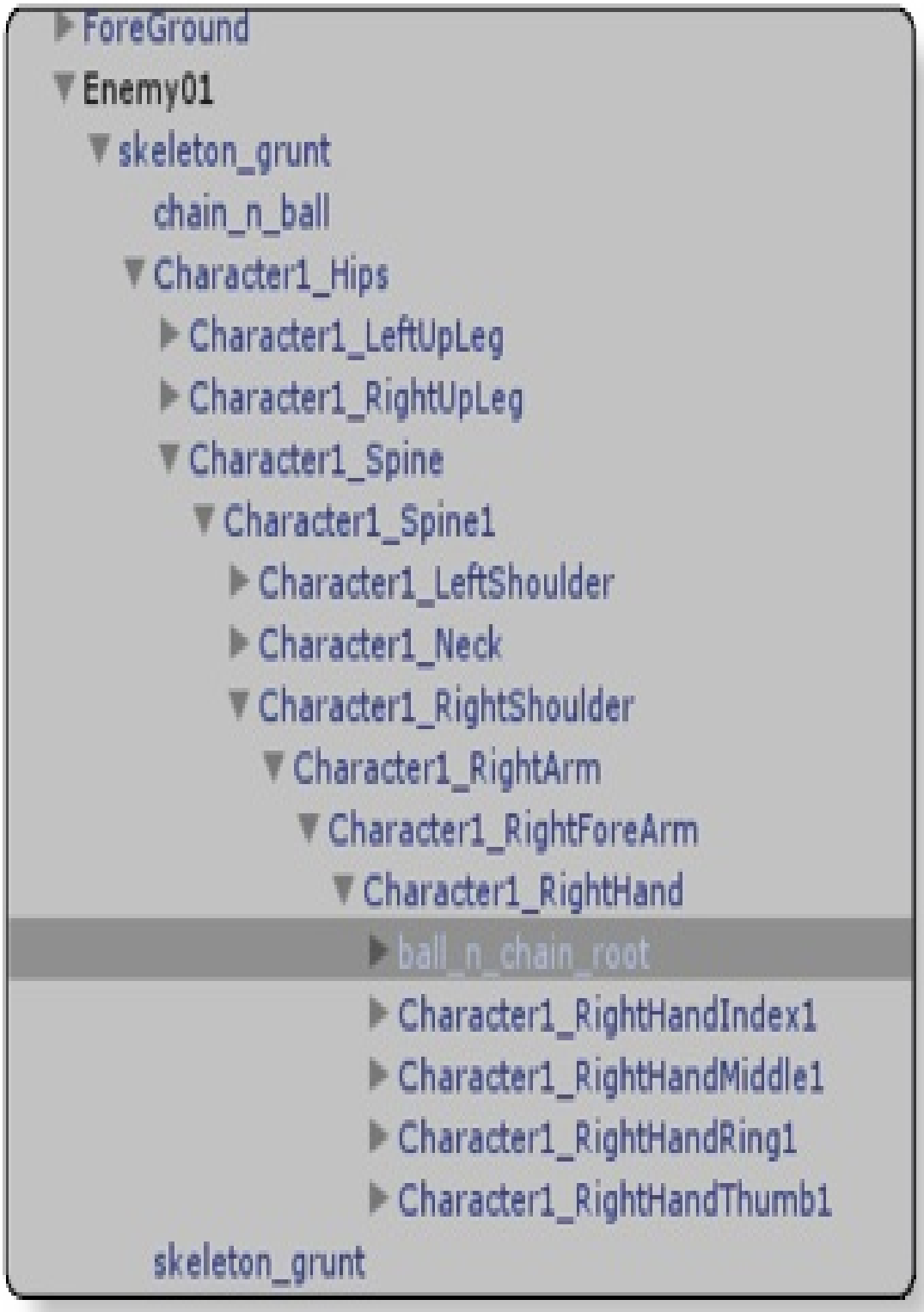
Asigne al enemigo un componente “Rigid body” y un “Capsule Collider”

Persp
Transform
Position X 41.35 Y 2 Z -0.3705
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Rigidbody
Mass 1
Drag Infinity
Angular Drag Infinity
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position X Y Z
Freeze Rotation X Y Z
Capsule Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center
X 0 Y 1.52 Z 0
Radius 0.81
Height 3.24
Direction Y-Axis
Add Component

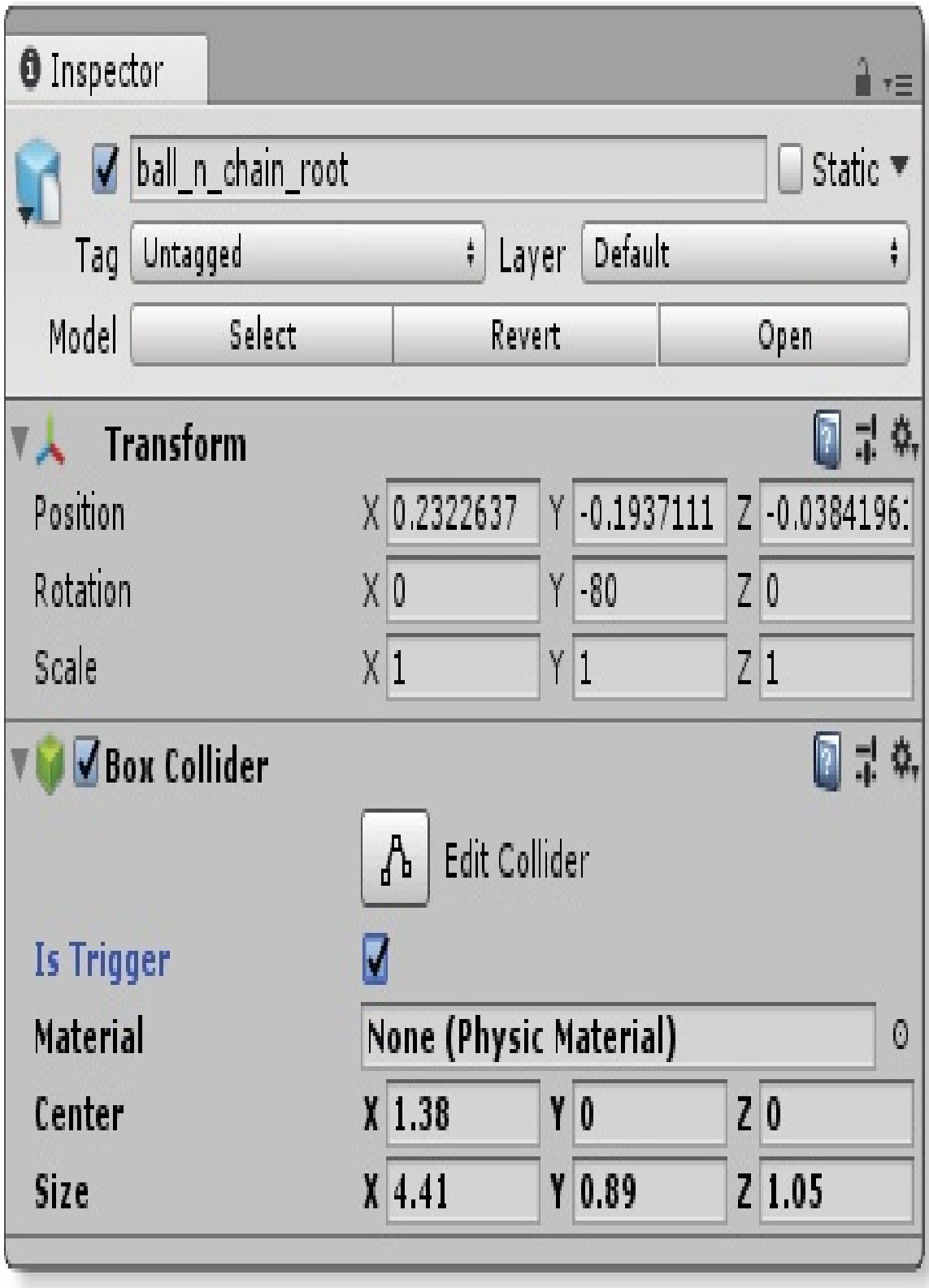
Componentes para “Enemy01”

En “Mass” y en “Drag” asignamos manualmente el valor “Infinity” que va a hacer que tanto su masa y arrastre sean infinitos.

Seleccione el “joint” que se muestra y asigne un “BoxCollider” con los valores mostrados.


ForeGround
Enemy01
skeleton_grunt
chain_n_ball
Character1_Hips
Character1_LeftUpLeg
Character1_RightUpLeg
Character1_Spine
Character1_Spine1
Character1_LeftShoulder
Character1_Neck
Character1_RightShoulder
Character1_RightArm
Character1_RightForeArm
Character1_RightHand
ball_n_chain_root
Character1_RightHandIndex1
Character1_RightHandMiddle1
Character1_RightHandRing1
Character1_RightHandThumb1
skeleton_grunt

Selección del “joint” del arma


Inspector
ball_n_chain_root
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 0.2322637
Y -0.1937111
Z -0.0384196
Rotation
X 0
Y -80
Z 0
Scale
X 1
Y 1
Z 1
Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 1.38
Y 0
Z 0
Size
X 4.41
Y 0.89
Z 1.05

BoxCollider del arma

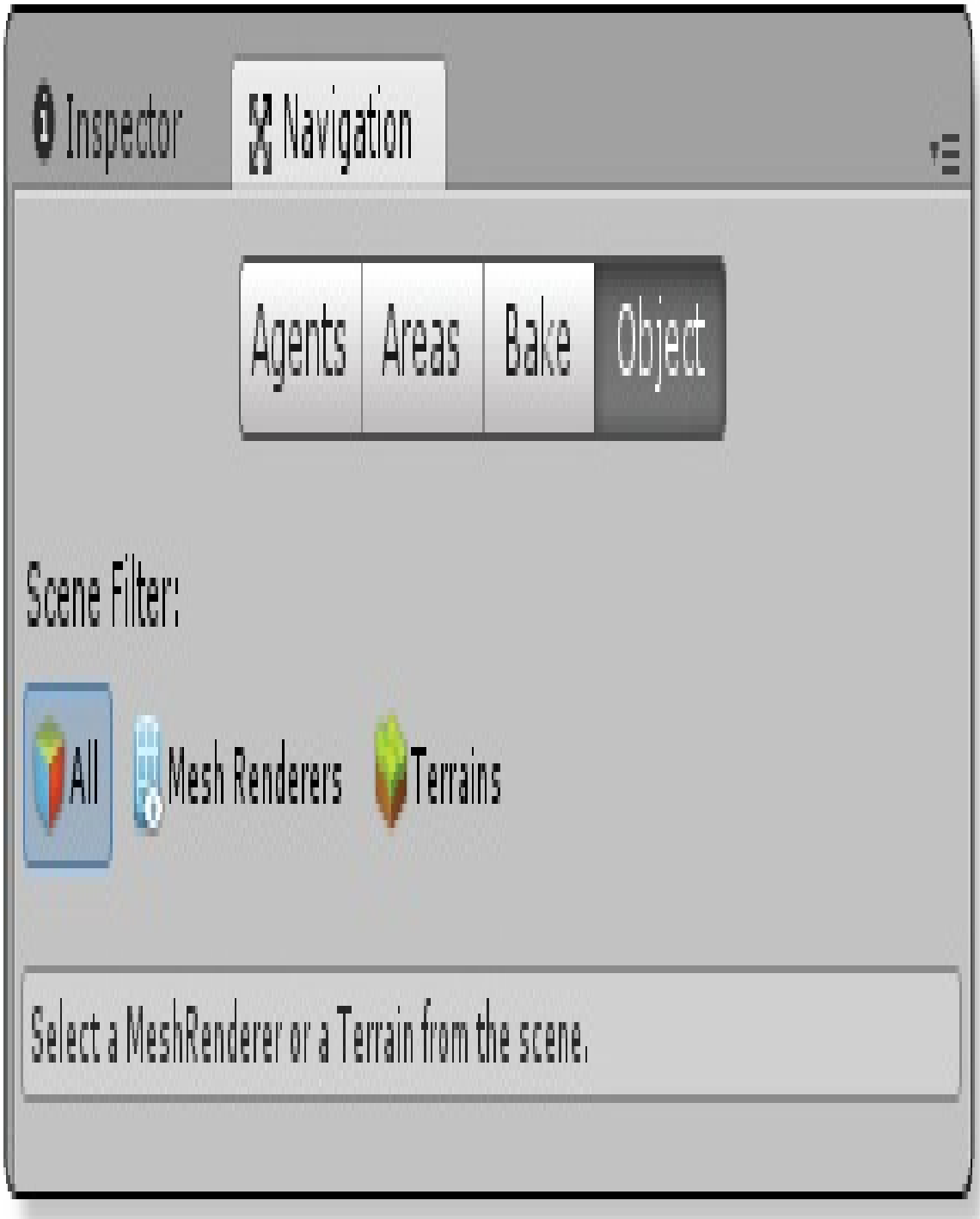
Box Collider del arma

USO DEL COMPONENTE “NAV MESH AGENT”

En este apartado vamos a trabajar con el movimiento del enemigo en función a que se active cuando se vaya acercando nuestro “Player” y que comience a seguirle.

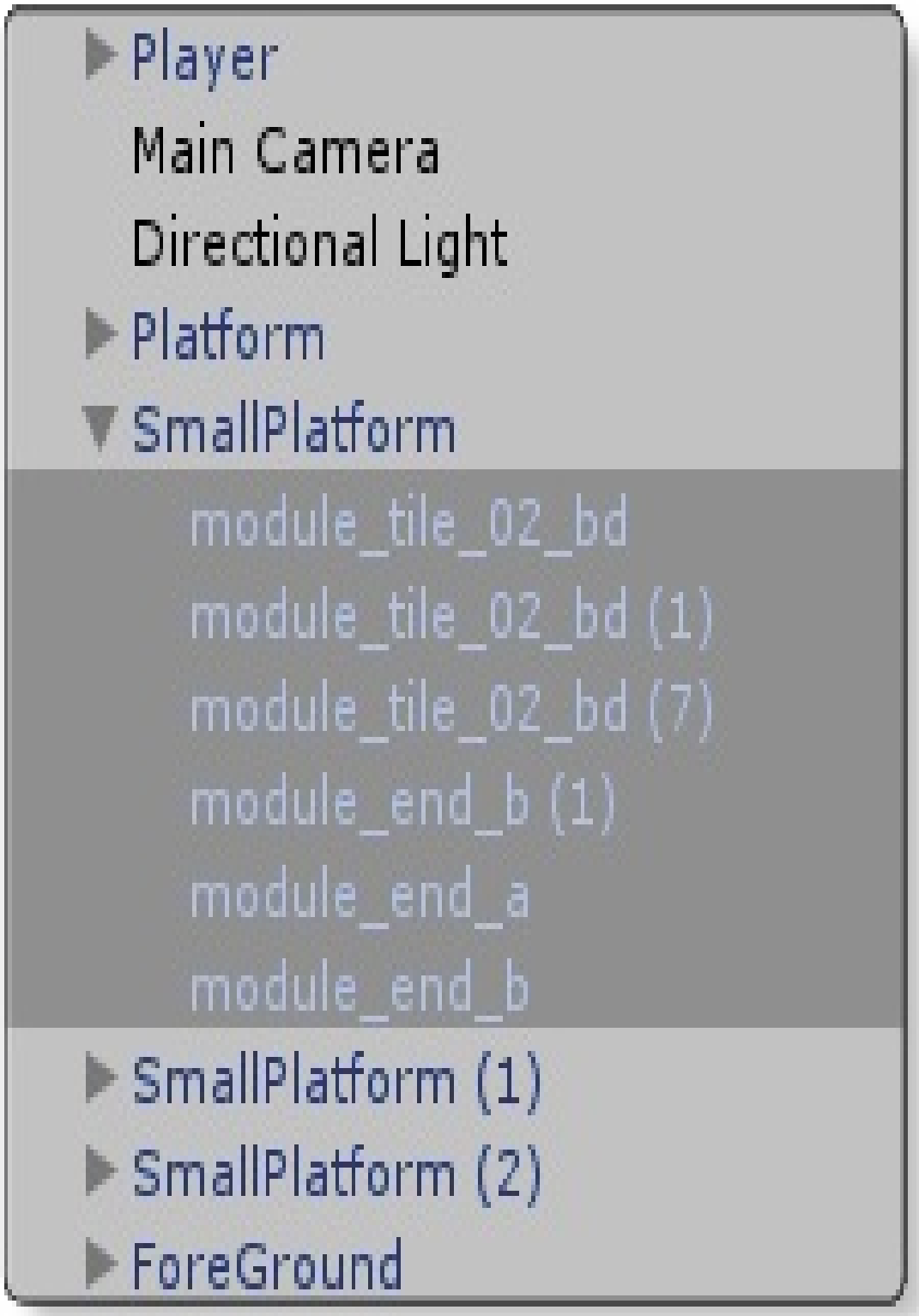
Vamos a utilizar el componente “Nav Mesh Agent” para referenciar y limitar el área donde va a moverse el enemigo y posteriormente trabajaremos en su programación.

En el menú general asigne “Window/ Navigation” para abrir la ventana “Navigation”.

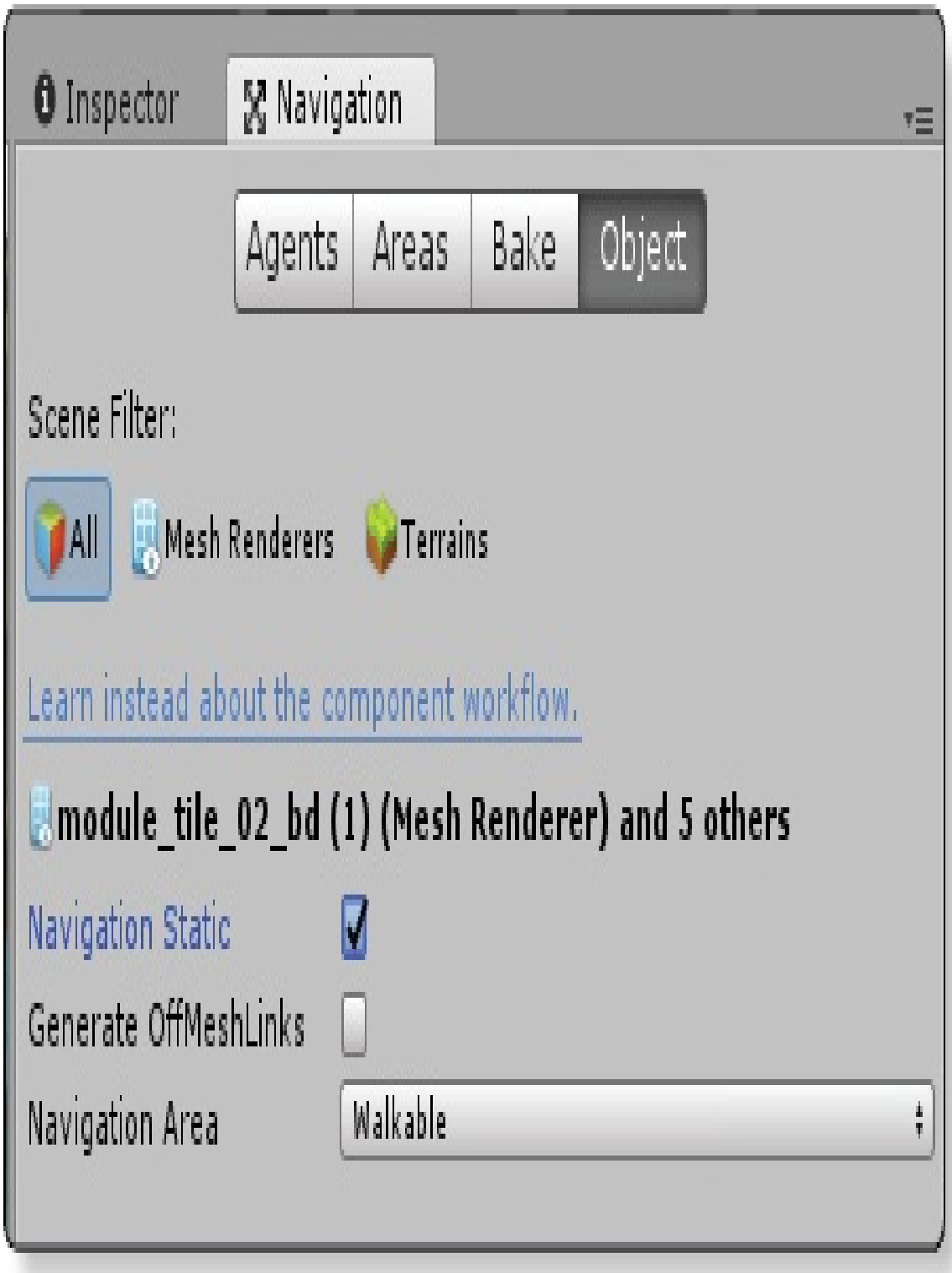

Inspector
Navigation
Agents
Areas
Bake
Object
Scene Filter:
All
Mesh Renderers
Terrains
Select a MeshRenderer or a Terrain from the scene.

Ventana Navigation

Seleccione toda la geometría que comprende la plataforma y en la pestaña “Navigation” asigne los parámetros que se indican.


Player
Main Camera
Directional Light
Platform
SmallPlatform
module_tile_02_bd
module_tile_02_bd (1)
module_tile_02_bd (7)
module_end_b (1)
module_end_a
module_end_b
SmallPlatform (1)
SmallPlatform (2)
ForeGround

Geometría de la plataforma


Inspector
Navigation
Agents
Areas
Bake
Object
Scene Filter:
All
Mesh Renderers
Terrains
Learn instead about the component workflow.
module_tile_02_bd (1) (Mesh Renderer) and 5 others
Navigation Static
Generate OffMeshLinks
Navigation Area
Walkable

Parámetros de Navigation

De esta manera iniciamos con el proceso para definir el recorrido del enemigo asignando la propiedad de “Walkable” o “Caminable” de la geometría

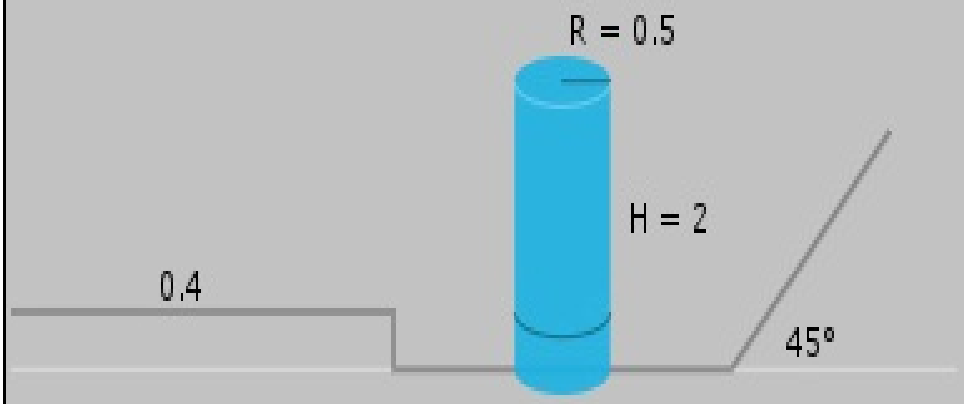
Seleccione la pestaña “Bake” y clic en el botón “Bake”.

Inspector | Navigation

Agents | Areas | Bake | Object

Learn instead about the component workflow.

Baked Agent Size



Agent Radius	0.5
Agent Height	2
Max Slope	45
Step Height	0.4

Generated Off Mesh Links

Drop Height	0
Jump Distance	0

▶ Advanced

Clear | Bake

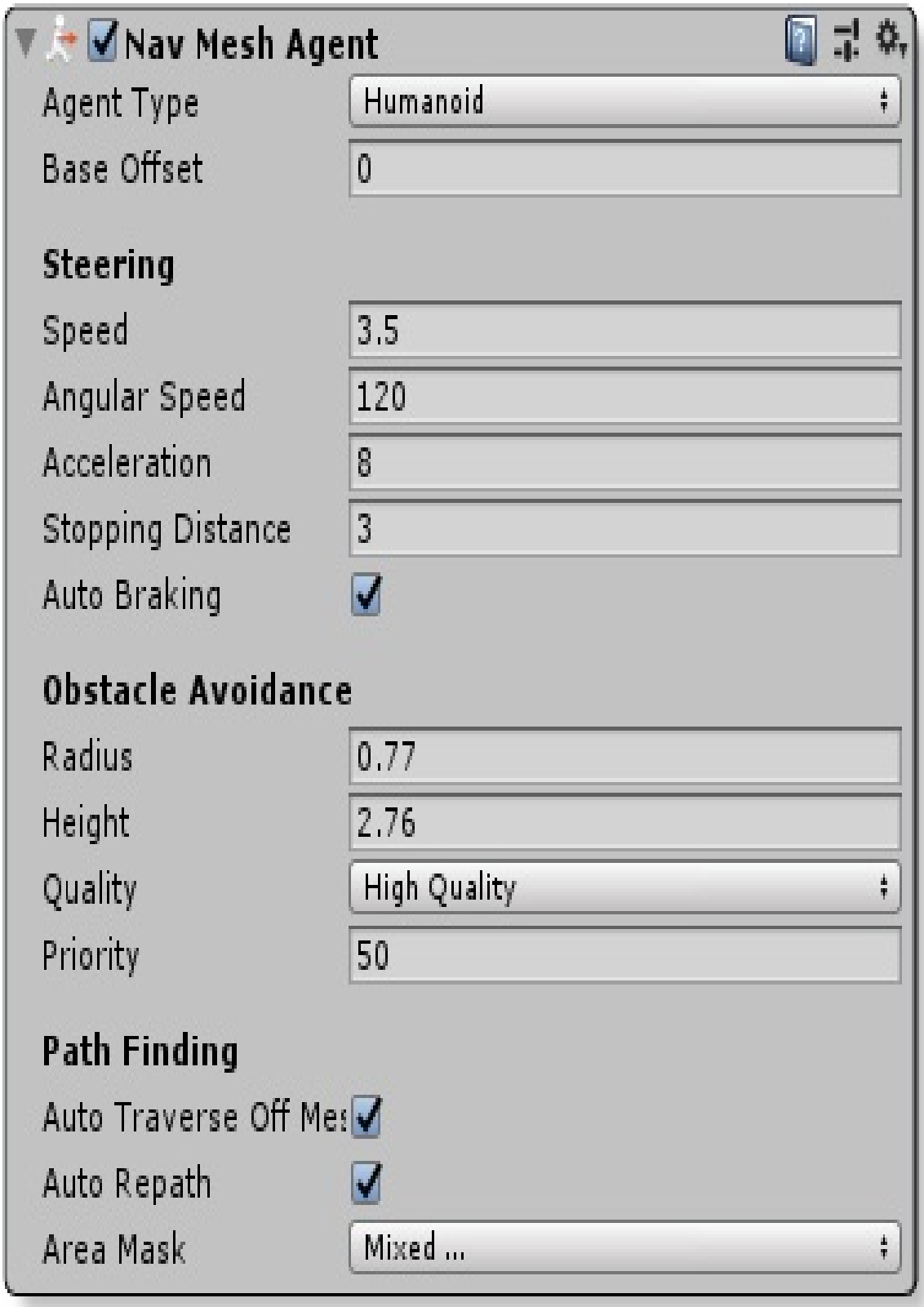
Opciones de “Bake”

En esta pestaña terminamos de “cocinar” el terreno donde va a poder caminar nuestro enemigo, a su vez si tuviéramos un terreno con elevación angular, también en esta pestaña podemos asignarle los límites angulares para caminar.

Persp

Geometría caminable

Seleccione “Enemy01” y asigne un componente “navMeshAgent” y asigne los valores que se muestran en “Inspector”.


Nav Mesh Agent
Agent Type
Humanoid
Base Offset
0
Steering
Speed
3.5
Angular Speed
120
Acceleration
8
Stopping Distance
3
Auto Braking
Obstacle Avoidance
Radius
0.77
Height
2.76
Quality
High Quality
Priority
50
Path Finding
Auto Traverse Off Mes
Auto Repath
Area Mask
Mixed ...

Componente NavMeshAgent

Establecemos el radio y la altura para la detección de obstáculos del componente “Nav Mesh Agent” así como su “Stopping Distance” para que el enemigo se detenga un poco antes de donde se encuentra el “Player”.

NavMeshAgent en Enemy01

Haga un nuevo C# Script y nómbrelo “Enemy01Move”.

Asigne el nuevo script al objeto “Enemy01”.

Abra el nuevo script, mande llamar la librería que se indica y escriba las siguientes variables.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class Enemy01Move : MonoBehaviour {

    [SerializeField] private Transform player;
    private NavMeshAgent nav;
    private Animator anim;

    // Use this for initialization
    void Start () {
```

Serialización de campo de variable

Cuando utilizamos el componente “Nav Mesh Agent” es necesario mandar llamar la librería “UnityEngine.AI” al inicio.

En este caso estamos creando una variable privada tipo “Transform” llamada “player”, queremos que sus propiedades continúen privadas, pero también queremos visualizarla en la ventana de “Inspector “de Unity, para ello utilizamos [SerializeField].

Inicialice las variables n y escriba el siguiente código en la función “Update”.

```
// Use this for initialization
void Start () {

    anim = GetComponent<Animator>();
    nav = GetComponent<NavMeshAgent>();

}

// Update is called once per frame
void Update () {

    if(Vector3.Distance (player.position, this.transform.position) < 12) {
        print ("Player in range");
    }

}
```

Función para comprobar el “Player” en rango

En la función “Update” estamos escribiendo una estructura condicional en donde indicamos que se va a comprobar que el “Player” se encuentre a menos de 12 puntos de distancia del objeto que tiene el script (this), en este caso nuestro enemigo.

El comando Print nos imprime en la consola lo que tengamos escrito entre comillas dentro de los paréntesis.

Vaya a Unity y asigne manualmente el objeto “Player” en la variable de objeto.

Enemy 01 Move (Script)
Script
Enemy01Move
Player
Player (Transform)

Asignación de objeto de transformación

Previsualice el juego para observar que cuando el “Player” se acerca al enemigo, en la ventana consola nos imprime “Player in range” (Player en rango).

Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
Player in range
UnityEngine.MonoBehaviour:print(Object)

Ventana “Console”

NOTA:

Active la opción “Collapse” para evitar que se muestre el mensaje infinidad de veces en la consola y que únicamente se colapse para mostrarnos en la parte derecha una capsula con la cuenta de las veces que se está imprimiendo el mensaje.

NOTA:

Para borrar el contenido de la consola, basta con oprimir el botón “Clear”.

En el script “Enemy01Move” edite la función “Update”.

```
void Update () {


    if(Vector3.Distance (player.position, this.transform.position) < 12) {
        nav.SetDestination  (player.position);
    }


}
```

Funcionalidad de destino hacia “Player”

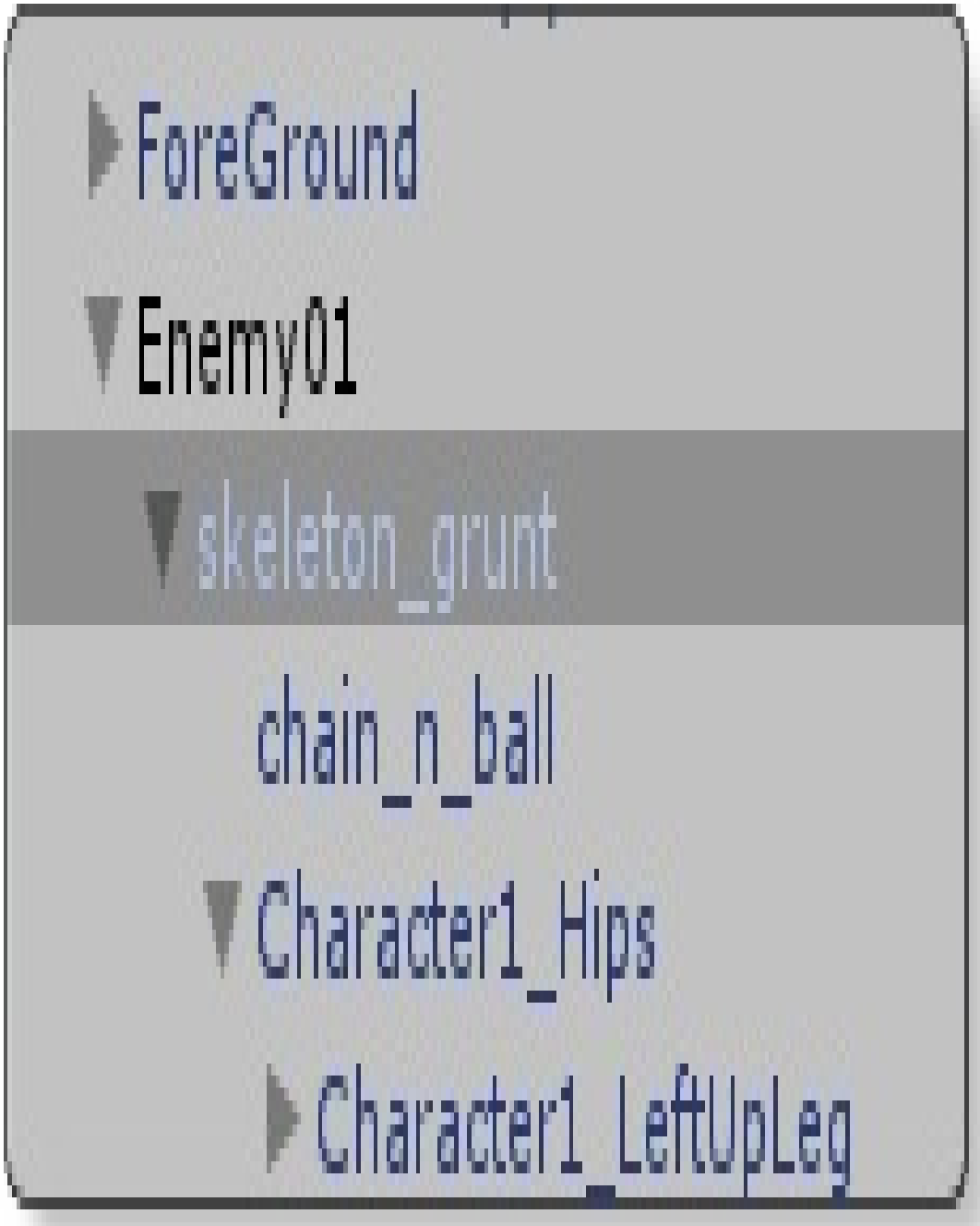
Sustituimos el comando relativo a “print” por el comando para que nuestro enemigo siga al “Player” cuando se encuentre en rango.

Previsualice el juego para observar que el enemigo comienza a seguir al “Player” cuando se encuentre en rango.

PROBLEMA: El enemigo cambia de dirección cuando comienza a seguir al “Player”.

Problema de dirección

SOLUCIÓN: Seleccione el modelo “Skeleton_Grunt” y rótelo.


ForeGround
Enemy01
skeleton_grunt
chain_n_ball
Character1_Hips
Character1_LeftUpLeg

Selección de modelo

Correcta rotación

Inspector
Navigation
skeleton_grunt
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y -369.1
Z 0
Scale
X 1
Y 1
Z 1

Correcto parámetro de rotación

Previsualice el juego para comprobar que la orientación se encuentra correcta.

Movimiento correcto

ANIMACIÓN DEL CAMINADO DEL ENEMIGO 01

En este apartado trabajaremos con la programación de las animaciones de nuestro enemigo. Haremos y programaremos las animaciones del estado “Idle” y del caminado.

En el folder de Animations, haga uno nuevo y nómbrelo “Enemy01 Animations”.

Dentro del nuevo folder haga un nuevo “Animator Controller” y nómbrelo “Enemy01AC”.

Con los conceptos aprendidos haga las animaciones del “Enemy01” conforme a su archivo “Skeleton Grunt Animation List” dentro de su folder “Animations” en “resources sección 03” y nómbrelas.

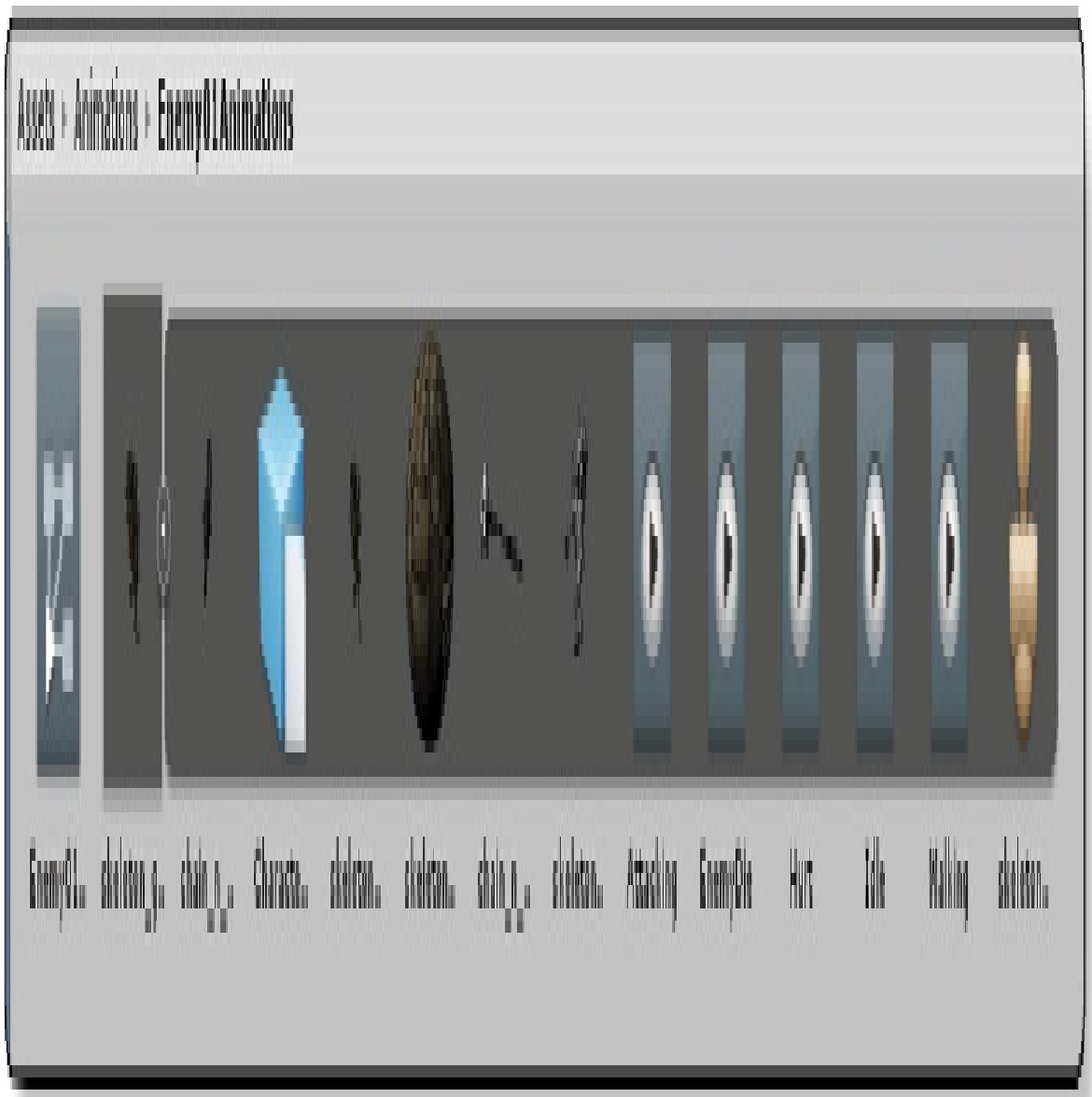
frame: 0090-0119: death (EnemyDie)

frame: 0125-0204: default attack (Attacking)

frame: 0379-0428: idle (loop) (Idle)

frame: 1629-1658: walk (loop) (Walking)

frame: 0240-0269: get hit from front (Hurt)


Assets ▸ Animations ▸ EnemyAIAnimations

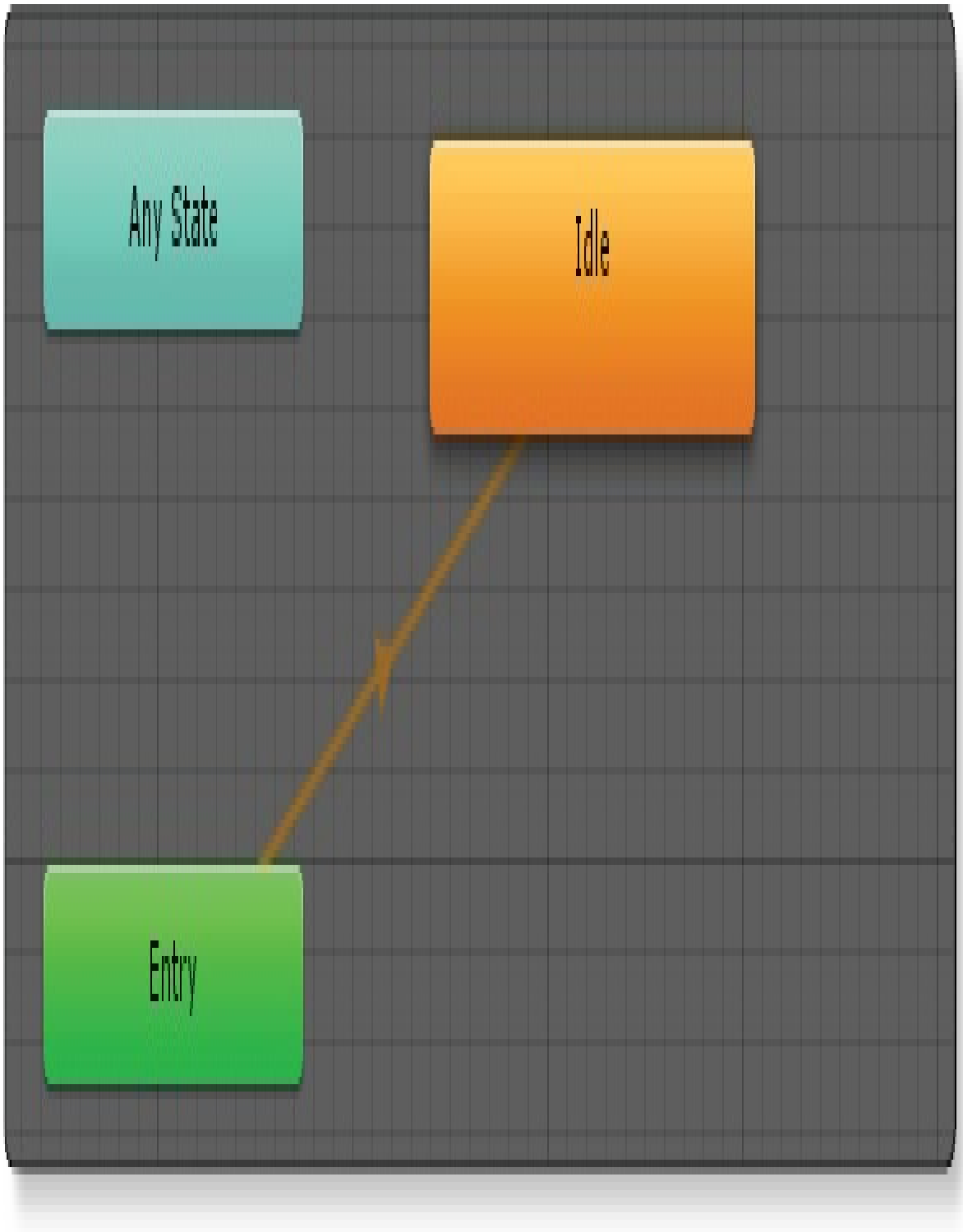
Creación de animaciones

Asigne un componente “Animator” al objeto “Enemy01” y asigne los valores que se muestran.

Enemy Move (Script)
Script
EnemyMove
Player
Player (Transform)
Animator
Controller
Enemy01AC
Avatar
skeleton_gruntAvatar
Apply Root Motion
Update Mode
Normal
Culling Mode
Always Animate
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0
PPtr: 0

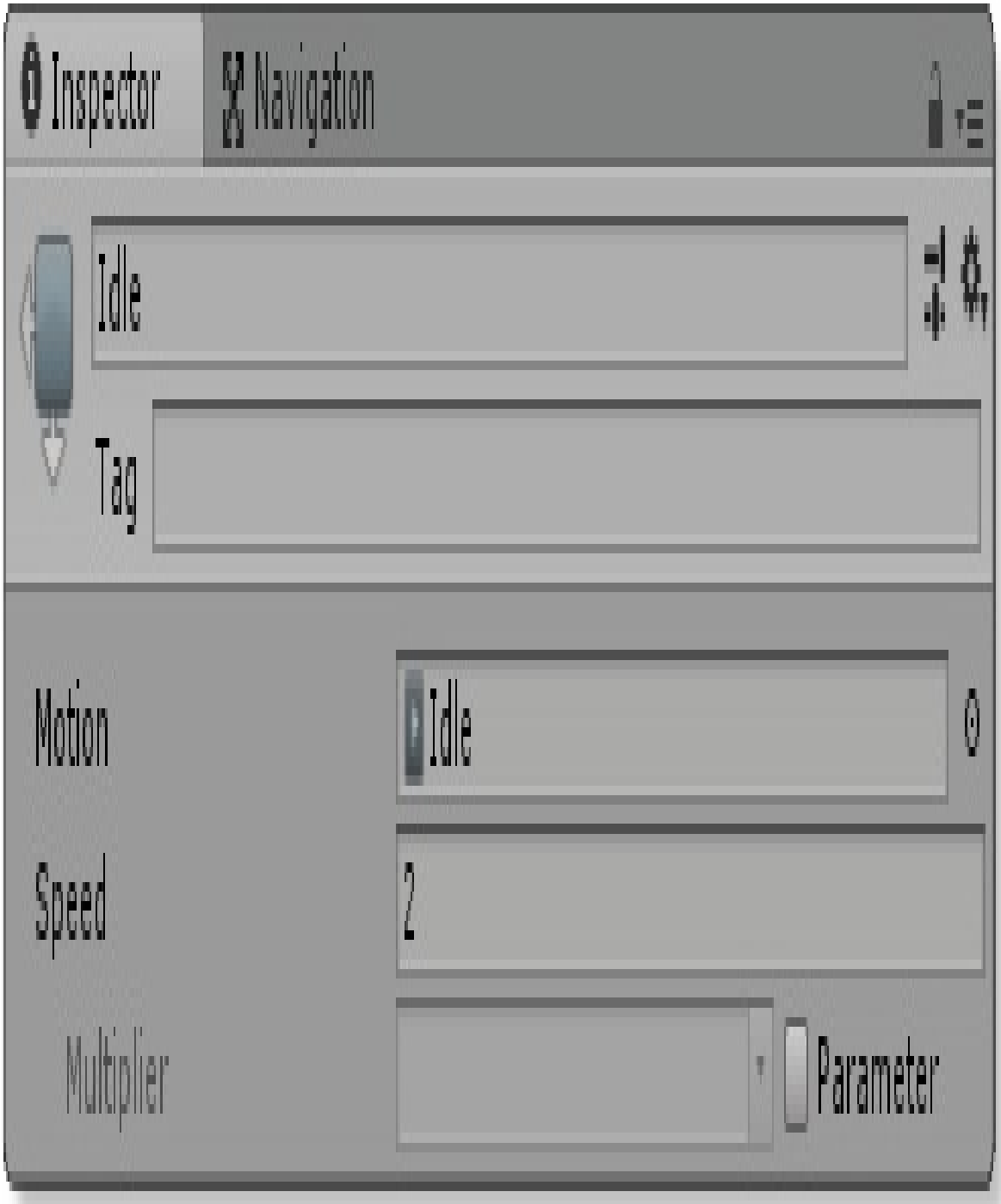
Parámetros de Animator

En la ventana “Animator” asigne su animación de “Idle”.


Any State
Idle
Entry

Animación Idle en Animator

Con el clip de “Idle” seleccionado en “Animator”, asigne un “Speed” de 2.


Inspector
Navigation
Idle
Tag
Motion
Idle
Speed
2
Multiplier
Parameter

Cambio de velocidad a clip de animación

Previsualice el juego para observar la animación.

Previsualización de animación Idle

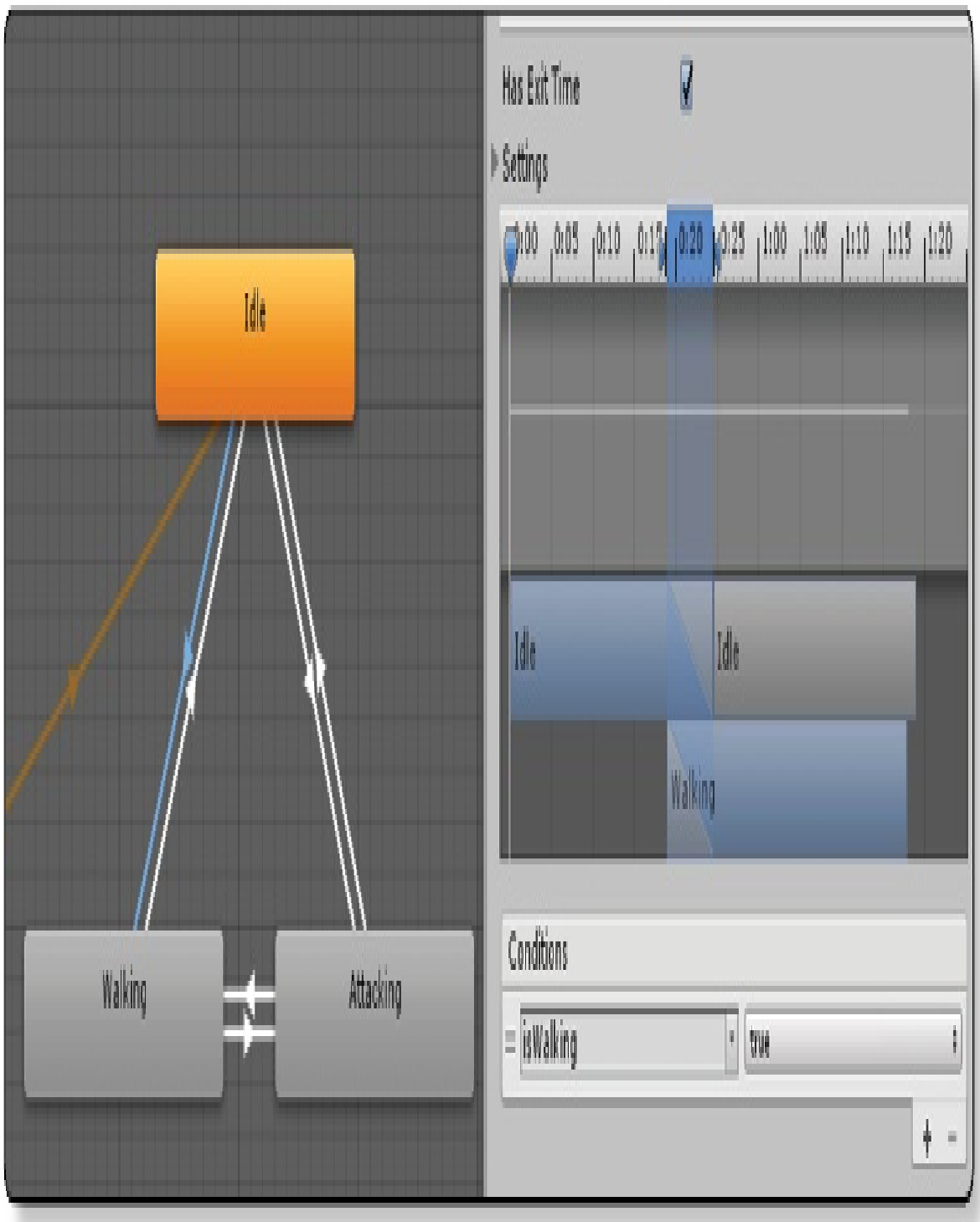
El enemigo perseguirá al “Player” pero su única animación será la de “Idle” en este punto.

Haga los tres parámetros booleanos que se muestran.

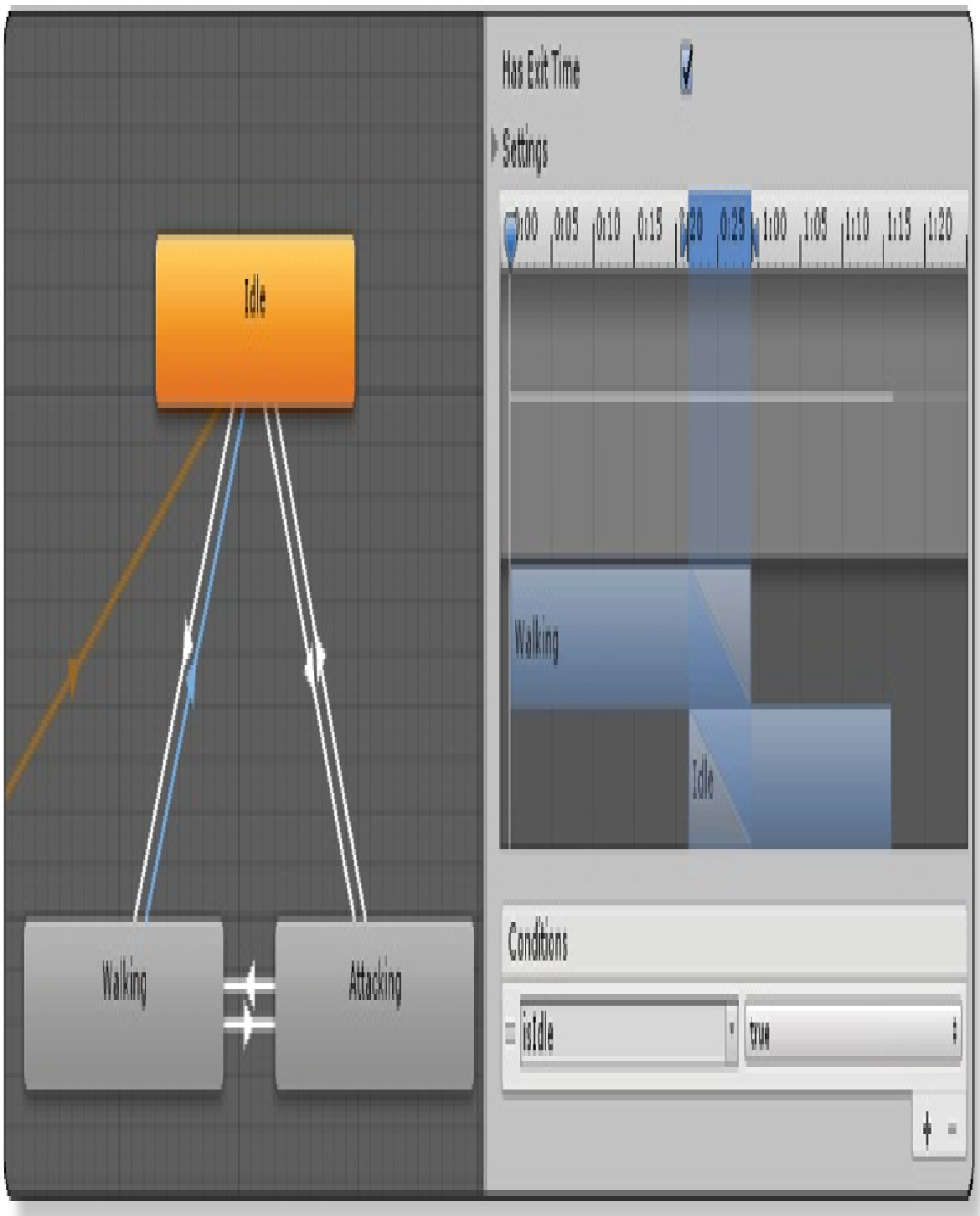
Scene
Game
Asset Stor
Layers
Parameters
Name
isWalking
isAttacking
isIdle

Parámetros booleanos de Animator

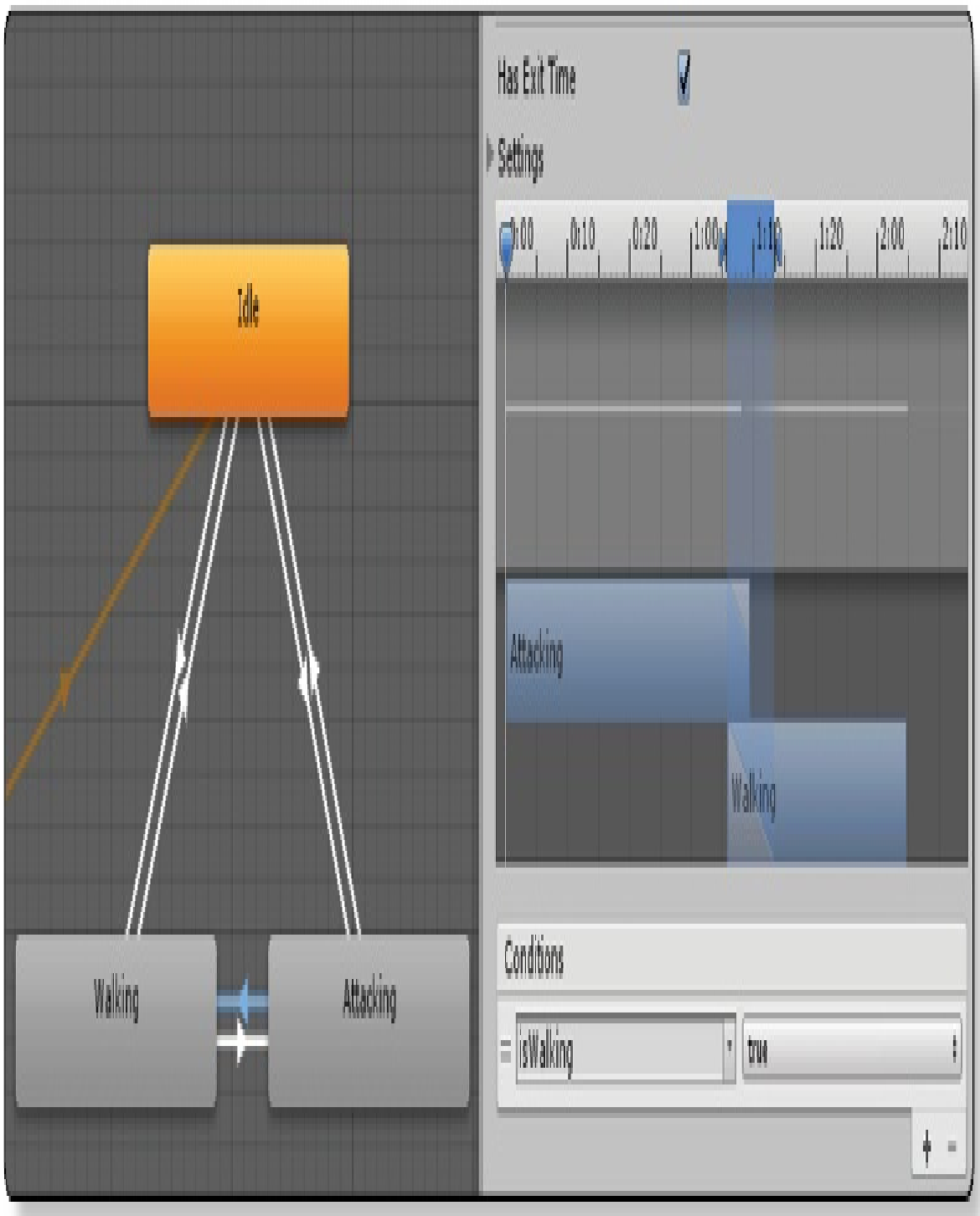
Arrastre los clips de animación que se muestran, haga las transiciones necesarias y asigne los parámetros que se indican para las transiciones.


Idle
Walking
Attacking
Has Exit Time
Settings
Idle
Idle
Walking
Conditions
isWalking
true

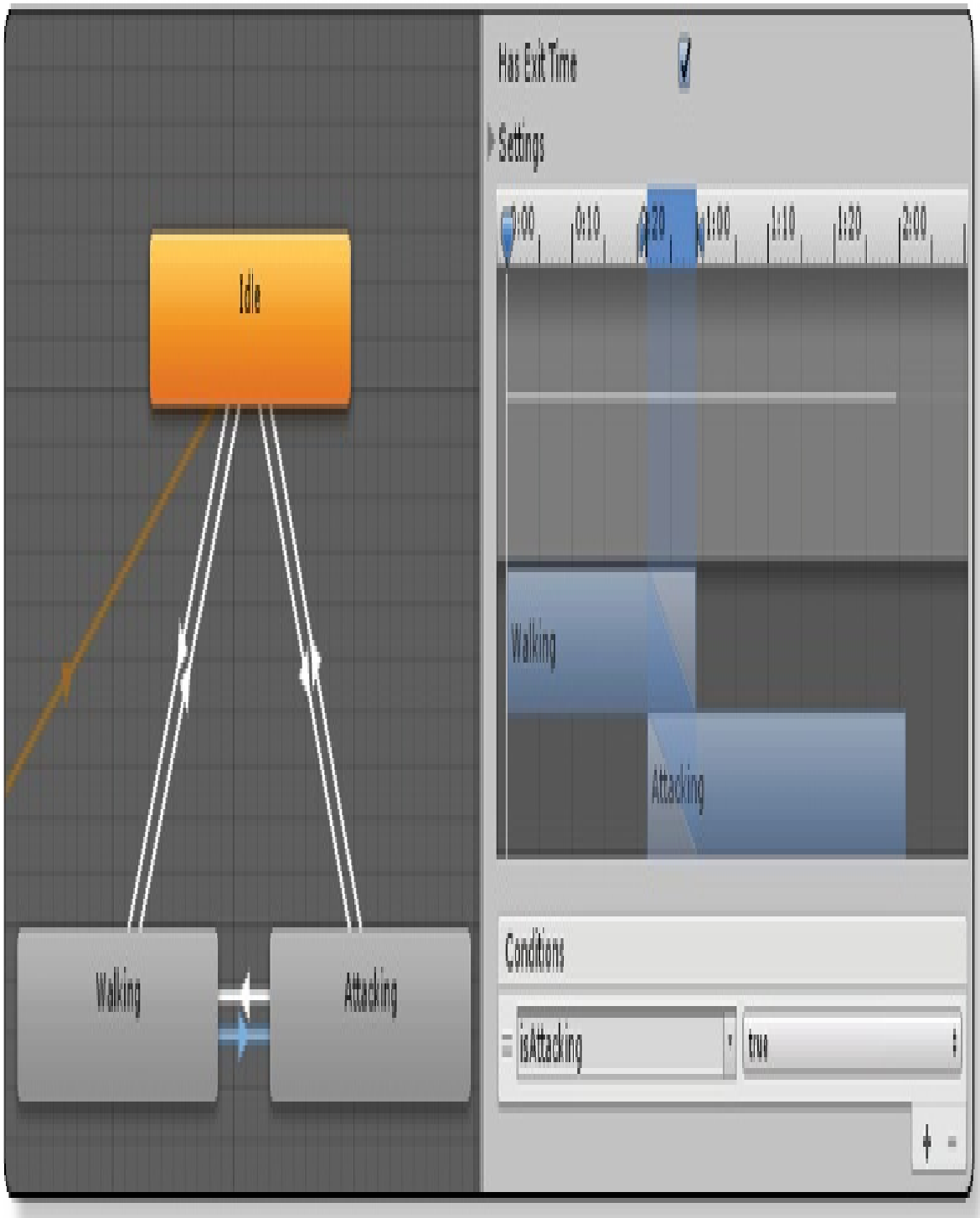
Transición de Idle a Walking


Has Exit Time
Settings
0:00
0:05
0:10
0:15
0:20
0:25
1:00
1:05
1:10
1:15
1:20
Idle
Walking
Idle
Walking
Attacking
Conditions
isIdle
true

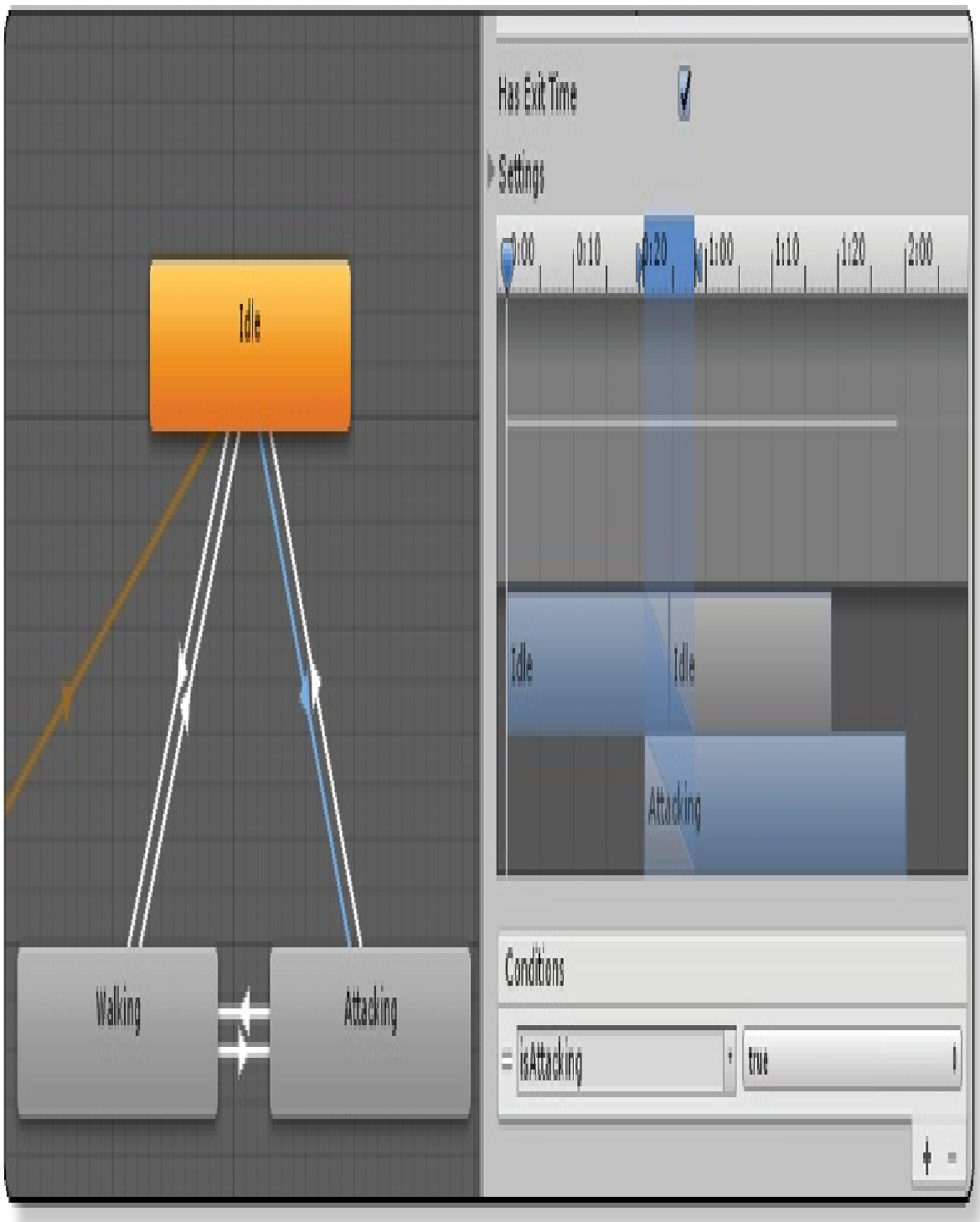
Transición de Walking a Idle


Has Exit Time
Settings
0:00
0:10
0:20
1:00
1:10
1:20
2:00
2:10
Idle
Attacking
Walking
Walking
Attacking
Conditions
isWalking
true

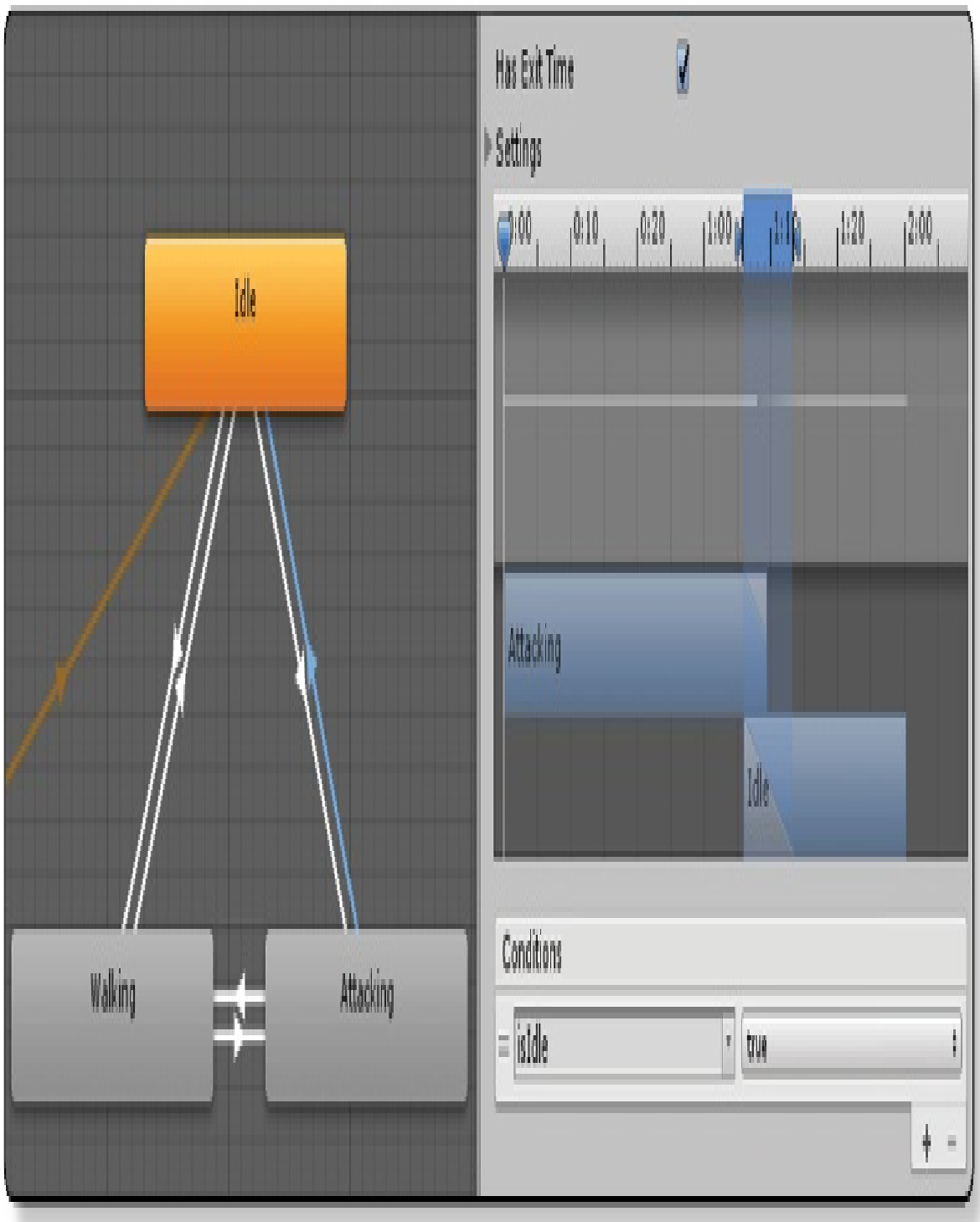
Transición de Attacking a Walking


Has Exit Time
Settings
Idle
Walking
Attacking
Conditions
isAttacking
true
Walking
Attacking

Transición de Walking a Attacking


Has Exit Time
Settings
0:00
0:10
0:20
1:00
1:10
1:20
2:00
Idle
Idle
Idle
Attacking
Walking
Attacking
Conditions
isAttacking
true

Transición de Idle a Attacking


Has Exit Time
Settings
Idle
Attacking
Idle
Walking
Attacking
Conditions
isIdle
true

Transición de Attacking a Idle

Vaya al script de “Enemy01Move” y escriba el siguiente código en la función “Update” para manejar las animaciones con los parámetros asignados.

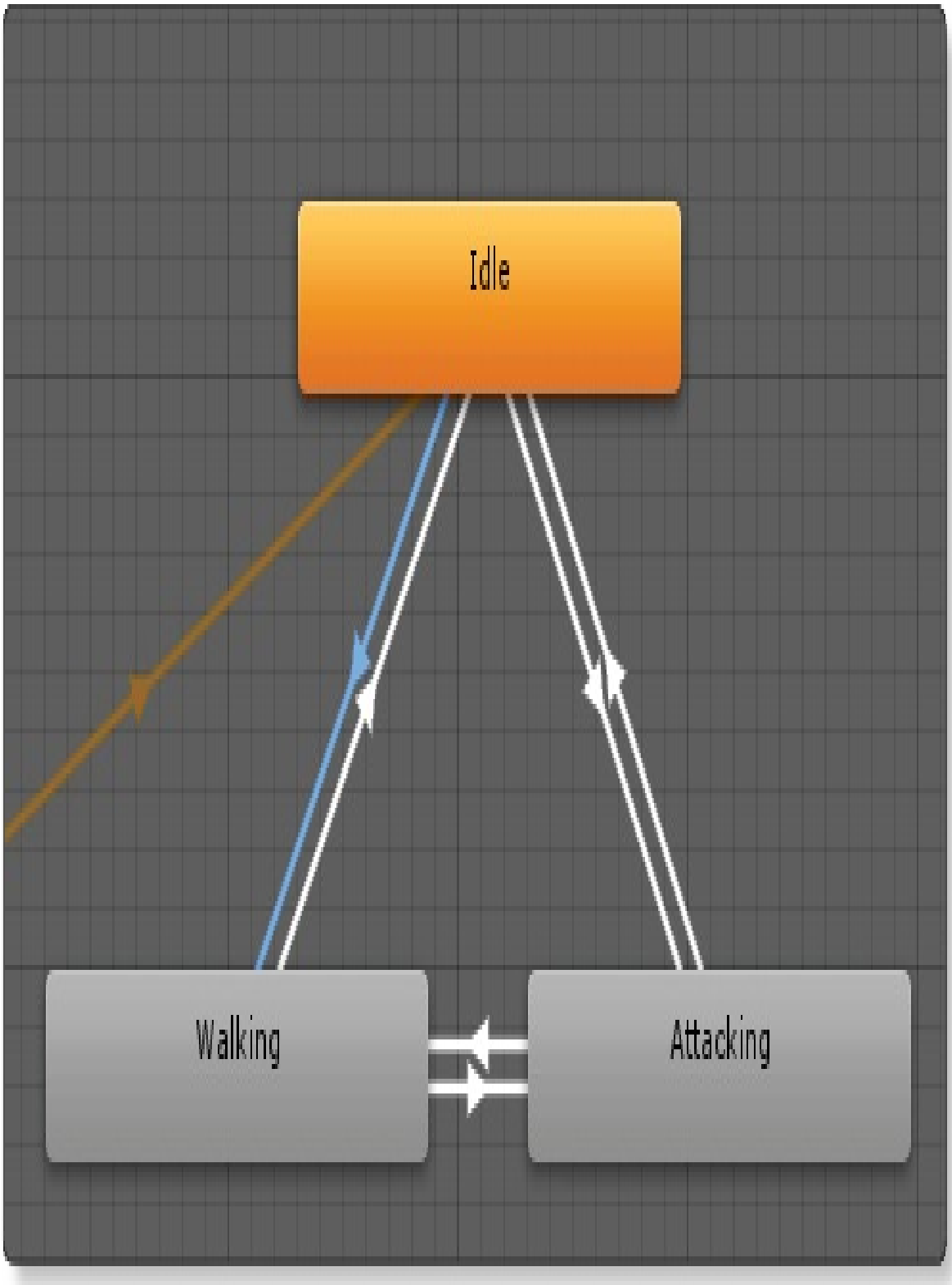
```
// Update is called once per frame
void Update () {

    if(Vector3.Distance (player.position, this.transform
        nav.SetDestination  (player.position);
        anim.SetBool ("isWalking", true);
        anim.SetBool ("isIdle", false);
    }

}
```

Comandos para arrancar parámetro de animación

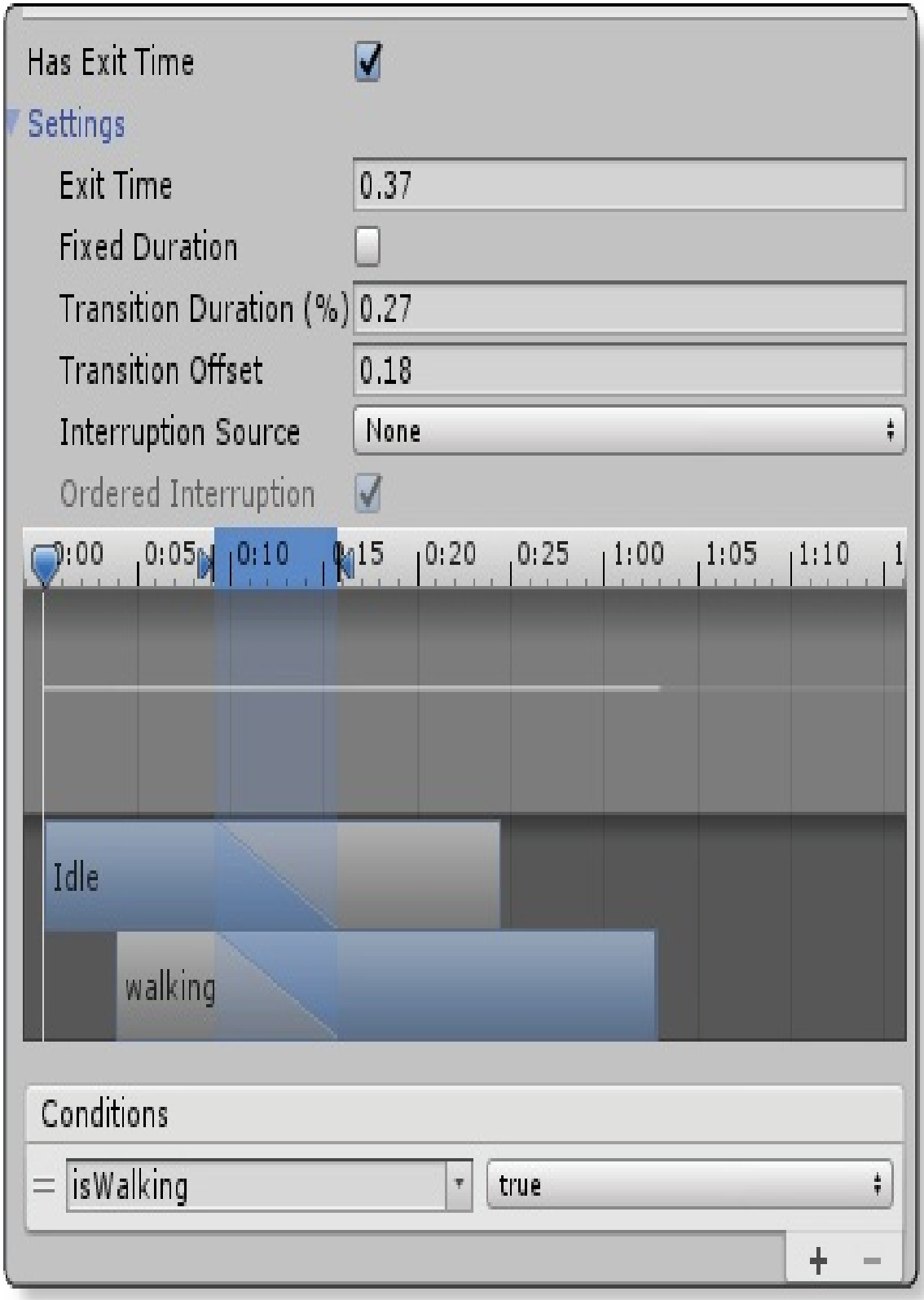
Previsualice el juego para observar la animación.

PROBLEMA: Cuando comienza a caminar no comienza inmediatamente la animación del caminado y se desplaza con animación “Idle”.

SOLUCIÓN: Seleccione la transición que se muestra y modifique sus parámetros como indica la ilustración.


Idle
Walking
Attacking

Transición de Idle a Walking


Has Exit Time
Settings
Exit Time
0.37
Fixed Duration
Transition Duration (%)
0.27
Transition Offset
0.18
Interruption Source
None
Ordered Interruption
0:00
0:05
0:10
0:15
0:20
0:25
1:00
1:05
1:10
Idle
walking
Conditions
isWalking
true

Parámetros correctos para la animación

Previsualice el juego para observar el buen funcionamiento de la animación “Walking”.

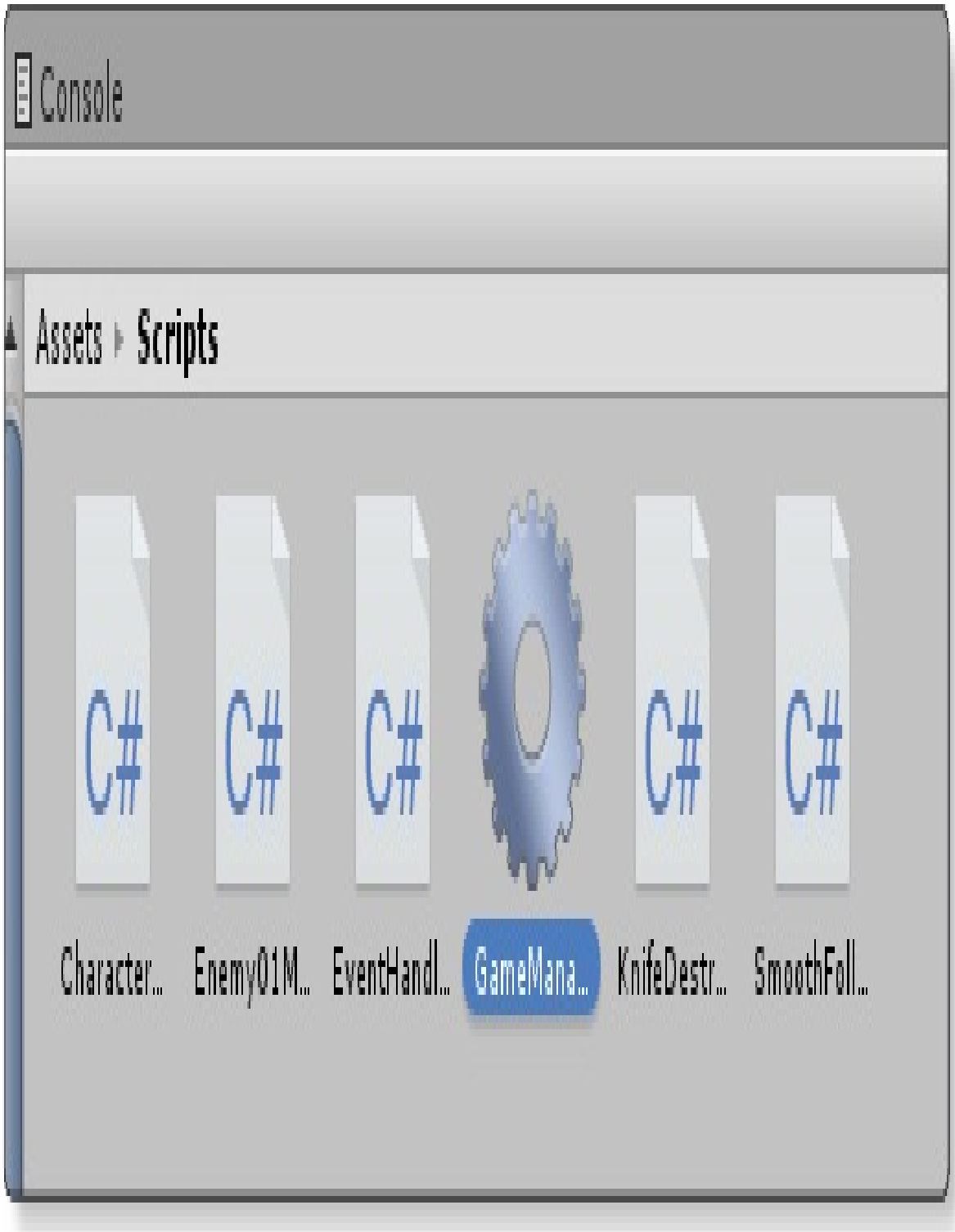
Solución animación Walking

CREACIÓN DE UN GAME MANAGER

En este apartado crearemos un “GameManager” el cual nos va a servir como opción para colocar dentro los objetos del juego privados que queremos mandar llamar en otros scripts.

Veremos la función de los “Getters” dentro de la metodología de la programación y crearemos estados de juego.

Haga un nuevo script y nómbrelo “GameManager”.


Console
Assets ▸ Scripts
Character...
Enemy01M...
EventHandl...
GameMana...
KnifeDestr...
SmoothFoll...

Icono reservado para script “GameManager”

Cuando creamos el script “GameManager” ya tiene un icono reservado el cual aparece como un engrane.

En Unity haga un nuevo “EmptyGameObject” y nómbrelo “GameManager”.

Asigne el script “GameManager” al objeto “GameManager”.

Convierta el objeto “GameManager” en “Prefab”.

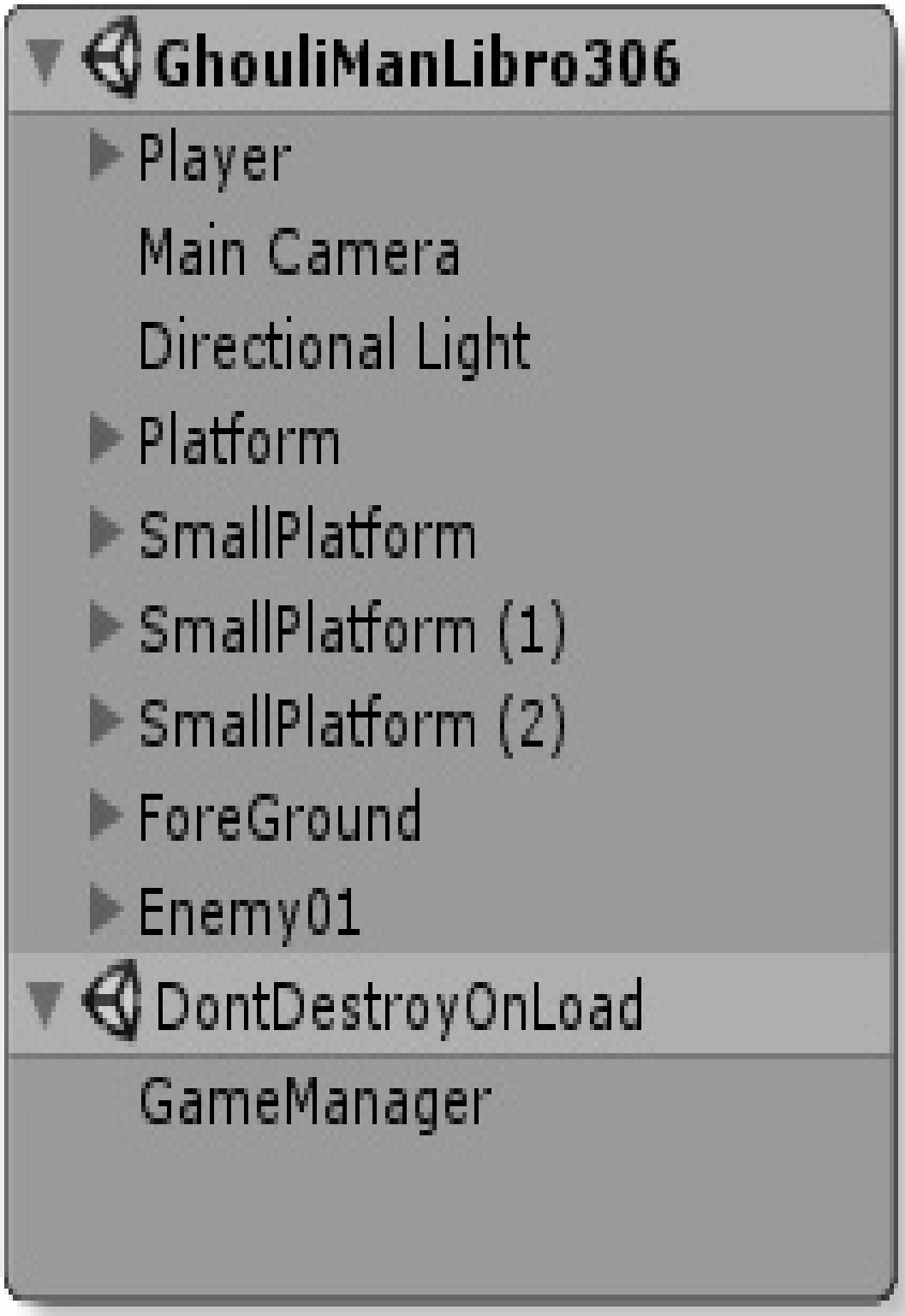
Console
Assets
Prefabs
Enemy
ForeGround
GameMana...
Knife_PFB
Platform 1
Player
SmallPlatfo...

Abra el script creado y escriba el siguiente código para hacer correctamente el “instance” que creará el “GameManager” cuando el juego despierte.

```
public class GameManager : MonoBehaviour {


    public static GameManager instance = null;


    void Awake () {
        if (instance == null) {
            instance = this;
        } else if (instance != this) {
            Destroy (gameObject);
        }
        DontDestroyOnLoad (gameObject);
    }


    // Use this for initialization
    void Start () {
```

Dado que solo existirá un solo GameManager en el juego, este código convierte a nuestro “GameManager” en un “Singleton”, y lo protegemos a través de estas líneas de código en donde nos aseguramos de que solo exista una instancia de él. De existir otro automáticamente será destruido.

Previsualice el juego para observar que cuando comienza a correr se crea el “GameManager” como hijo de “DontDestroyOnLoad”.


GhouliManLibro306
Player
Main Camera
Directional Light
Platform
SmallPlatform
SmallPlatform (1)
SmallPlatform (2)
ForeGround
Enemy01
DontDestroyOnLoad
GameManager

Vaya al script “GameManager” y haga las siguientes variables y escriba sus “Getters” correspondientes.

```
public class GameManager : MonoBehaviour {

    public static GameManager instance = null;

    [SerializeField] private GameObject player;
    private bool gameOver = false;

    public GameObject Player {
        get {return player;}
    }

    public bool GameOver {
        get {return gameOver;}
    }

    void Awake () {
        if (instance == null) {
```

Creación de “getters”

En primera instancia creamos las variables privadas.

Queremos mandar llamar estas variables a través de otro script, pero es imposible ya que se encuentran privadas.

La solución a esto es crear su “getter”.

public GameObject Player {
get {return player;}
}

Inicialización del “getter”

El “getter” lo establecemos como público y le asignamos el mismo nombre que la variable, pero con la primera letra mayúscula.

public GameObject Player {
get {return player;}
}

Regreso de la variable

En seguida escribimos “get” y dentro le decimos que regrese la variable con “return” y el nombre original de la variable (con minúscula).

NOTA:

Para mandar llamar la variable en otro script, la llamamos con el nombre de su “getter”, con la primera letra mayúscula.

Haga la siguiente función.

```
    // Update is called once per frame
    void Update () {

    }

    public void PlayerHit (int currentHP){

        if (currentHP > 0) {
            gameOver = false;
        } else {
            gameOver = true;
        }
    }
}
```

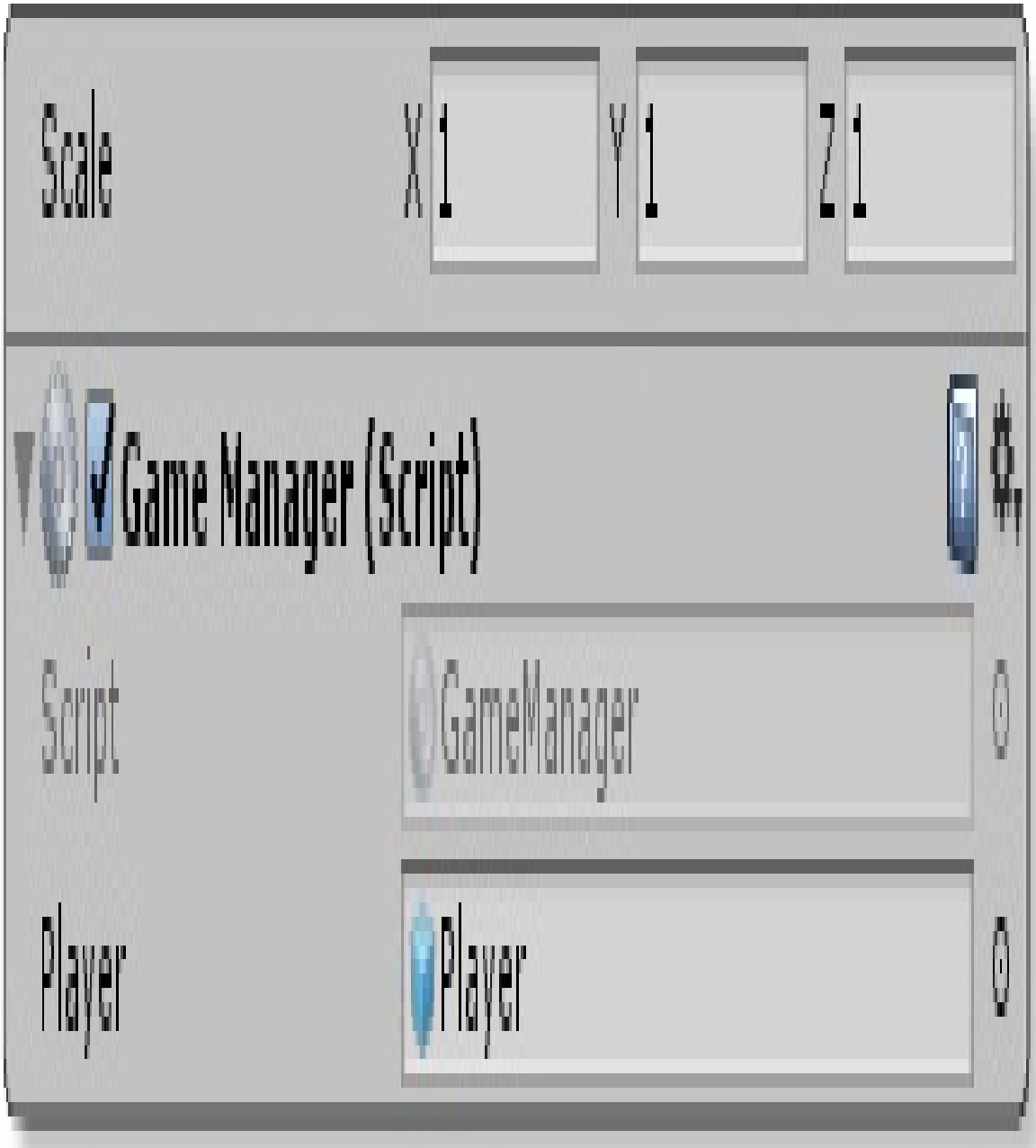
Función “PlayerHit”

Al crear la función estamos colocando como parámetro una nueva variable tipo “int” y llamada “currentHP”. Esta variable todavía no tiene un valor establecido, únicamente estamos indicando que cuando esta variable sea mayor a cero, entonces la variable “gameOver” tomará un valor falso. Y de otra manera tomará el valor de verdadera.

NOTA:

La variable “gameOver” en esta función no necesitamos llamarla con su “getter” ya que pertenece al mismo script donde la creamos.

Seleccione el Prefab “GameManager” y en su componente script asigne el objeto “Player” desde “Hierarchy”.


Scale
X 1
Y 1
Z 1
Game Manager (Script)
Script
GameManager
Player
Player

Asignación de GameObject en el script

Previsualice el juego para comprobar que todo corre normalmente.

PROGRAMACIÓN DEL ATAQUE DEL ENEMIGO 01

En este apartado nos ocuparemos de programar la funcionalidad de ataque de nuestro enemigo 01.

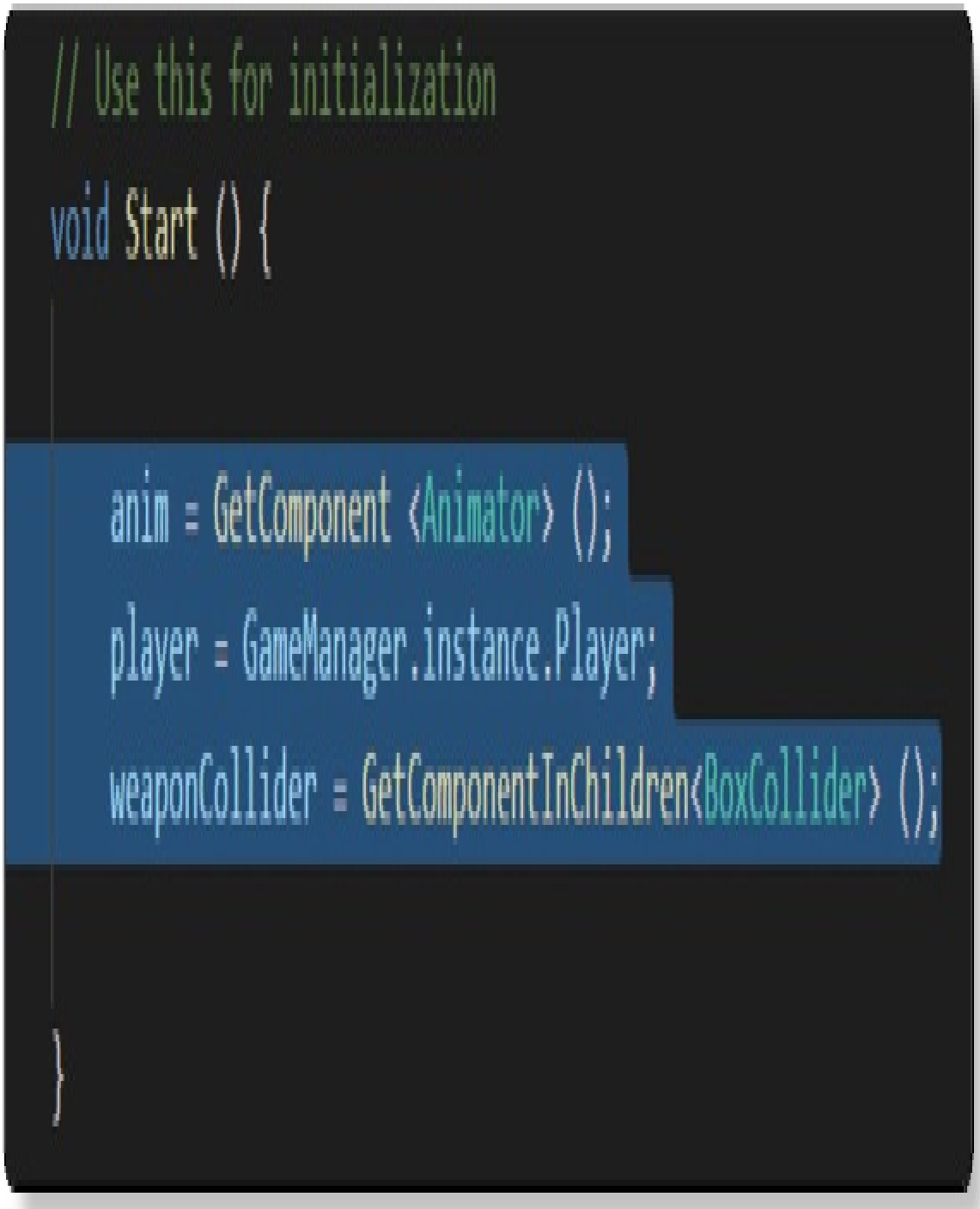
Veremos un nuevo concepto dentro de la programación que es el “IEnumerator” y el “StartCoroutine” que son funciones que nos van a servir para establecer intervalos de tiempos para que pase una acción o para que la retrase.

Haga un nuevo script y nómbrelo “Enemy01Attack”, asígnelo al enemigo, ábralo y escriba las siguientes variables.

```
public class Enemy01Attack : MonoBehaviour {


    [SerializeField] private  float range = 3f;
    [SerializeField] private float timeBetweenAttacks =1f;


    private Animator anim;
    private GameObject player;
    private bool playerInRange;
    private BoxCollider weaponCollider;


    // Use this for initialization
    void Start () {
```

Creación de variables

Inicialice las siguientes variables.


// Use this for initialization
void Start () {
anim = GetComponent <Animator> ();
player = GameManager.instance.Player;
weaponCollider = GetComponentInChildren<BoxCollider> ();
}

Inicialización de variables

La variable “player” la estamos llamando desde el script “GameManager” a través de su “getter” “Player”, creando una instancia del objeto.

En la variable “weaponCollider”, dado que el BoxCollider del arma se encuentra en el “joint” hijo del objeto al que está enlazado nuestro script, por ello lo inicializamos con “GetComponentInChildren”.

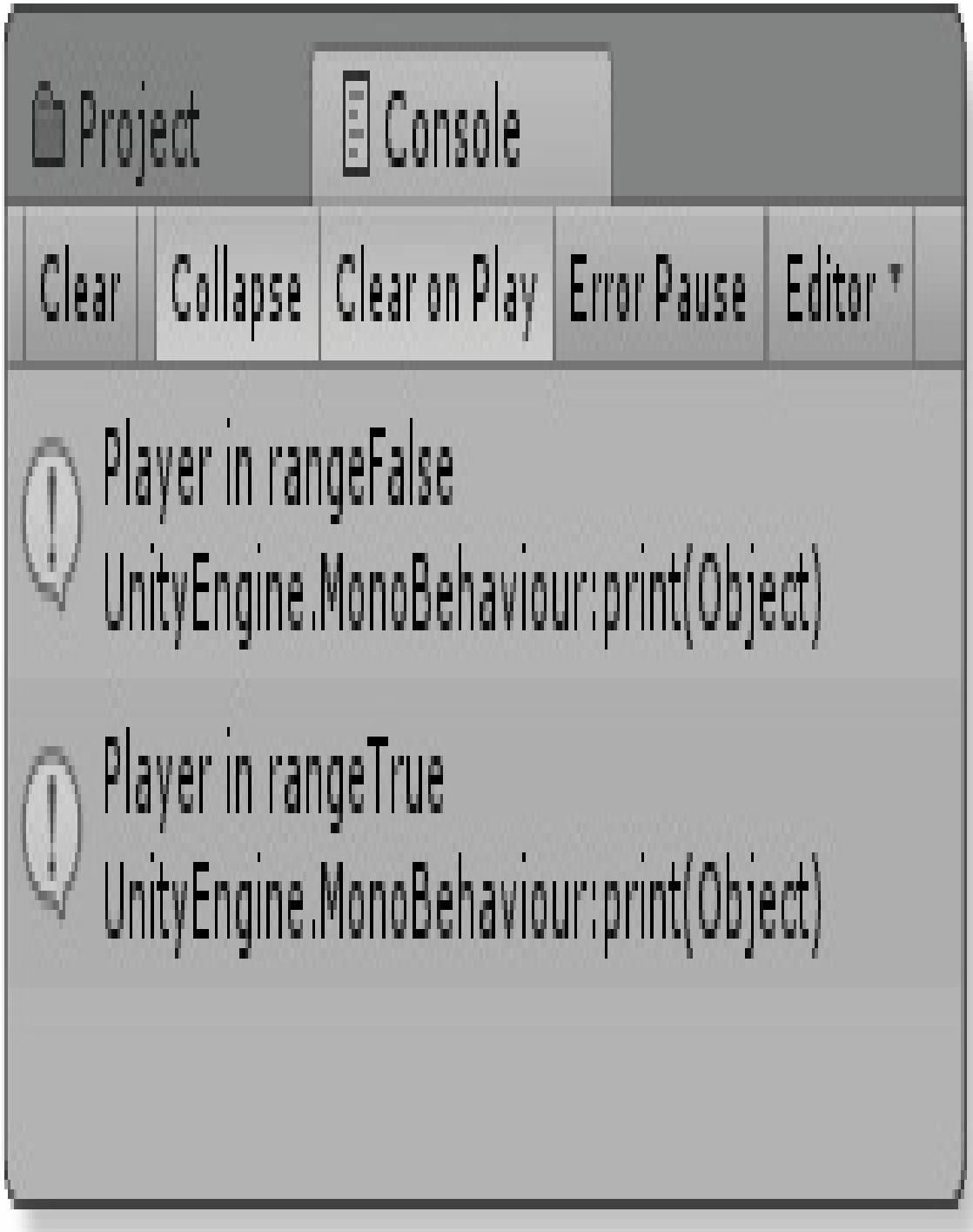
Escriba el siguiente código en la función “Update” para asignar el valor a la variable “playerInRange” así como comprobar que nuestro “Player” se encuentra en el rango para el ataque.

```
// Update is called once per frame
void Update () {


    if (Vector3.Distance (transform.position, player.transform.position) < range) {
        playerInRange = true;
    } else {
        playerInRange = false;
    }
    print ("Player in range" + playerInRange);


}
```

Función para comprobar el rango del player

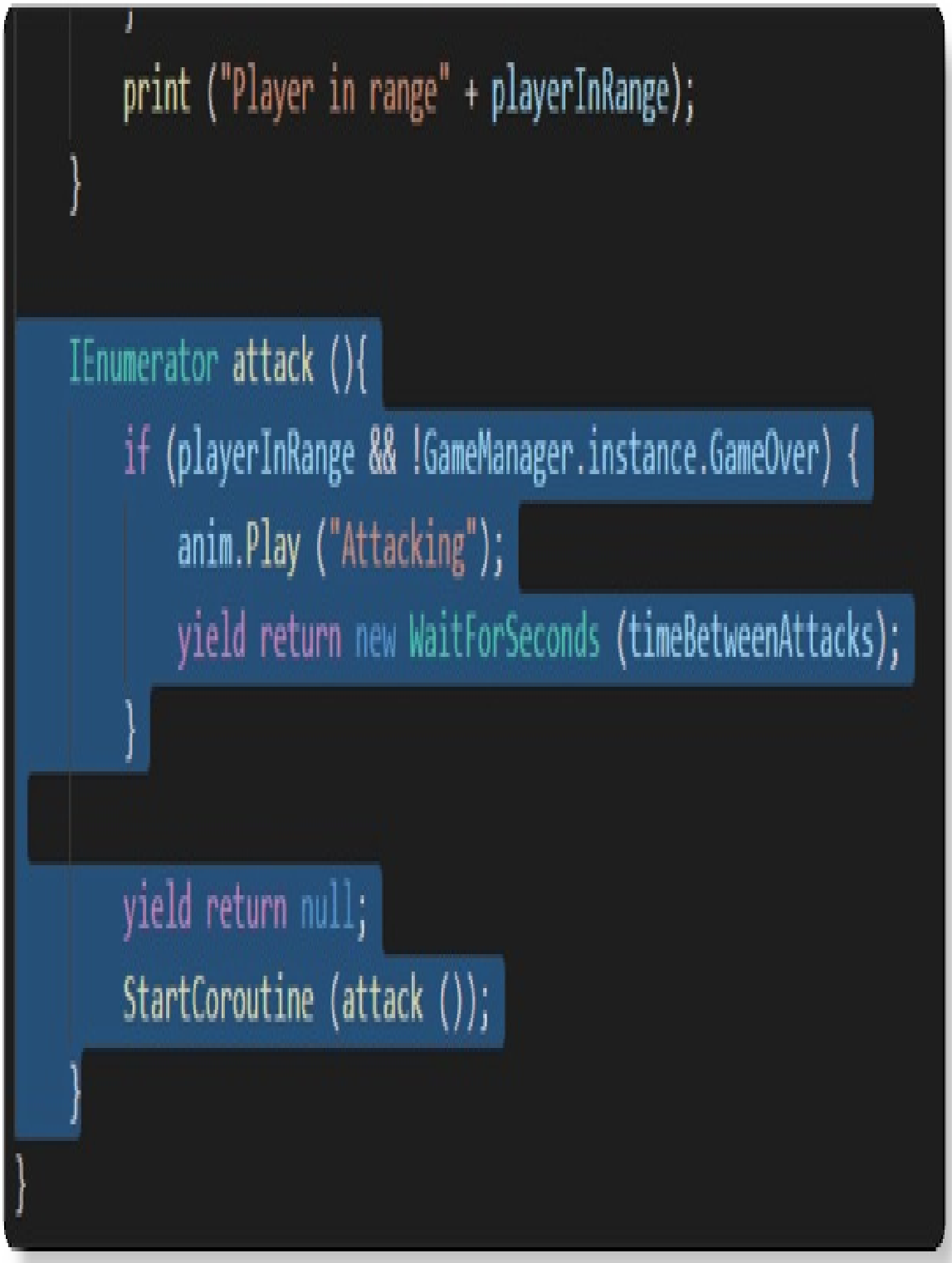
En el comando “print” estamos escribiendo dentro de las comillas lo que queremos que se imprima textualmente, fuera de las comillas estamos escribiendo el nombre de la variable, lo que dará por resultado que se imprima el valor de la variable.

Previsualice el juego para observar que se imprime la línea al momento en que el “Player” se encuentre cerca del enemigo.


Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
Player in rangeFalse
UnityEngine.MonoBehaviour:print(Object)
Player in rangeTrue
UnityEngine.MonoBehaviour:print(Object)

Comprobación del rango del Player

Escriba la siguiente función tipo “IEnumerator” para correr la animación de ataque conforme al intervalo de tiempo de la variable “timeBetweenAttacks”.


print ("Player in range" + playerInRange);
}
IEnumerator attack (){
if (playerInRange && !GameManager.instance.GameOver) {
anim.Play ("Attacking");
yield return new WaitForSeconds (timeBetweenAttacks);
}
yield return null;
StartCoroutine (attack ());
}
}

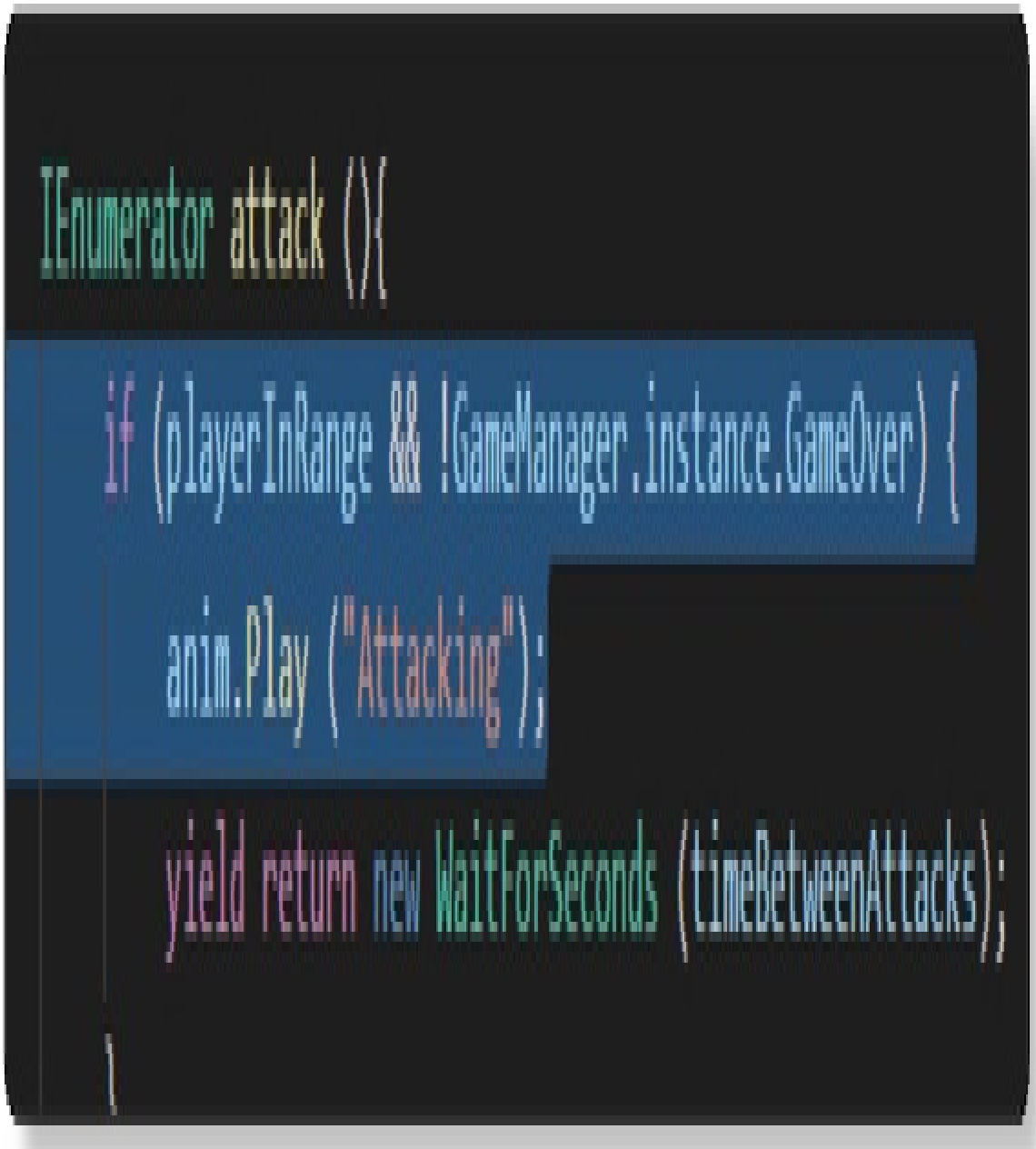
Función IEnumerator

La función tipo “IEnumerator” la utilizamos cuando necesitamos añadir un intervalo de tiempo a una acción.

IEnumerator attack (){
if (playerInRange &&

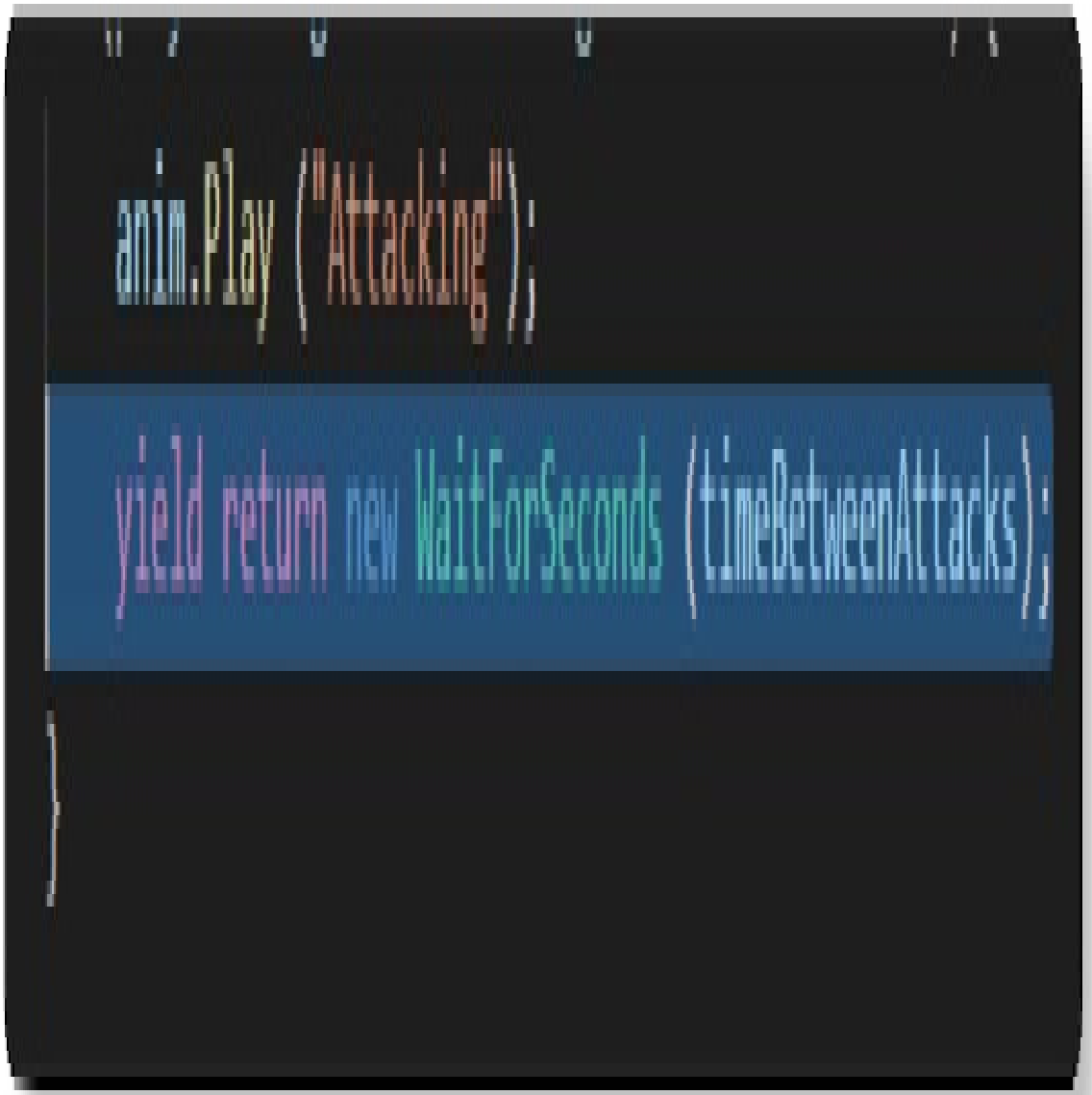
Creación de la función

En primera instancia se creará la función con la nomenclatura “IEnumerator” seguida del nombre de la función, en este caso “attack”.


IEnumerator attack (){
if (playerInRange && !GameManager.instance.GameOver) {
anim.Play ("Attacking");
yield return new WaitForSeconds (timeBetweenAttacks);
}

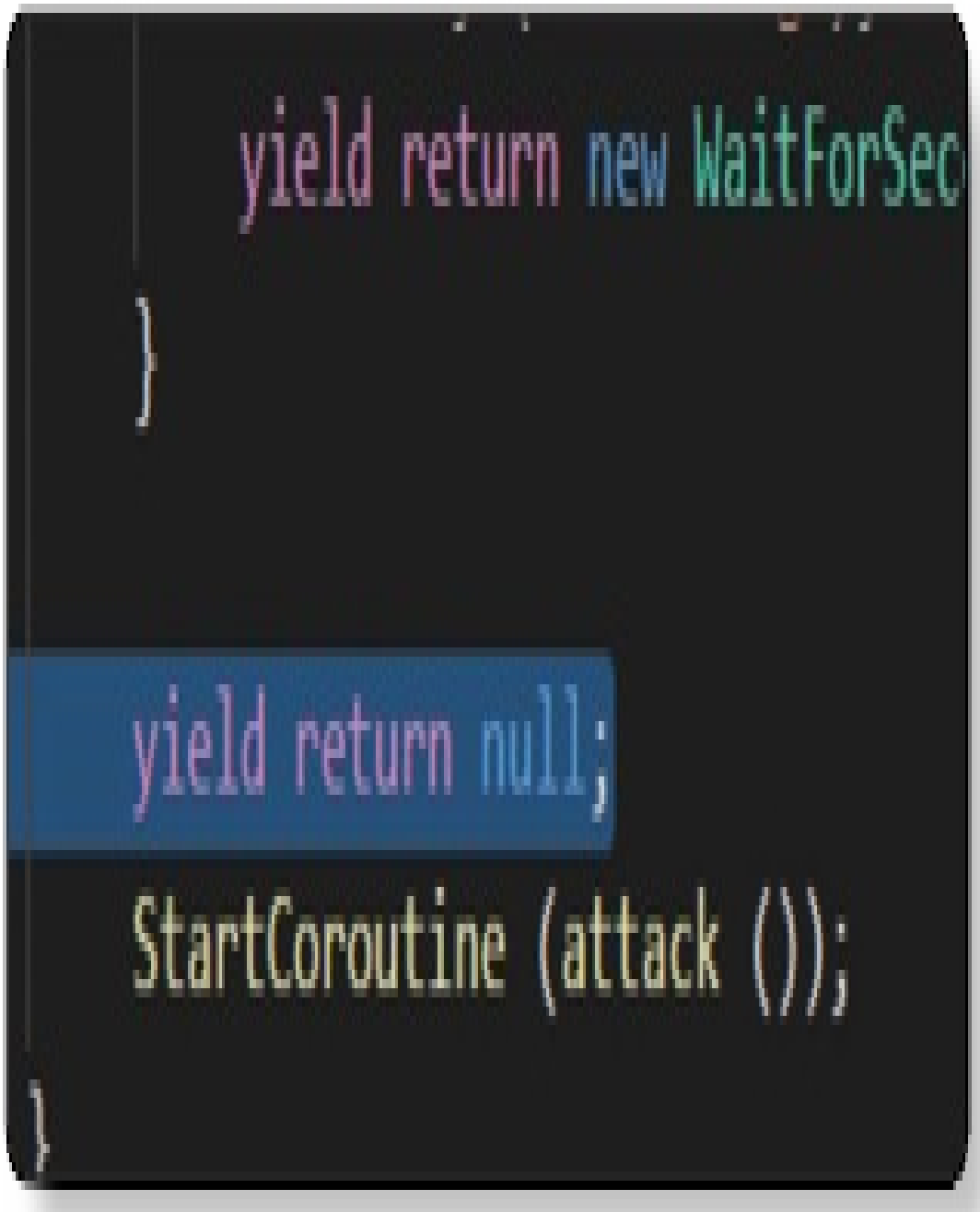
Estructura condicional

En este caso queremos correr la animación de ataque, en primera instancia cuando la condición “Si el “Player” está en rango y la instancia del GameManager NO es GameOver, entonces se correrá la animación “Attacking” con el comando “Play”.


anim.Play("Attacking");
yield return new WaitForSeconds(timeBetweenAttacks);
}

Intervalo de tiempo

En seguida se establece un nuevo tiempo en segundos para volver a realizar la comprobación del “if” con un parámetro de los segundos a esperar, en este caso el valor de la variable “timeBetweenAttacks”.


yield return new WaitForSec
}
yield return null;
StartCoroutine (attack ());
}

Regreso nulo

Como parte del código, debemos de escribir “yield return null” de otra manera nos aparecerá un error en el código:

yield return null;
StartCoroutine (attack ());

Arranque de la función

Por sí sola la función “IEnumerator” no toma acción hasta que sea llamada con el comando “StartCoroutine” seguida y dentro de paréntesis el nombre de la función, en este caso “attack()”.

También en este caso estamos llamando la acción del “StartCoroutine” dentro de la función “IEnumerator” para que se realice, pero como es parte de la misma función “IEnumerator” debemos de mandarla llamar desde otra función que esté sucediendo en el juego.

Vaya a la función “Start” y mande llamar la función creada.

```
void Start () {


    anim = GetComponent <Animator> ();
    player = GameManager.instance.Player;
    weaponCollider = GetComponentInChildren<BoxCollider> ();
    StartCoroutine (attack ());
```

Arranque de la función en “Start”

Previsualice el juego para observar que cuando el player este en rango, comienza la animación de ataque del enemigo.

Animación de ataque

PROGRAMACIÓN DEL DAÑO AL PLAYER

En este apartado programaremos el sistema para que el “Player” reciba daño. Para esto debemos de crear un sistema de salud para nuestro “Player” así como generar y programar sus animaciones de daño y muerte.

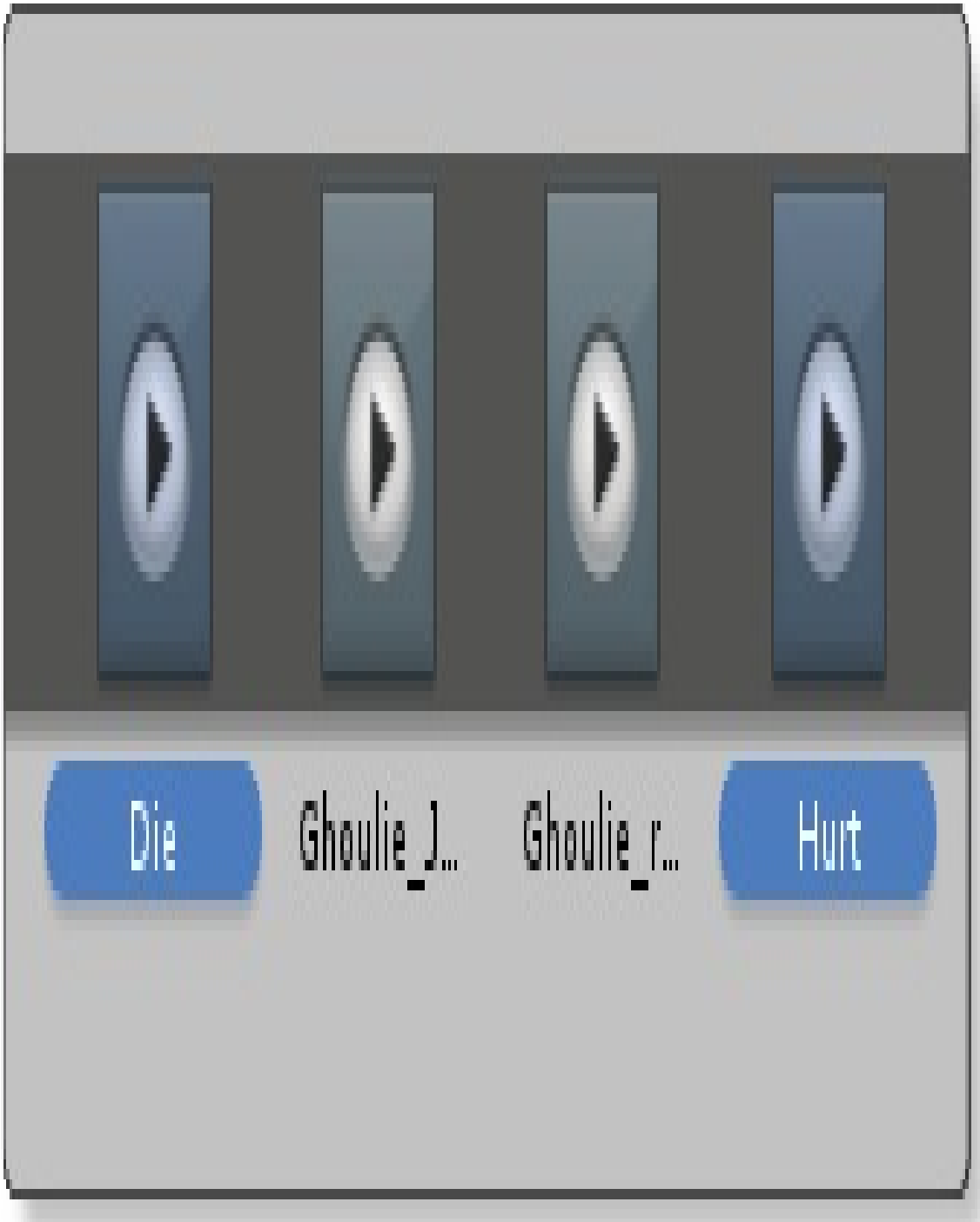
Haga un nuevo “Tag” y nómbrelo “Weapon”.

Asigne el nuevo “Tag” al “joint” que contiene el “BoxCollider” del arma del enemigo.

Haga los clips de animación de daño y de muerte del “Player” tomando en cuenta los siguientes valores y su nombre correspondiente.

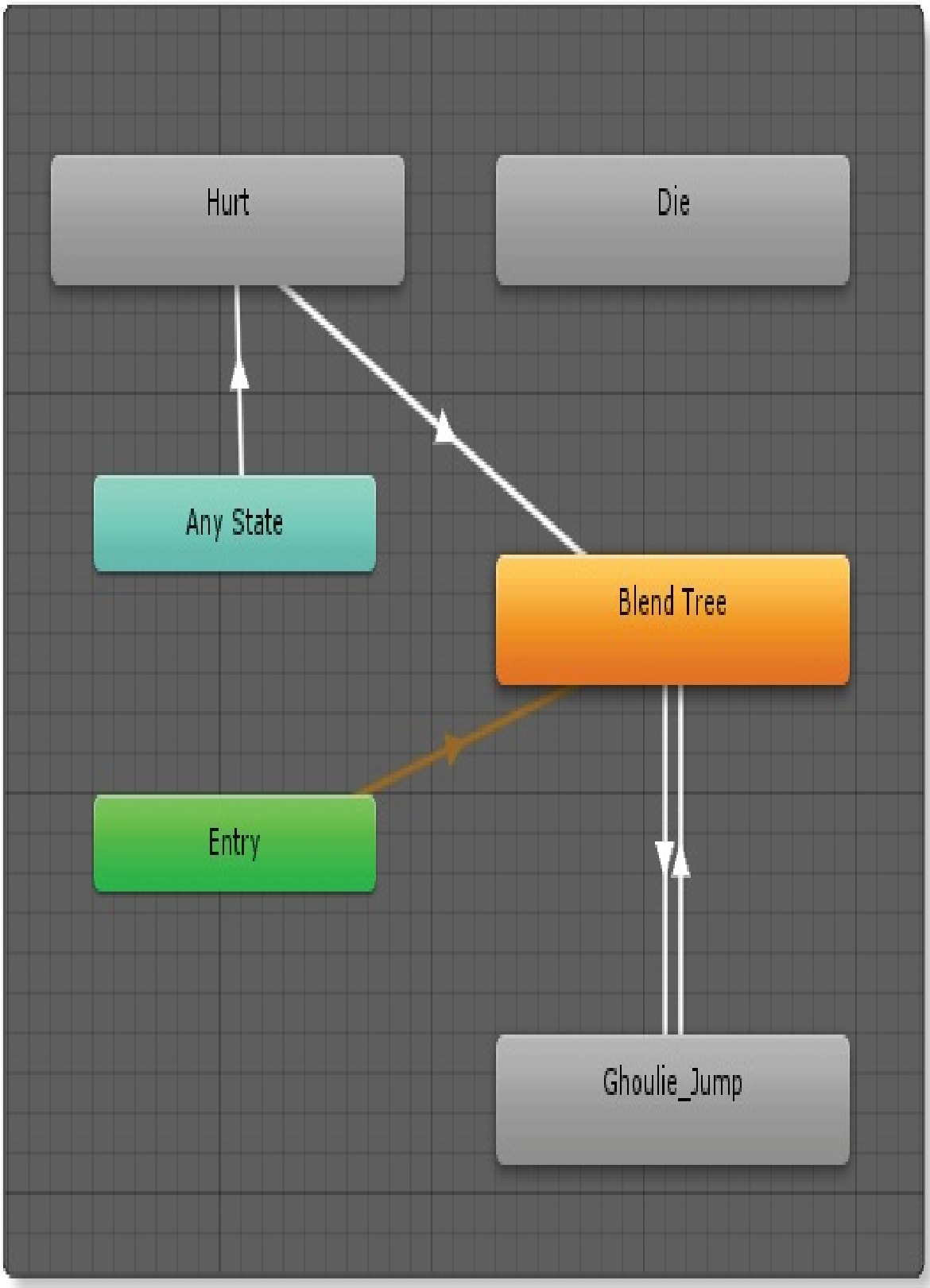
frame: 0090-0116 murderer death (Die)

frame: 0165-0209 get hit from front (Hurt)


Die
Ghoulie_l...
Ghoulie_r...
Hurt

Clips de animación de Player

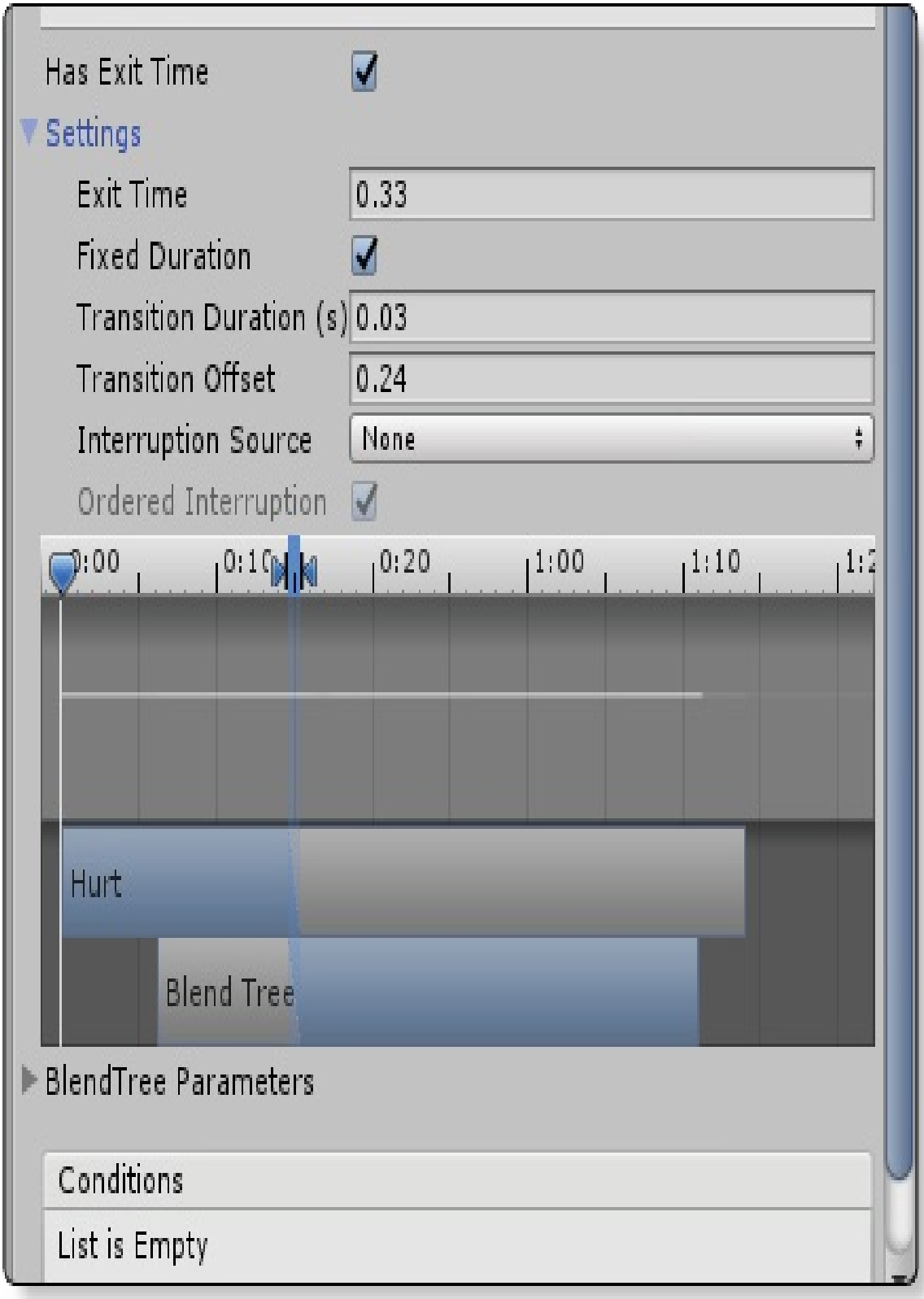
Vaya a “Animator” y coloque los nuevos clips de animación n y haga las transiciones mostradas.


Hurt
Die
Any State
Blend Tree
Entry
Ghoulie_Jump

Transiciones de animación

Cuando hacemos una transición desde “Any State” significa que la podemos mandar llamar desde cualquier estado.

Seleccione la transición de “Hurt” a “Blend Tree” y asigne los siguientes parámetros.


Has Exit Time
Settings
Exit Time
0.33
Fixed Duration
Transition Duration (s)
0.03
Transition Offset
0.24
Interruption Source
None
Ordered Interruption
0:00
0:10
0:20
1:00
1:10
Hurt
Blend Tree
BlendTree Parameters
Conditions
List is Empty

Parámetros de transición

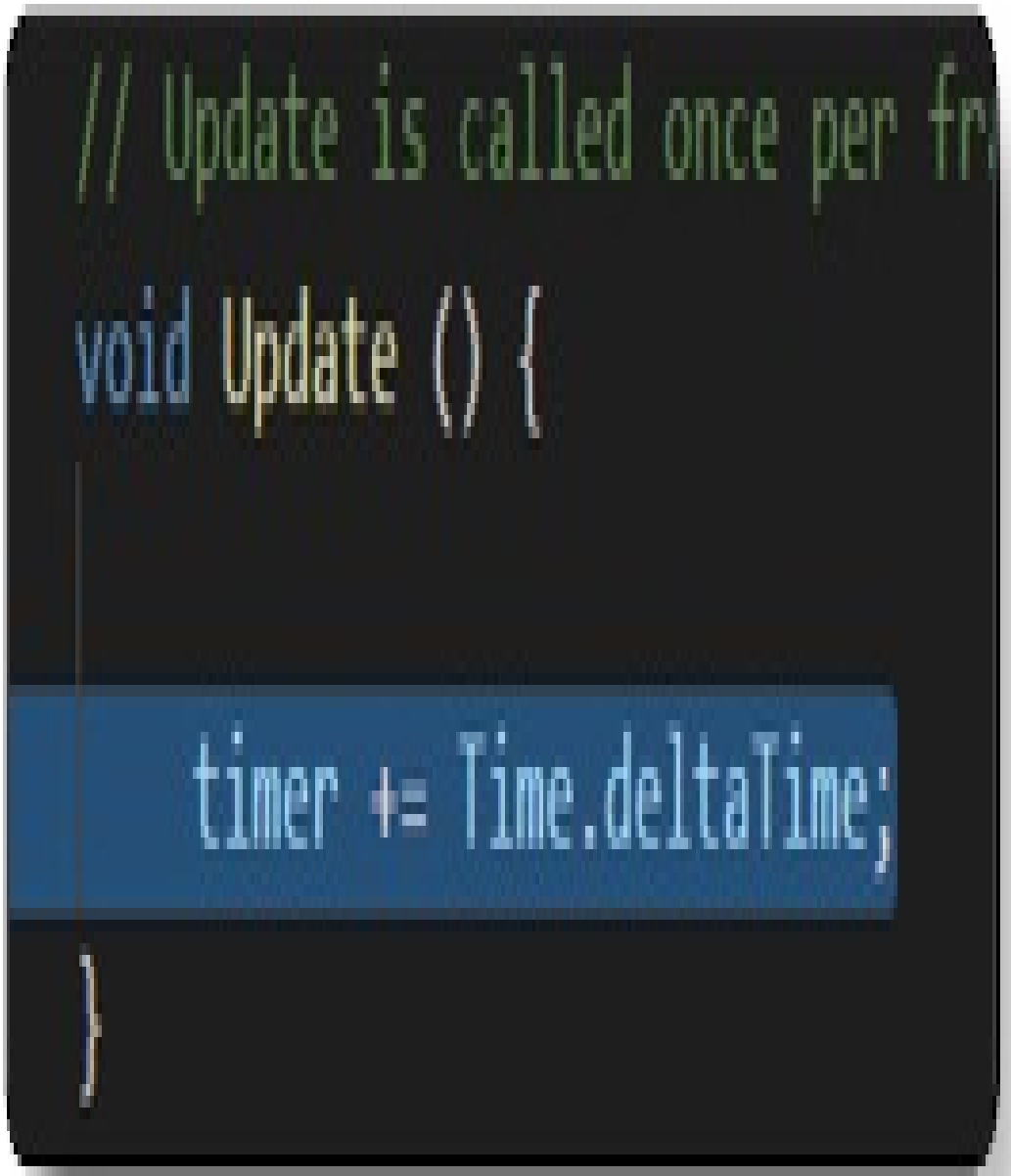
Haga un nuevo script, nómbrelo “PlayerHealth” y asígnelo al “Player”.

Abra el script para hacer las siguientes variables inicialícelas.

```
public class PlayerHealth : MonoBehaviour {

    [SerializeField] int startingHealth = 100;
    [SerializeField] float timeSinceLastHit = 2.0f;
    [SerializeField] int currentHealth;
    [SerializeField]private float timer = 0f;
    private Animator anim;

    // Use this for initialization
    void Start () {

        anim = GetComponent<Animator> ();
        currentHealth = startingHealth;
```

Declaración e inicialización de variables

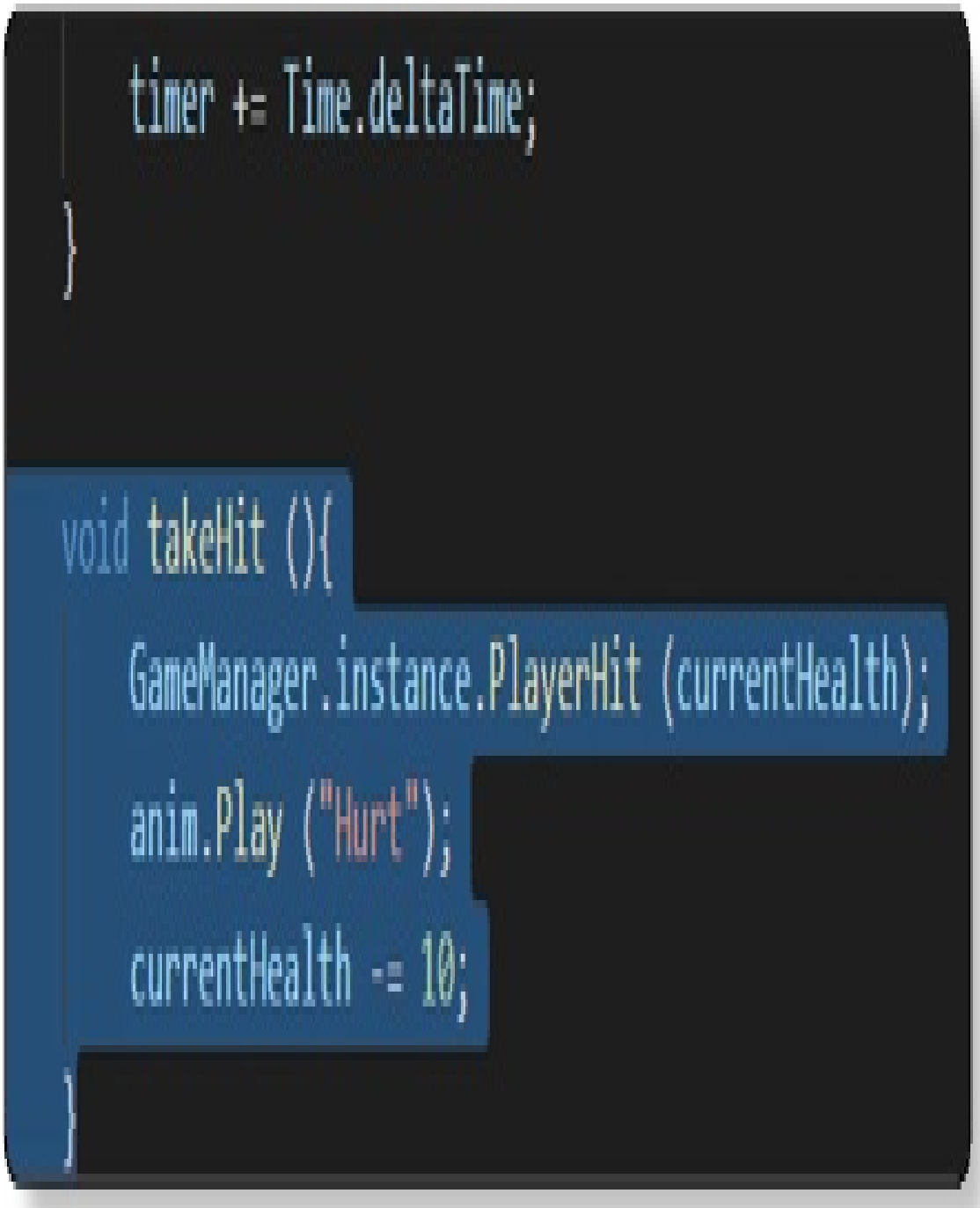
Escriba el siguiente código en la función “Update”.


// Update is called once per fr
void Update () {
timer += Time.deltaTime;
}

Inicialización del timer

De esta manera estamos inicializando la variable “timer” conforme al tiempo real de nuestro juego.

Escriba la siguiente función:


timer += Time.deltaTime;
}
void takeHit (){
GameManager.instance.PlayerHit (currentHealth);
anim.Play ("Hurt");
currentHealth -= 10;
}

.Función takeHit

Estamos mandando llamar desde el script “GameManager” la función “PlayerHit” y estamos colocando como parámetro la variable “currentHealth” previamente declarada.

Al correr esta función correrá la animación “Hurt” y le restará 10 puntos a la variable “currentHealth”.

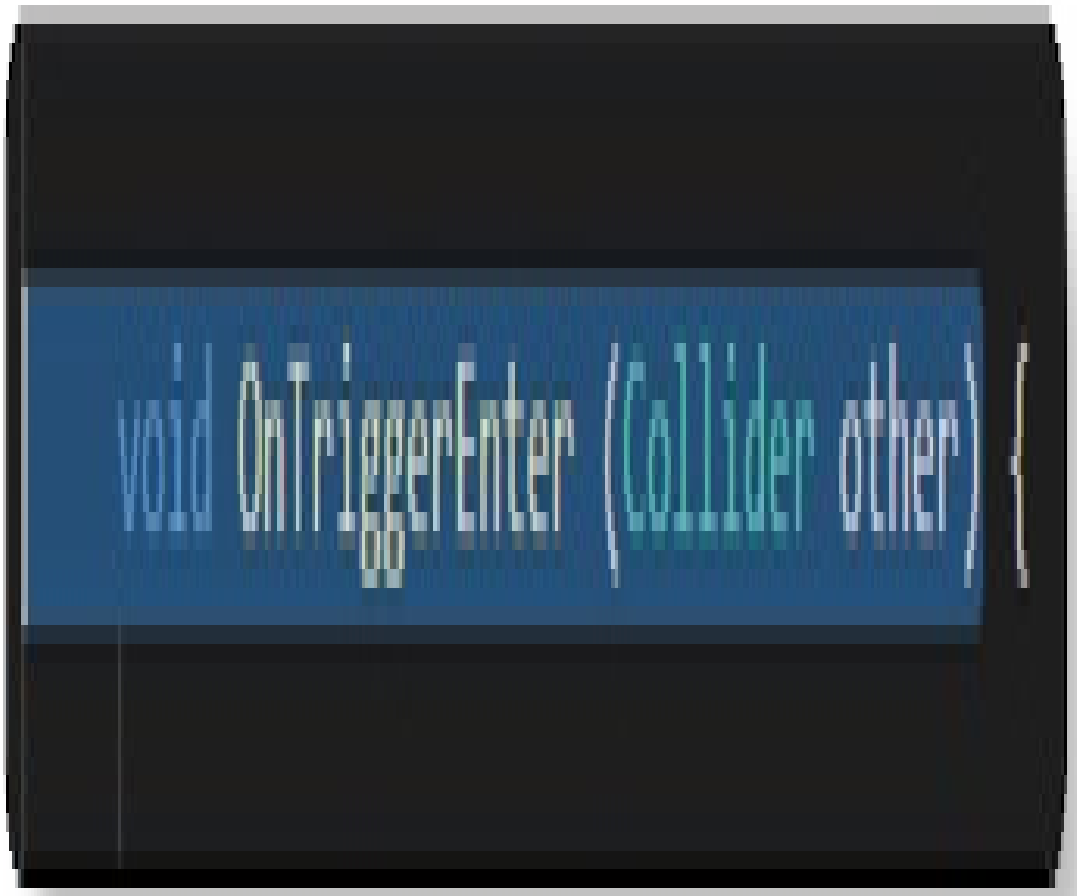
Vaya a la función “Update” y escriba la siguiente estructura condicional para mandar llamar la función “takeHit” creada.

```
        timer += Time.deltaTime;
    }


    void OnTriggerEnter (Collider other) {

        if (timer >= timeSinceLastHit && !GameManager.instance.GameOver) {

            if (other.tag == "Weapon") {
                takeHit ();
                timer = 0;
            }
        }
    }

    void takeHit (){
        GameManager.instance.PlayerHit (currentHealth);
```

Función OnTriggerEnter

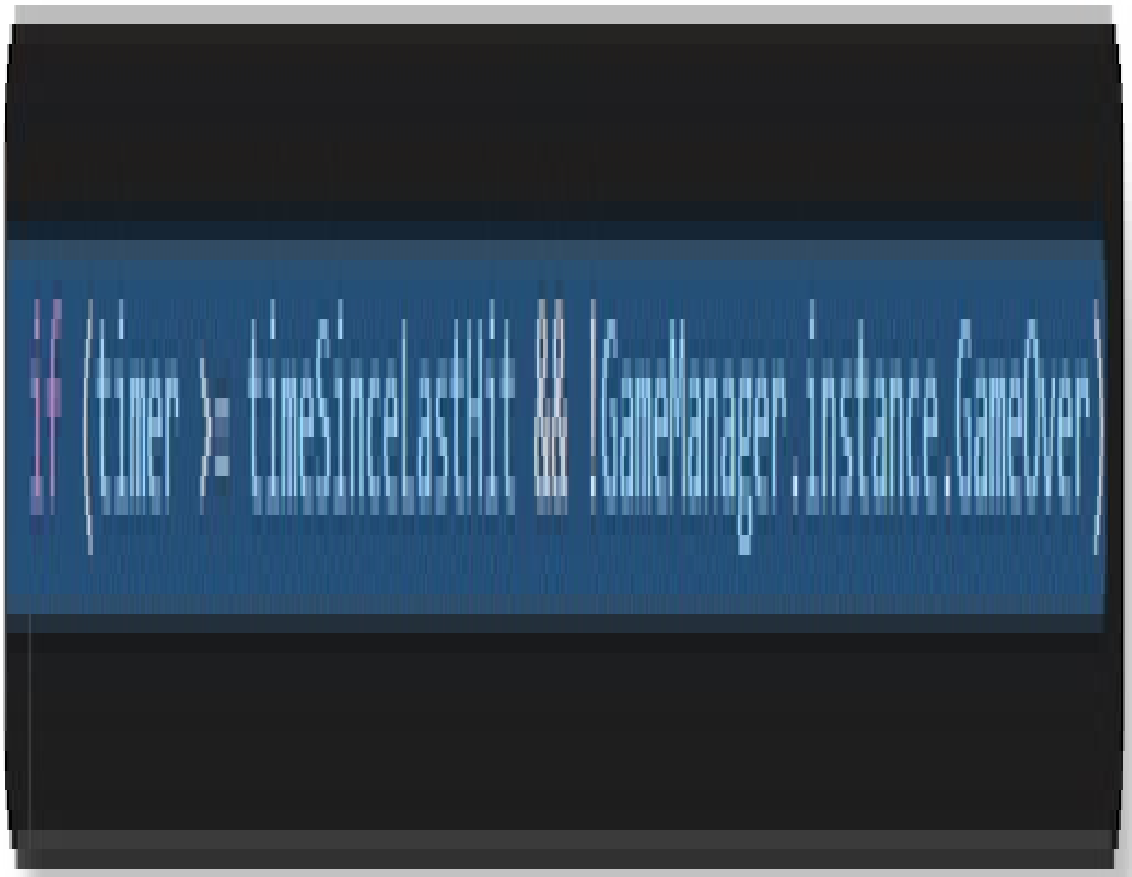
La función “OnTriggerEnter” es una función con acciones preestablecidas la cual nos indica que sucederá cuando un objeto con la propiedad “Trigger” colisione con él.

Al asignar (Collider other), estamos indicando que su colindancia será con algún otro objeto.


void OnTriggerEnter (Collider other) {

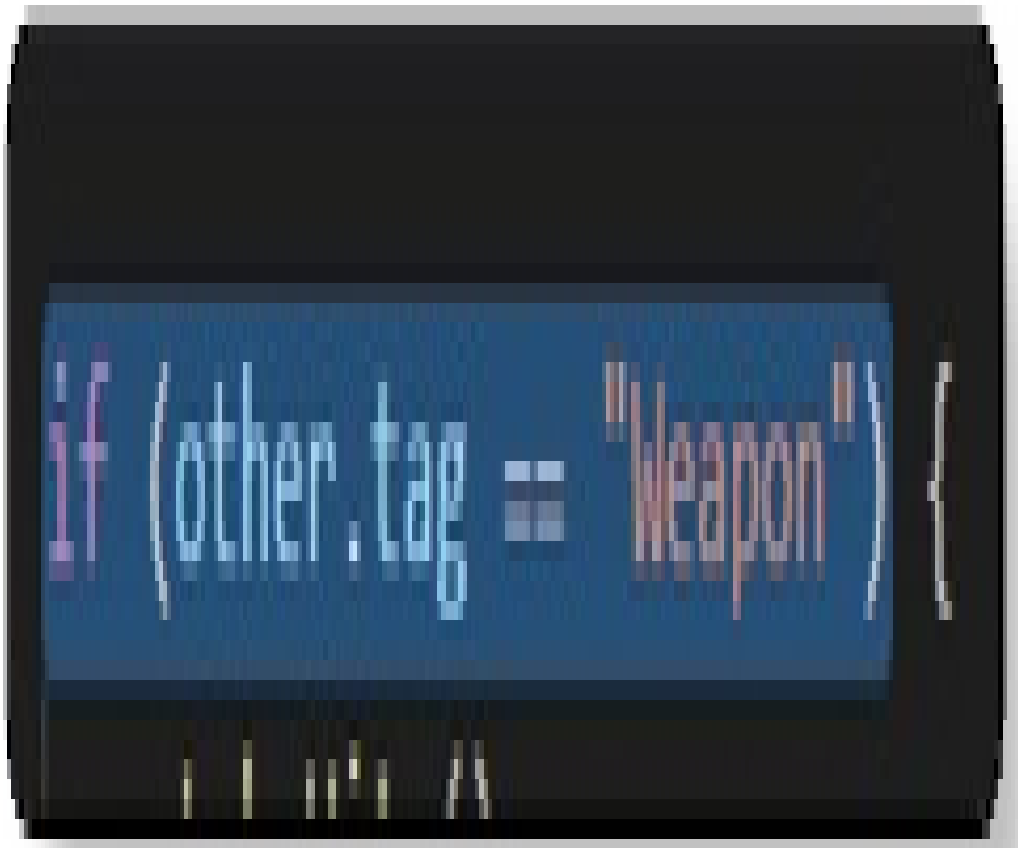
Creación de función

Posteriormente establecemos una estructura condicional que nos compare si el tiempo es mayor a la variable “timeSinceLastHit” y que no se encuentre en estado “GameOver”, entonces:



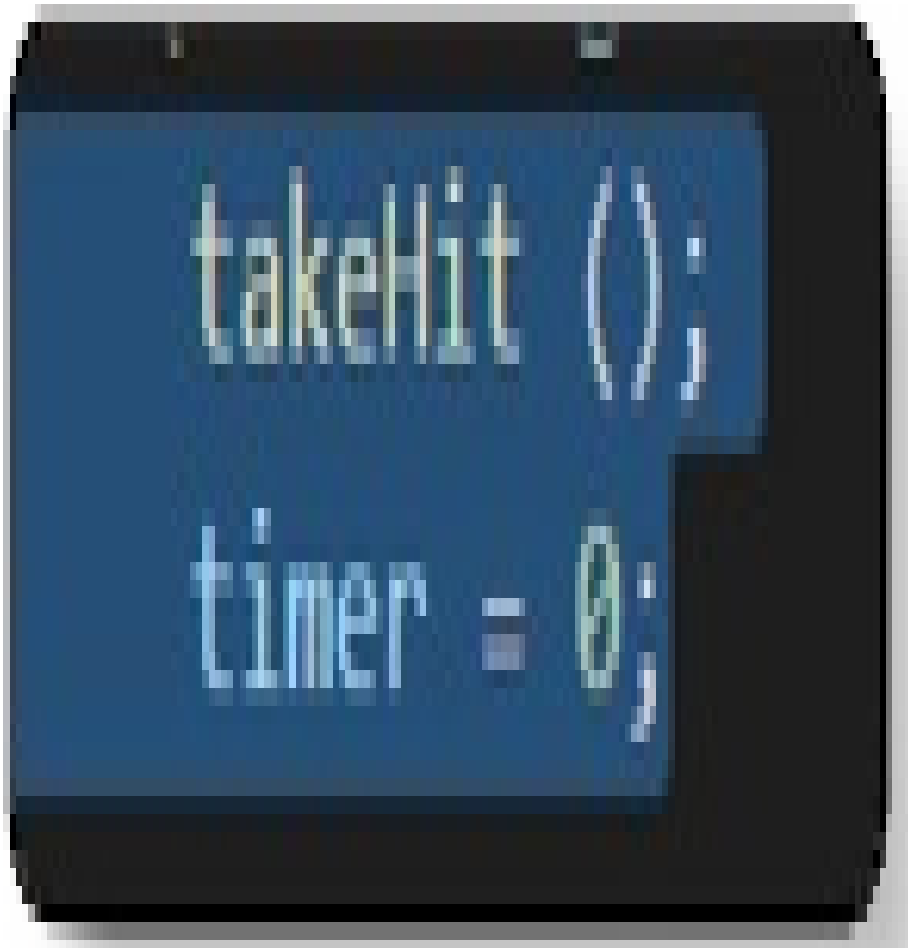
Primera estructura condicional

Establecemos otra estructura condicional donde nos haga una nueva comprobación si el objeto con el que está colindando tiene el Tag “Weapon” entonces:


if (other.tag == "Weapon") {

Segunda estructura condicional

Se correrá la animación “Hurt” y se reiniciará el “timer” a cero.


takeHit ();
timer = 0;

Llamar función takeHit

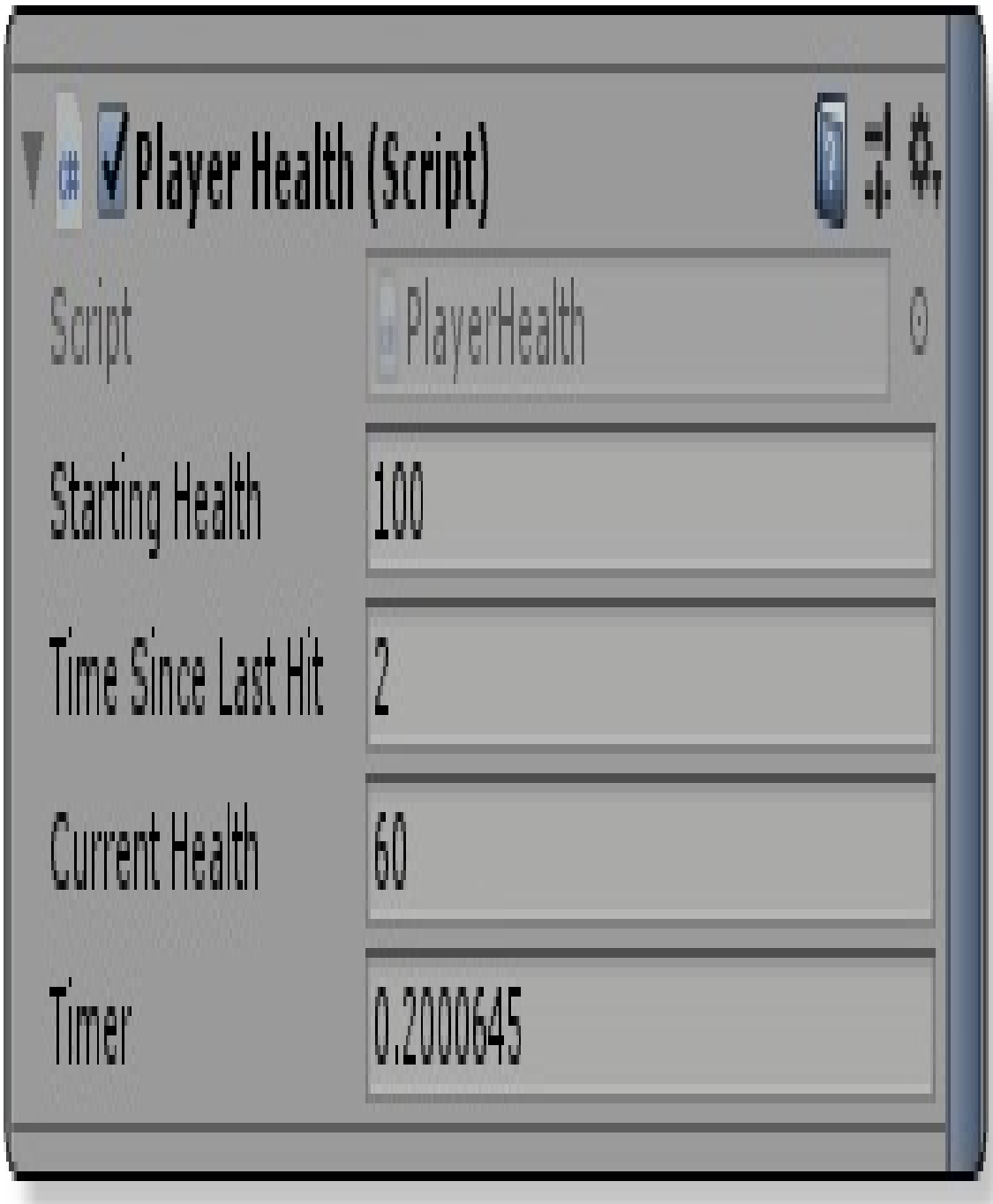
Previsualice el juego para observar que cuando el enemigo daña al “Player”.

Corre la animación “Hurt” del “Player

La variable “currentHealth” baja 10 puntos

El “Timer” se regresa a cero

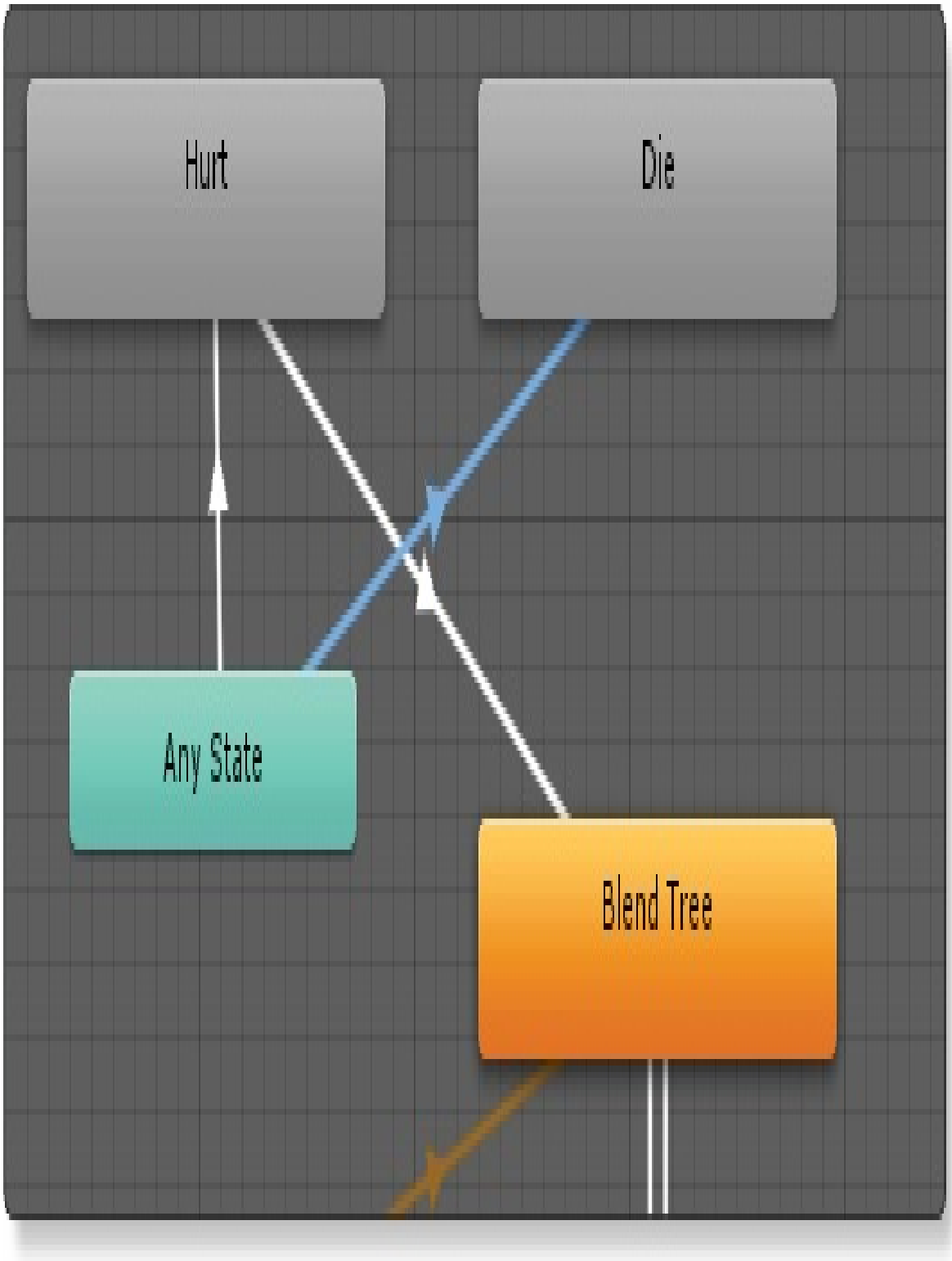
Animación de “Hurt”del Player


Player Health (Script)
Script
PlayerHealth
Starting Health
100
Time Since Last Hit
2
Current Health
60
Timer
0.2000645

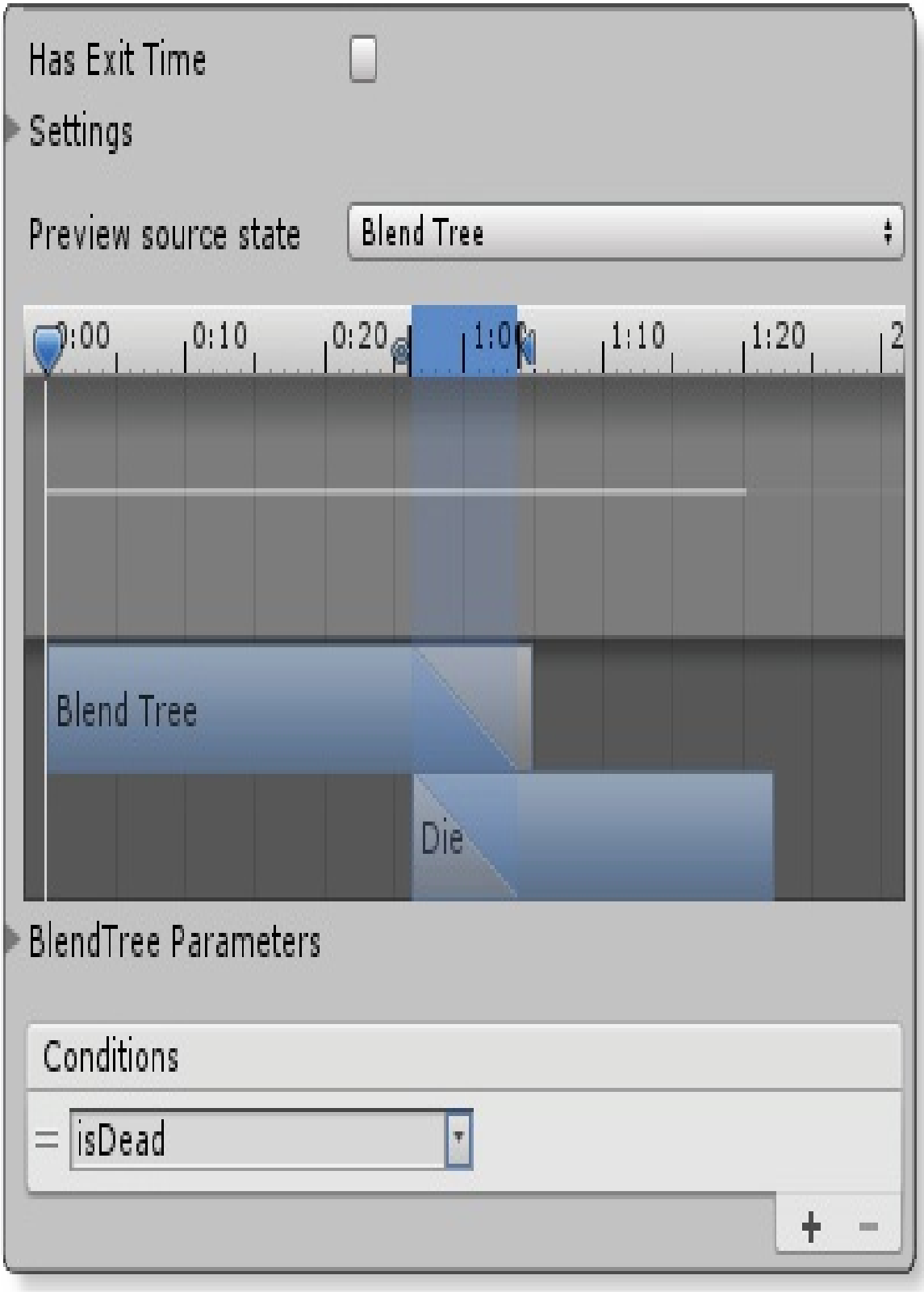
Funcionamiento del script PlayerHealth

Vaya a “Animator” y haga un nuevo parámetro tipo “Trigger” y renómbrelo “isDead”.

Haga la transición que se indica y asigne los parámetros mostrados.


Hurt
Die
Any State
Blend Tree

Transición para la animación de muerte


Has Exit Time
Settings
Preview source state
Blend Tree
0:00
0:10
0:20
1:00
1:10
1:20
Blend Tree
Die
BlendTree Parameters
Conditions
= isDead
+
-

Parámetros de la transición

Necesitamos deshabilitar el componente “CharacterMovement” del “Player” al momento de su muerte, para ello desde el script “PlayerHealth” vamos a mandar llamarlo para deshabilitarlo cuando su “currentHealth” llegue a cero.

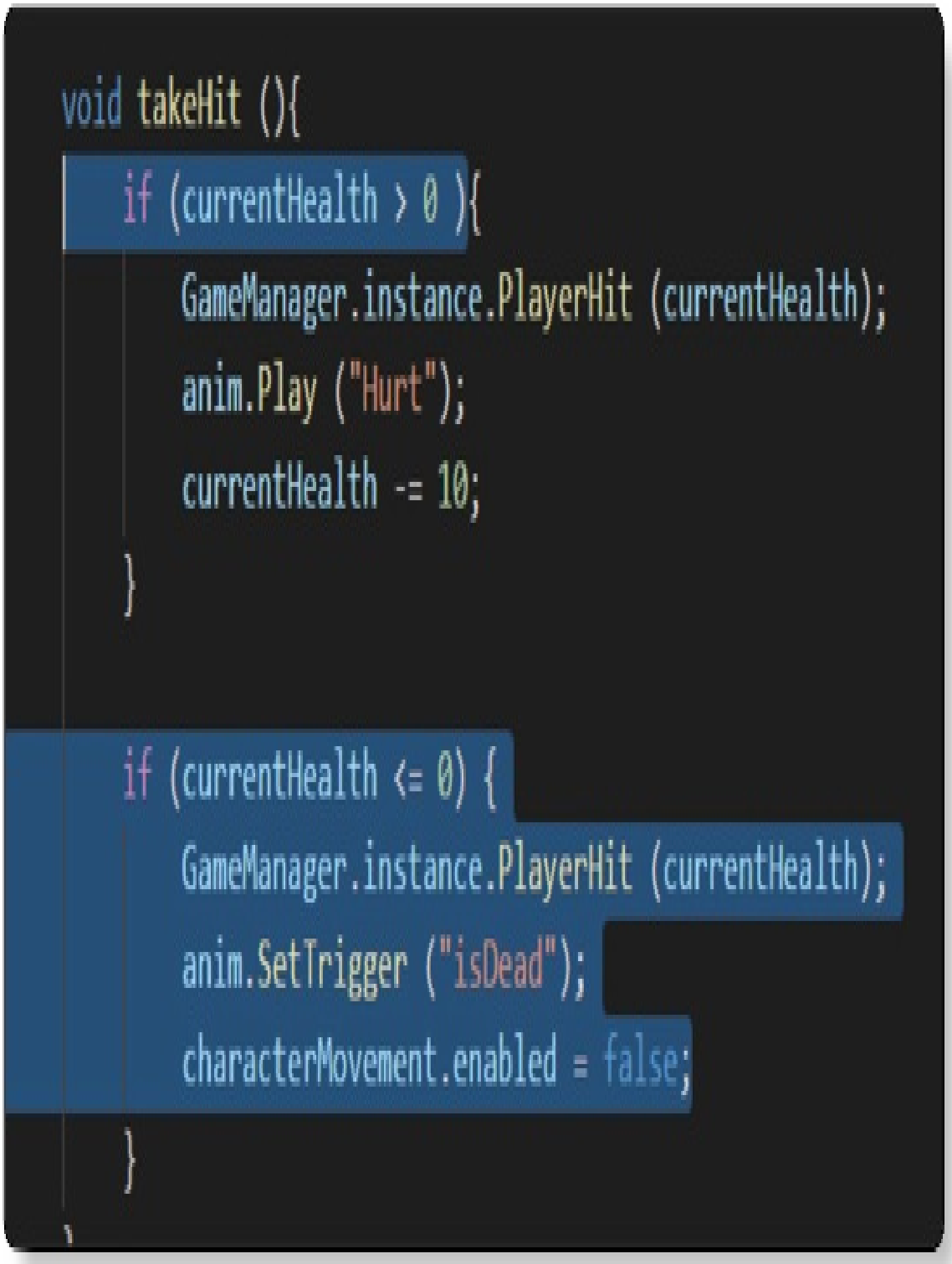
Vaya al script “PlayerHealth” para hacer e inicializar la variable que se muestra.

```
[SerializeField]private float timer = 0f;
private Animator anim;

private  CharacterMovement characterMovement;

// Use this for initialization
void Start () {

    anim = GetComponent<Animator> ();
    currentHealth = startingHealth;
    characterMovement = GetComponent<CharacterMovement> ();
```

Variable para llamar script CharacterMovement

De esta manera mandamos llamar el script “CharacterMovement” que se encuentra en el mismo objeto que el script de “PlayerHealth”.

Haga la siguiente estructura condicional y modifique la anterior.


void takeHit (){
if (currentHealth > 0){
GameManager.instance.PlayerHit (currentHealth);
anim.Play ("Hurt");
currentHealth -= 10;
}
if (currentHealth <= 0) {
GameManager.instance.PlayerHit (currentHealth);
anim.SetTrigger ("isDead");
characterMovement.enabled = false;
}

Estructuras condicionales para sistema de salud

Previsualice el juego para observar que cuando la vida del “Player” llega a cero:

Corre la animación de la muerte

El componente “CharacterMovement” se desactiva

Animación de muerte del Player

PROBLEMA: Cuando muere el “Player” el enemigo sigue avanzando y atacando.

SOLUCIÓN: Vaya al script “EnemyMovement” y escriba el siguiente código en la función “Update”.

```
void Update () {

    if(Vector3.Distance (player.position, this.trar
        nav.SetDestination  (player.position);
        anim.SetBool ("isWalking", true);
        anim.SetBool ("isIdle", false);
    }

    if (GameManager.instance.GameOver){
        print ("GameOver");
        anim.SetBool("isWalking", false);
        anim.SetBool("isIdle", true);
    }
```

Solución de problemática de animación

Previsualice el juego para observar que cuando el “Player” muere, el enemigo vuelve a su animación “Idle”.

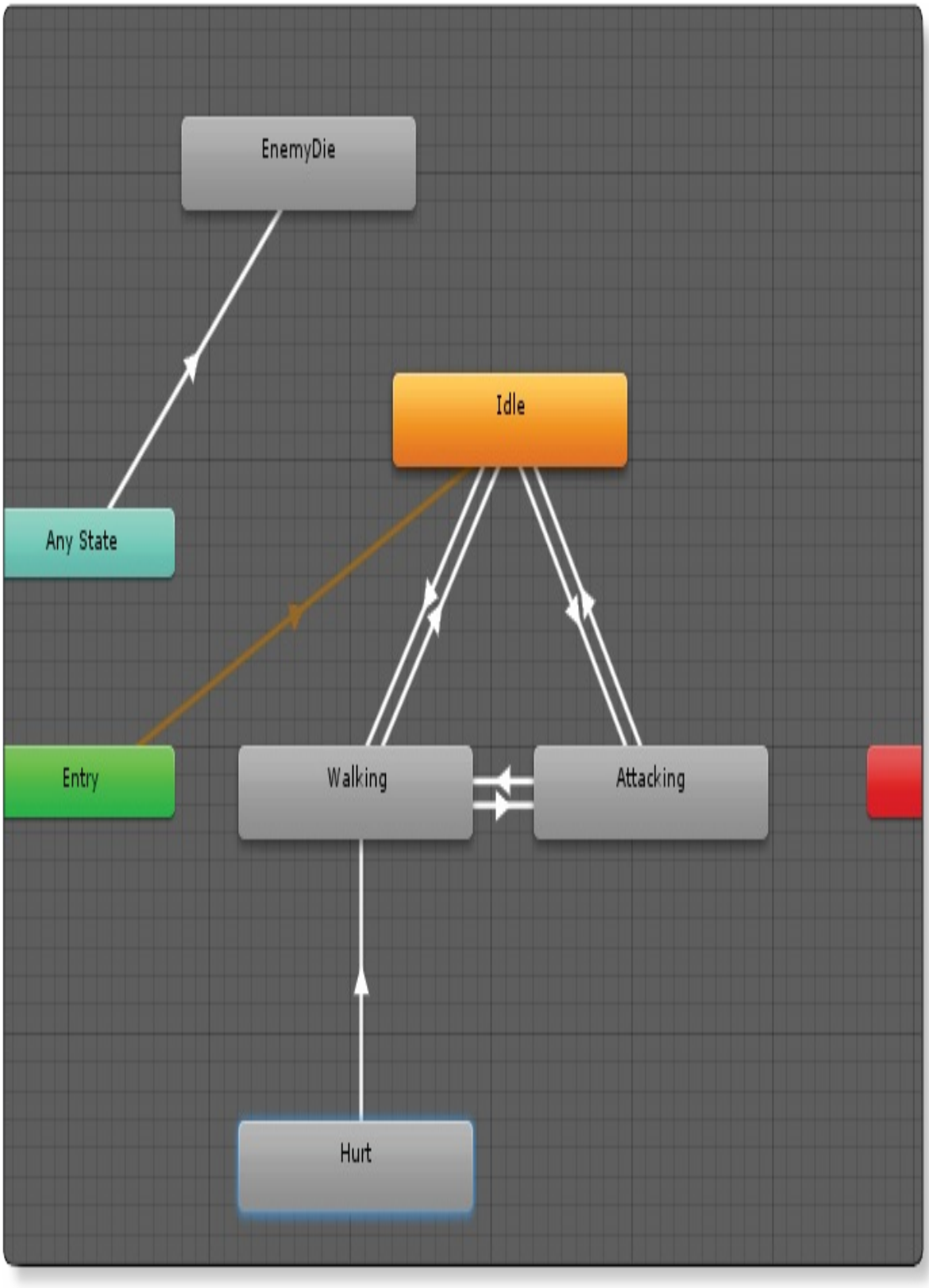
Enemigo01 en animación Idle

PROGRAMACIÓN DEL DAÑO AL ENEMIGO

En este apartado programaremos un sistema de vida y de daño a nuestro enemigo, a su vez, también programaremos sus animaciones de muerte, de desaparición y la deshabilitación de sus componentes al morir.

Seleccione “Enemy01”, haga un nuevo parámetro tipo “Trigger” y nómbrelo “EnemyDie”.

Coloque las animaciones de “Hurt” y de “EnemyDie” n con sus transiciones correspondientes.


EnemyDie
Idle
Any State
Entry
Walking
Attacking
Hurt

Animaciones del enemigo

Asigne el parámetro “EnemyDie” a la condición de la transición de “EnemyDie”.

Haga un nuevo “Tag” y nómbrelo “PlayerWeapon”.

Seleccione el Prefab del proyectil del “Player” (Knife_PFB) y asigne el Tag “PlayerWeapon”.

Haga un nuevo C# Script, nómbrelo “Enemy01Health” y asígnelo a “Enemy01”.

Abra el script creado y escriba el siguiente código para declarar las variables que se muestran.

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class Enemy01Health : MonoBehaviour {

    [SerializeField] private int startingHealth = 20;
    [SerializeField] private float timeSinceLastHit = 0.5f;
    [SerializeField] private float dissapearSpeed = 2f;
    [SerializeField] private int currentHealth;


    private float timer = 0f;
    private Animator anim;
    private NavMeshAgent nav;
    private bool isAlive;
    private Rigidbody rigidbody;
    private CapsuleCollider capsuleCollider;
    private bool dissapearEnemy = false;

    public bool IsAlive {
        get {return isAlive; }
    }
```

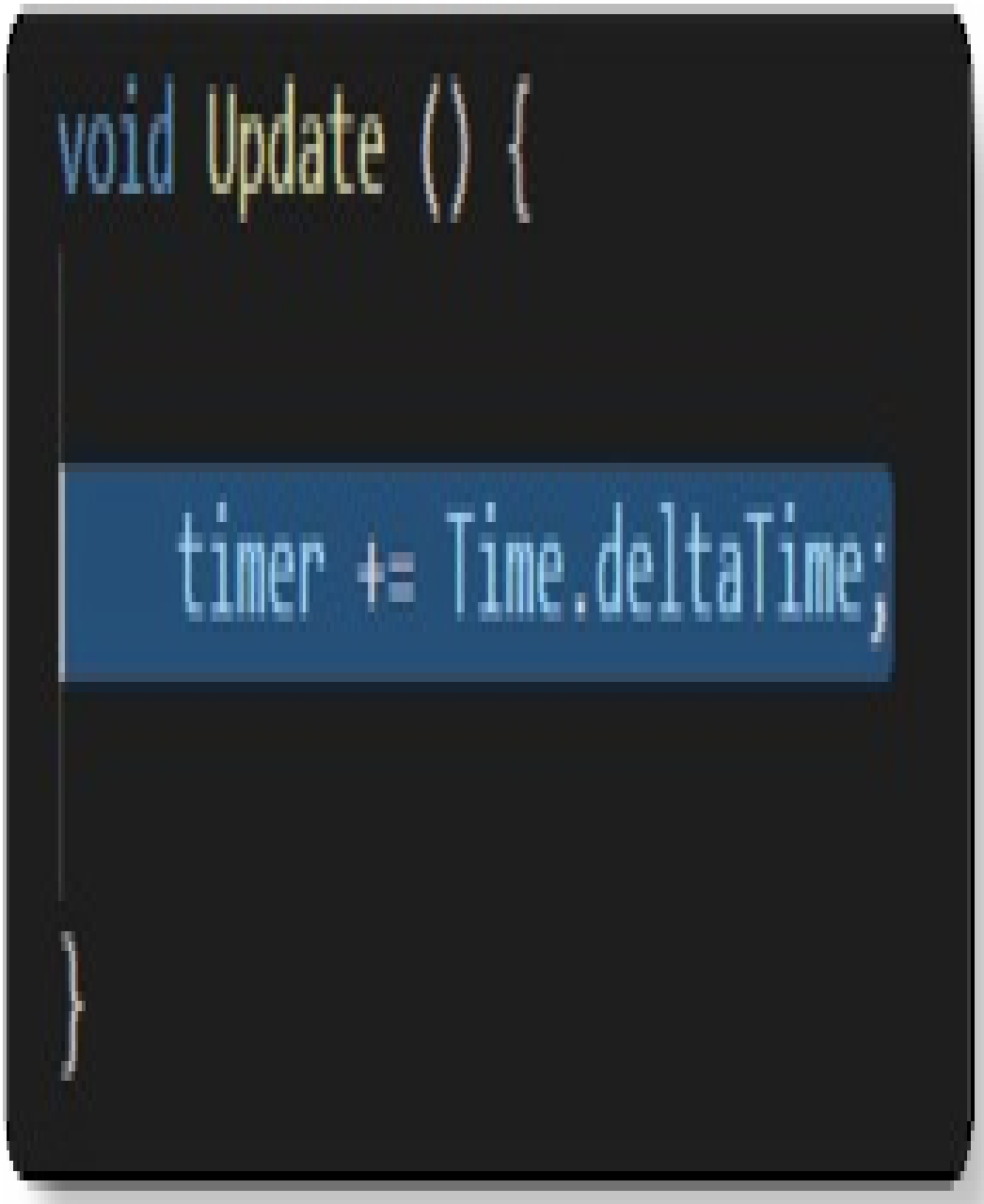
Variables para Enemy01Health

Inicialice las variables que se muestran.

```
void Start () {

    rigidbody = GetComponent <Rigidbody> ();
    capsuleCollider = GetComponent <CapsuleCollider> ();
    nav = GetComponent <NavMeshAgent> ();
    anim = GetComponent <Animator> ();
    isAlive = true;
    currentHealth = startingHealth;
}
```

Inicialización de variables para Enemy01Health

Inicialice “timer” en “Update”.


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

Inicialización del “timer”

Escriba el código con las funciones

```
void Update () {

    timer += Time.deltaTime;

}

void OnTriggerEnter (Collider other) {

    if (timer >= timeSinceLastHit && !GameManager.instance.GameOver) {
        if (other.tag == "PlayerWeapon") {
            takeHit ();
            timer = 0f;
        }
    }
}

void takeHit () {

    if (currentHealth > 0) {
        anim.Play ("Hurt");
        currentHealth -= 10;
    }

    if (currentHealth <= 0) {
        isAlive = false;
        KillEnemy ();
    }
}

void KillEnemy () {

    capsuleCollider.enabled = false;
    nav.enabled = false;
    anim.SetTrigger ("EnemyDie");
    rigidbody.isKinematic = true;
}
```

Funciones para Enemy01Health

En la función “KillEnemy” estamos deshabilitando su CapsuleCollider, el componente “navMeshAgent”, corriendo su animación “EnemyDie” y su rigidBody lo convertimos en “Kinematic”.

Previsualice el juego para observar que con el primer proyectil corre la animación de “Hurt” del enemigo y con el segundo proyectil corre la animación de “EnemyDie” del mismo.

Animación de EnemyDie

A continuación, queremos esperar un pequeño intervalo de tiempo, luego que el personaje se comience a hundir y paso seguido que se destruya.

Escriba la siguiente función “IEnumerator”:

```
        rigidbody.isKinematic = true;
    }


    IEnumerator removeEnemy() {

        yield return new WaitForSeconds (2f);
        dissapearEnemy = true;
        yield return new WaitForSeconds (1f);
        Destroy (gameObject);
    }
}
```

Función IEnumerator

Mande llamar la función creada como con “StartCoroutine” en la función “KillEnemy”.

```
void KillEnemy () {

    capsuleCollider.enabled = false;
    nav.enabled = false;
    anim.SetTrigger ("EnemyDie");
    rigidbody.isKinematic = true;

    StartCoroutine (removeEnemy ());
}
```

Mandar llamar el IEnumerator

Escriba el siguiente código para hundir al enemigo en la función “Update”.

```
void Update () {

    timer += Time.deltaTime;

    if (dissapearEnemy) {
        transform.Translate (-Vector3.up * dissapearSpeed * Time.deltaTime);
    }
```

Código para hundir al personaje

Queremos que el enemigo se hunda hacia abajo pero no tenemos ningún comando “down”, así que lo invocamos negativamente en el vector “up”.

Previsualice el juego para observar que cuando muere el enemigo:

Corre su animación de muerte

Se queda unos instantes estático

Se comienza a hundir

Al momento de estarse hundiendo desaparece

Enemigo hundiéndose

PROBLEMA: Cuando desaparece el enemigo nos aparece en la consola el siguiente error referente al “NavMeshAgent”.

Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
Player in range:False
UnityEngine.MonoBehaviour:print(Object)
"SetDestination" can only be called on an active agent that has been placed on a NavMesh.
UnityEngine.AI.NavMeshAgent:SetDestination(Vector3)

Error por NavMeshAgent

SOLUCIÓN: Vaya al script “Enemy01Movement”, haga e inicialice la variable que se muestra y modifique el código en la función “Update”.

```
    [SerializeField] private Transform player;
    private NavMeshAgent nav;
    private Animator anim;
    private Enemy01Health enemy01Health;

// Use this for initialization
void Start () {

    anim = GetComponent<Animator>();
    nav = GetComponent<NavMeshAgent>();
    enemy01Health = GetComponent<Enemy01Health> ();

}

// Update is called once per frame
void Update () {

    if(Vector3.Distance (player.position, this.transform.position) < 12) {
        if (!GameManager.instance.GameOver && enemy01Health.IsAlive) {
        nav.SetDestination  (player.position);
        anim.SetBool ("isWalking", true);
        anim.SetBool ("isIdle", false);
        }
    } else if  (GameManager.instance.GameOver || !enemy01Health.IsAlive){
        anim.SetBool("isWalking", false);
        anim.SetBool("isIdle", true);
        nav.enabled = false;
    }
}
```

Solución de error de NavMeshAgent

Para solucionar el error hemos encapsulado algunas condiciones en función a la modalidad del juego y desactivado su “NavMeshAgent”.

Previsualice el juego para observar que al morir el enemigo ya no aparece el error.

PROBLEMA: Al morir el enemigo, si el “Player” se acerca a él, el enemigo vuelve a atacar.

SOLUCIÓN: Vaya al script “Enemy01Attack”, escriba e inicialice la siguiente variable y en la función “Update” añada el código que se muestra.

```
private bool playerInRange;
private BoxCollider weaponCollider;
private Enemy01Health enemy01Health;

// Use this for initialization
void Start () {

    enemy01Health = GetComponent<Enemy01Health>();
    anim = GetComponent <Animator> ();
    player = GameManager.instance.Player;
    weaponCollider = GetComponentInChildren<BoxCollider> ();
    StartCoroutine (attack ());

}

// Update is called once per frame
void Update () {
    if (Vector3.Distance (transform.position, player.transform.position) < range && enemy01Health.IsAlive) {
        playerInRange = true;
```

Solución para problema de ataque

Mandamos llamar la variable “IsAlive” desde el script “Enemy01Health” para establecer condición.

Previsualice el juego para observar que el enemigo deja de atacar.

PROBLEMA: Antes de que desaparezca el enemigo, el “Collider” del arma sigue estando activa y dañando al “Player”.

SOLUCIÓN: Vaya al script de “Enemy01Health”, haga la siguiente variable, inicialícela y escriba el código que se muestra en la función “KillEnemy”.

```
private CapsuleCollider capsuleCollider;
private bool dissapearEnemy = false;
private BoxCollider weaponCollider;

public bool IsAlive {
    get {return isAlive; }
}

// Use this for initialization
void Start () {

    weaponCollider = GetComponentInChildren<BoxCollider>();
    rigidbody = GetComponent <Rigidbody> ();
    capsuleCollider = GetComponent <CapsuleCollider> ();
```

Creación e inicialización de variable

```
void KillEnemy () {

    capsuleCollider.enabled = false;
    nav.enabled = false;
    anim.SetTrigger ("EnemyDie");
    rigidbody.isKinematic = true;
    weaponCollider.enabled = false;

    StartCoroutine (removeEnemy ());
}
```

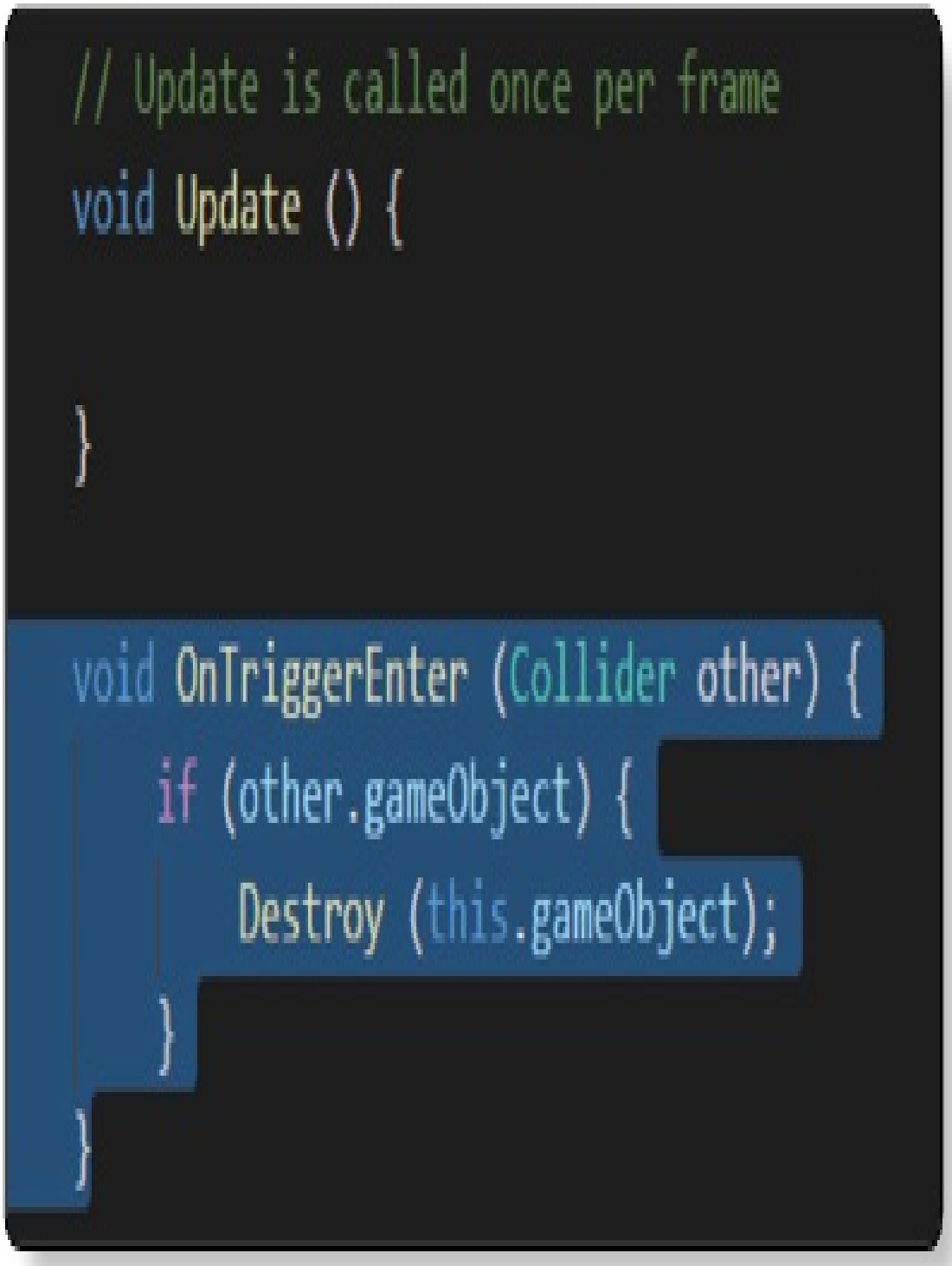
Deshabilitación del “Collider” del arma

De esta manera deshabilitamos el “Collider” del arma.

Previsualice el juego para observar que al morir el enemigo el arma ya no daña al “Player”.

PROBLEMA: El proyectil “Kinfe” del “Player” no se destruye al colisionar con el enemigo (o con cualquier otro objeto).

SOLUCIÓN: Vaya al script de “KnifeDestruction” y escriba la siguiente función.

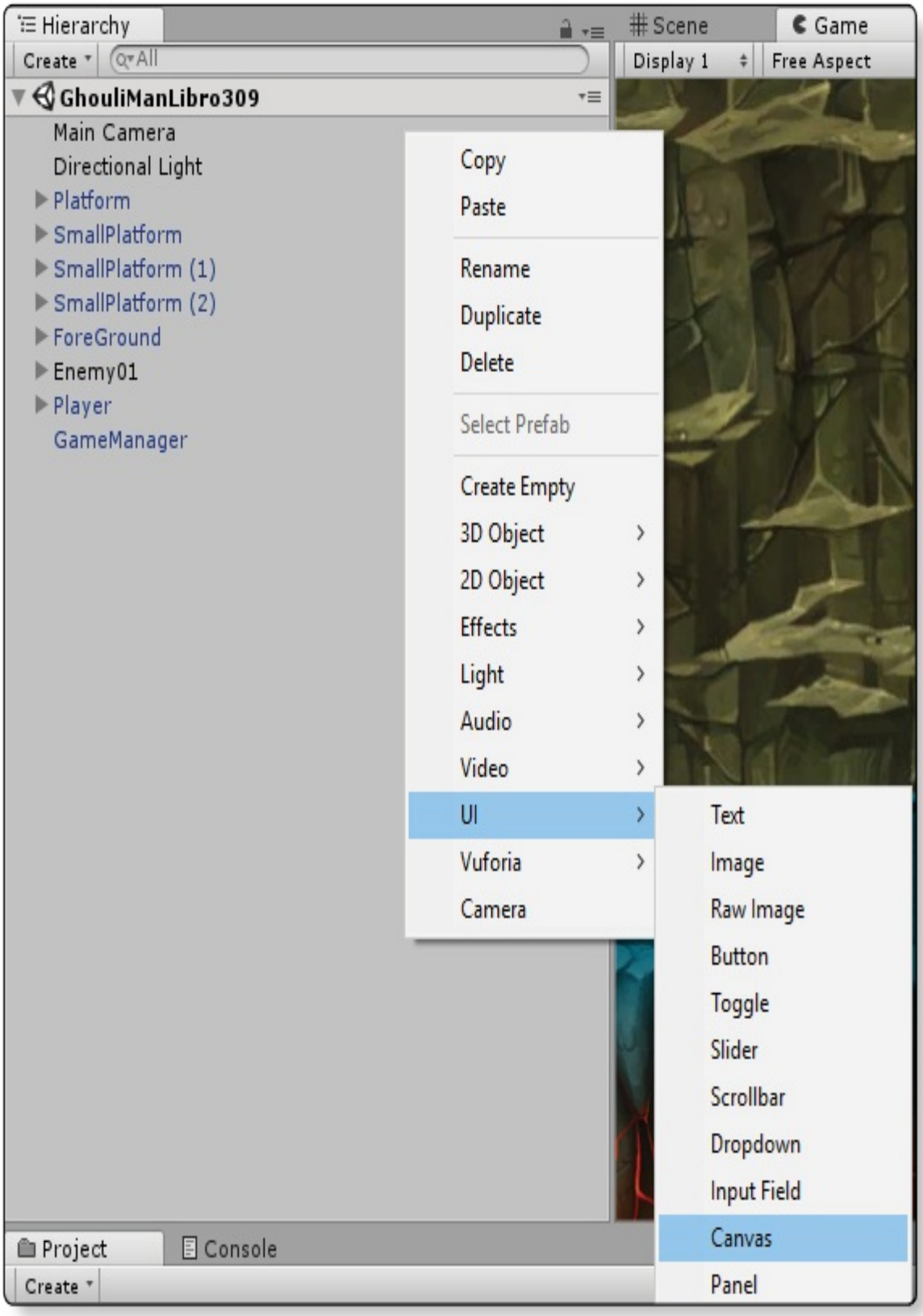

// Update is called once per frame
void Update () {
}
void OnTriggerEnter (Collider other) {
if (other.gameObject) {
Destroy (this.gameObject);
}
}

Solución para destruir proyectiles

CREACIÓN DE UNA BARRA DE ENERGÍA PARA EL PLAYER

En este apartado aprenderemos a crear elemento GUI (Graphic User Interfase) para generar una barra de vida para nuestro “Player” y a su vez la enlazaremos vía scripting con su parámetro de salud para que vaya bajando gráficamente conforme baja su salud.

En la ventana “Hierarchy” haga clic derecho y asigne “UI / Canvas” para crear un nuevo elemento “Canvas”.

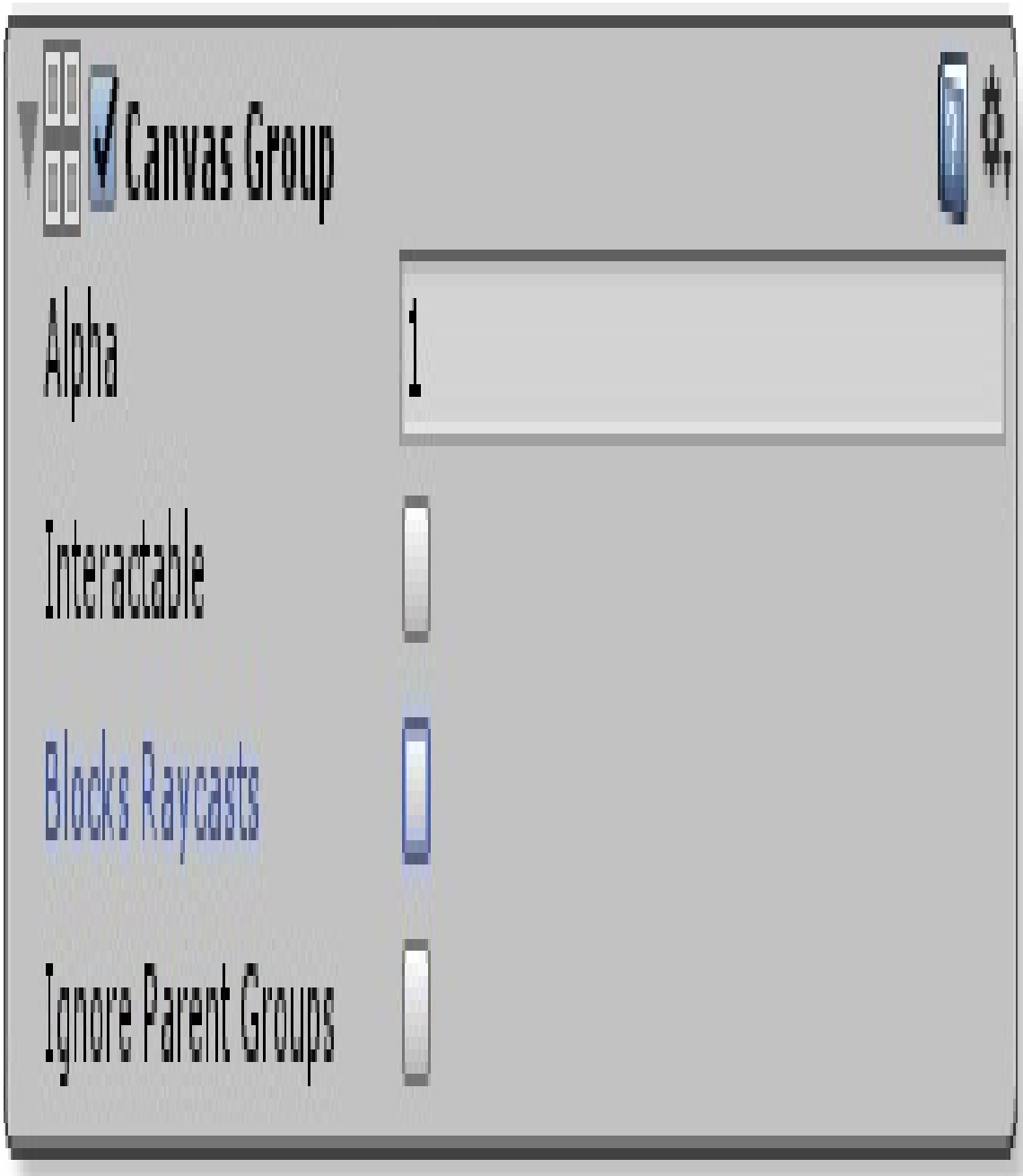

Hierarchy
Scene
Game
Create
All
Display 1
Free Aspect
GhouliManLibro309
Main Camera
Directional Light
Platform
SmallPlatform
SmallPlatform (1)
SmallPlatform (2)
ForeGround
Enemy01
Player
GameManager
Copy
Paste
Rename
Duplicate
Delete
Select Prefab
Create Empty
3D Object
2D Object
Effects
Light
Audio
Video
UI
Vuforia
Camera
Text
Image
Raw Image
Button
Toggle
Slider
Scrollbar
Dropdown
Input Field
Canvas
Panel
Project
Console
Create

Creación de un Canvas

El elemento “Canvas” es como un lienzo que nos abarcará toda la pantalla de nuestro juego.

Renombre el elemento “Canvas” como “GUI”.

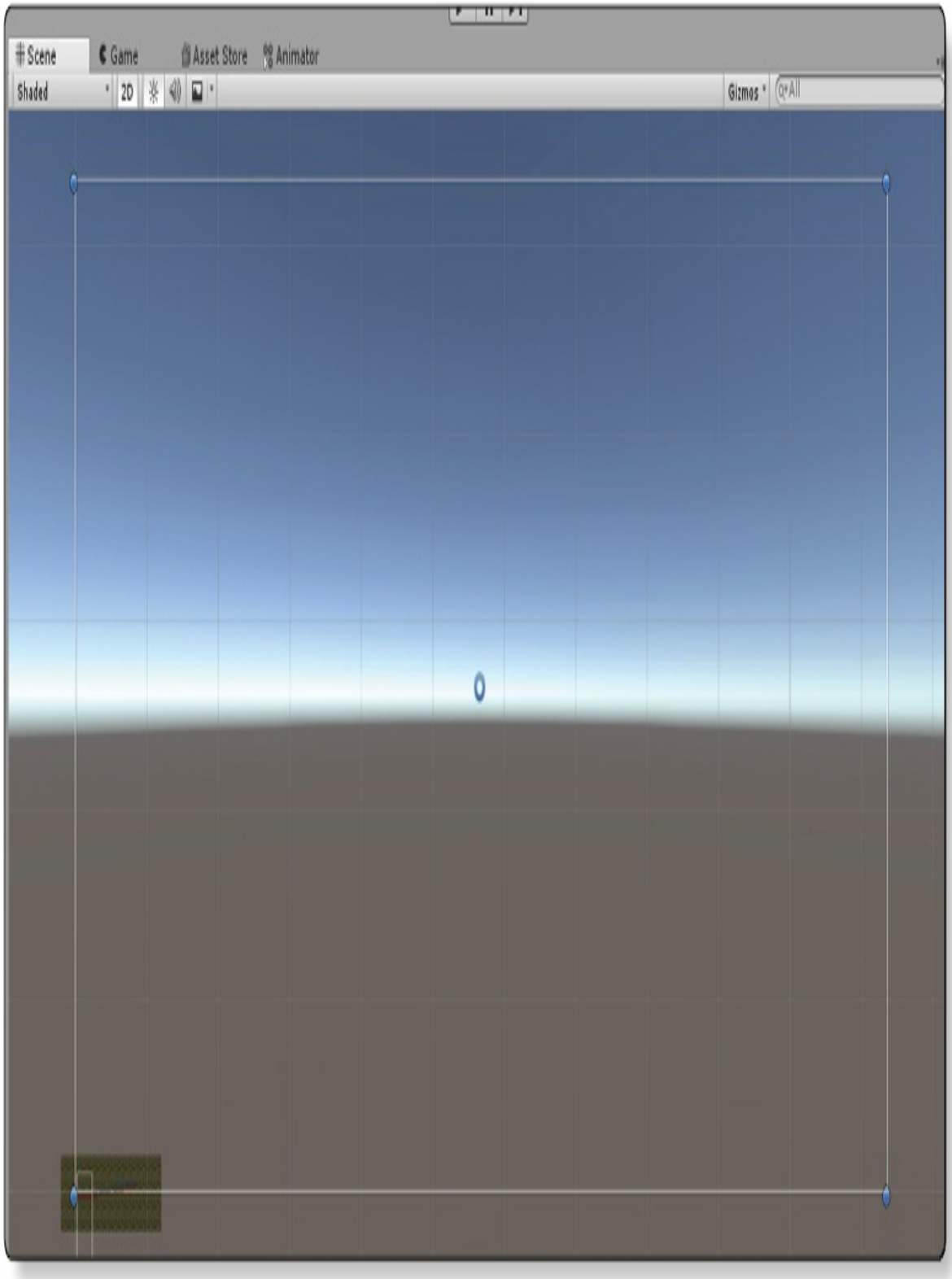
Seleccione el objeto “GUI” y asigne un componente “CanvasGroup” y asigne los valores mostrados.


Canvas Group
Alpha
1
Interactable
Blocks Raycasts
Ignore Parent Groups

Componente Canvas Group

El componente “CanvasGroup” nos permitirá controlar diferentes elementos UI dentro del mismo Canvas.

En la ventana “Scene” haga “Zoom Back” para visualizar el “Canvas” creado.


Scene
Game
Asset Store
Animator
Shaded
2D
Gizmos
Q*All

Visualización del Canvas

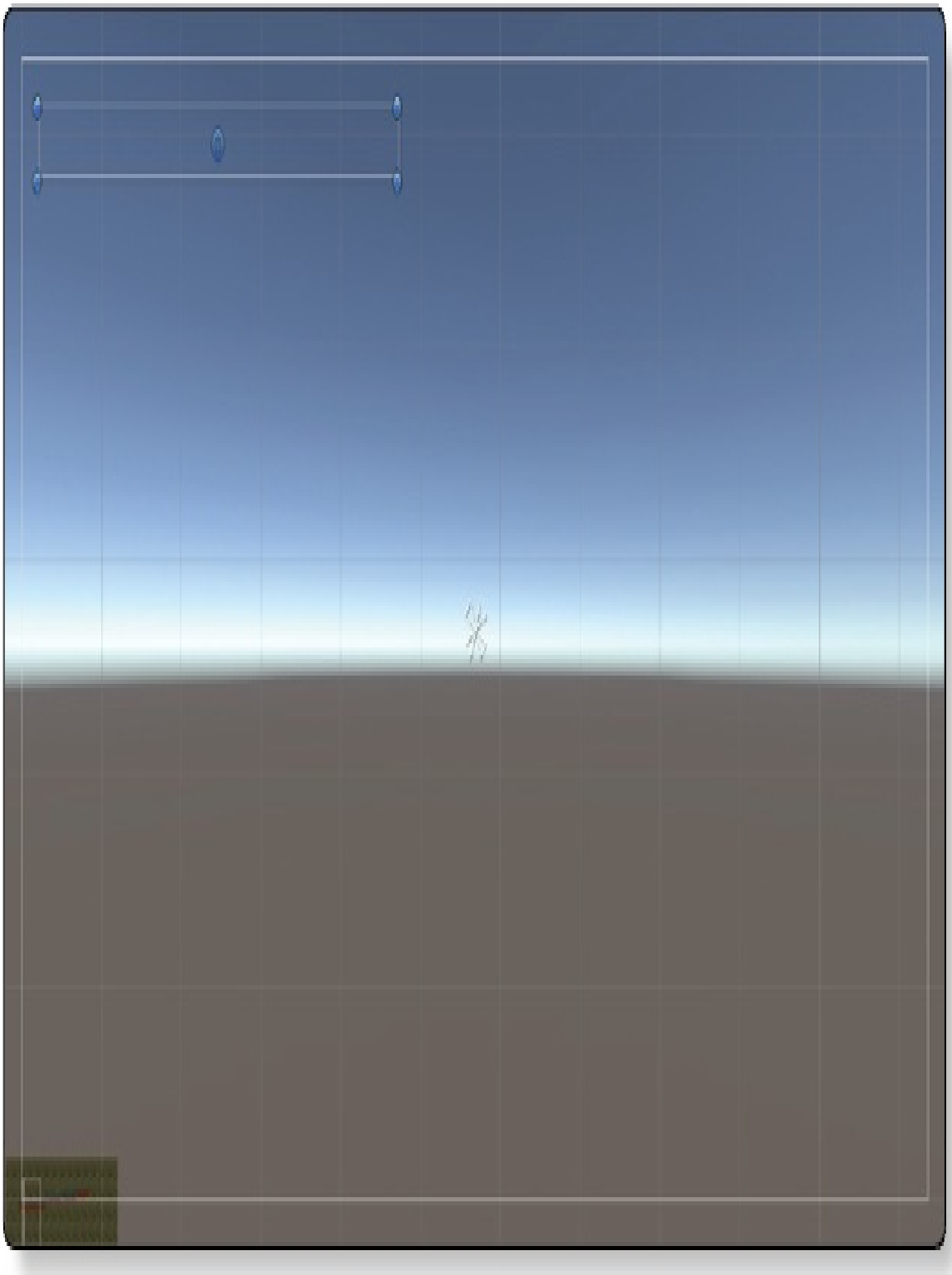
La representación del “Canvas” nos aparece de esta forma, puede asignar el botón 2D para facilitar la visualización de los elementos que constituirán nuestro GUI.

En Hierarchy selección GUI, asigne clic derecho para crear un nuevo “EmptyObject” y renómbrelo “HealthBar”.

GameManager
EventSystem
GUI
HealthBar

Creación de HealthBar

Escale el objeto “HealthBar”.



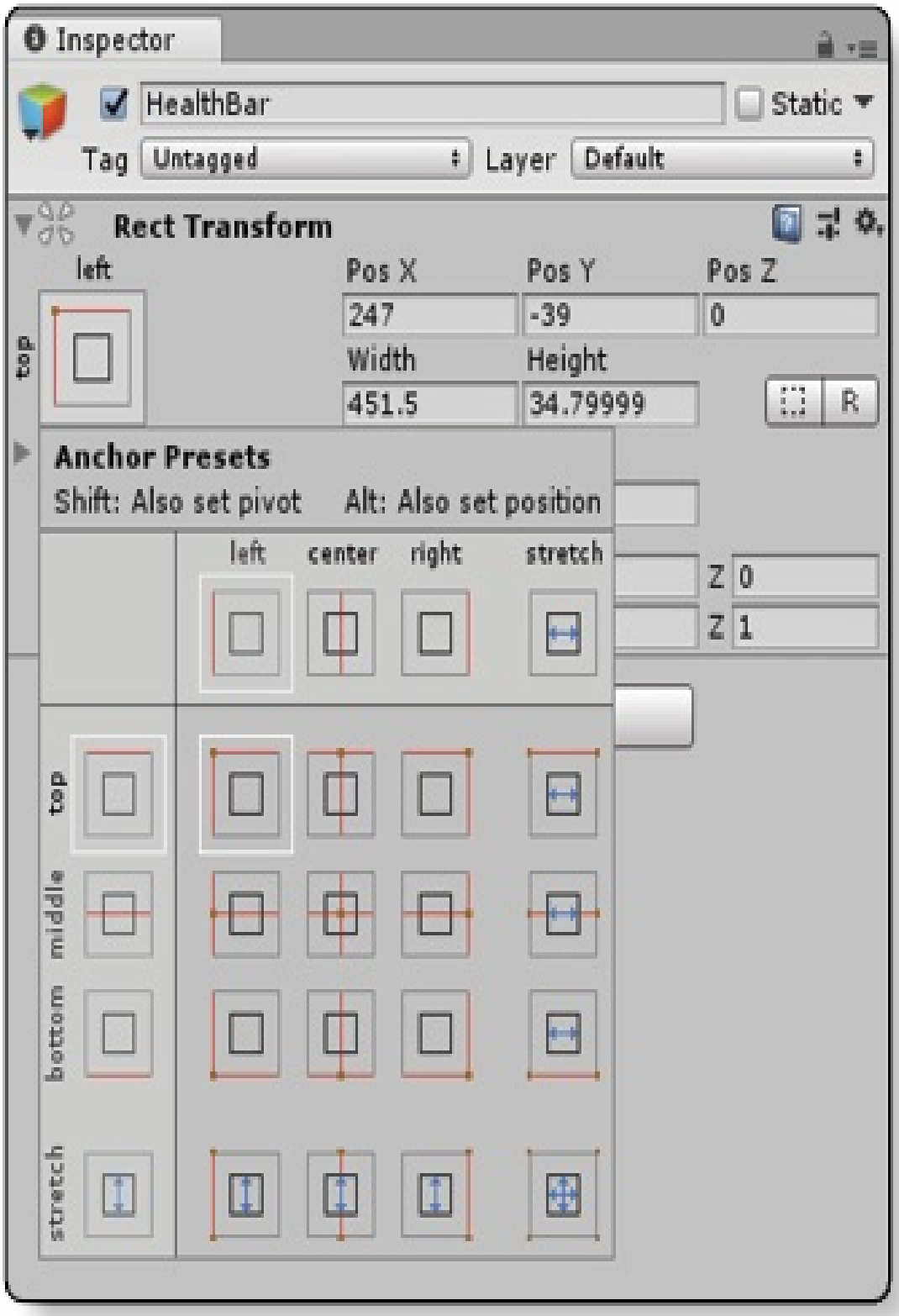
Posicionamiento de HealthBar

Es importante notar el anclaje del elemento, en este punto el anclaje se encuentra al centro y podemos dar cuenta de ello por el icono de cruz que podemos visualizar al centro.

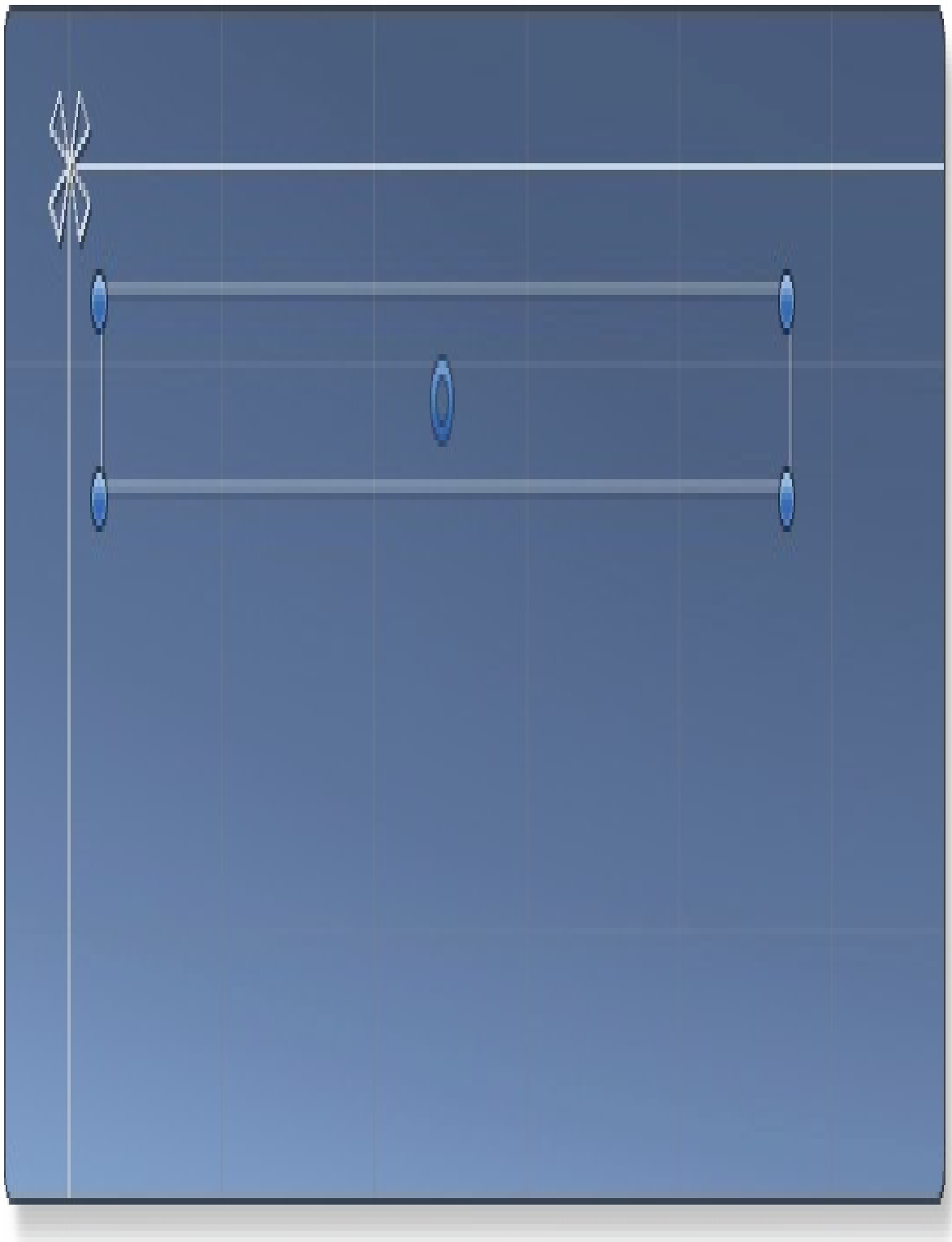
Es poco recomendable que un elemento GUI se encuentre anclado de esta manera ya que en el modo juego puede reacomodarse en función a los diferentes tipos de pantalla donde lo estemos corriendo.

Los más recomendable es anclarlos en las esquinas cercanas a donde van a encontrarse nuestros elementos.

En la ventana de “Inspector” clic en el icono de anclaje y seleccione la opción de la esquina superior izquierda.


Inspector
HealthBar
Static
Tag
Untagged
Layer
Default
Rect Transform
left
top
Pos X
Pos Y
Pos Z
247
-39
0
Width
Height
451.5
34.79999
R
Anchor Presets
Shift: Also set pivot
Alt: Also set position
left
center
right
stretch
top
middle
bottom
stretch
Z 0
Z 1

Anclaje de HealthBar



Visualización de anclaje

De esta manera podemos visualizar que el nuevo anclaje será en la esquina de nuestra pantalla del juego.

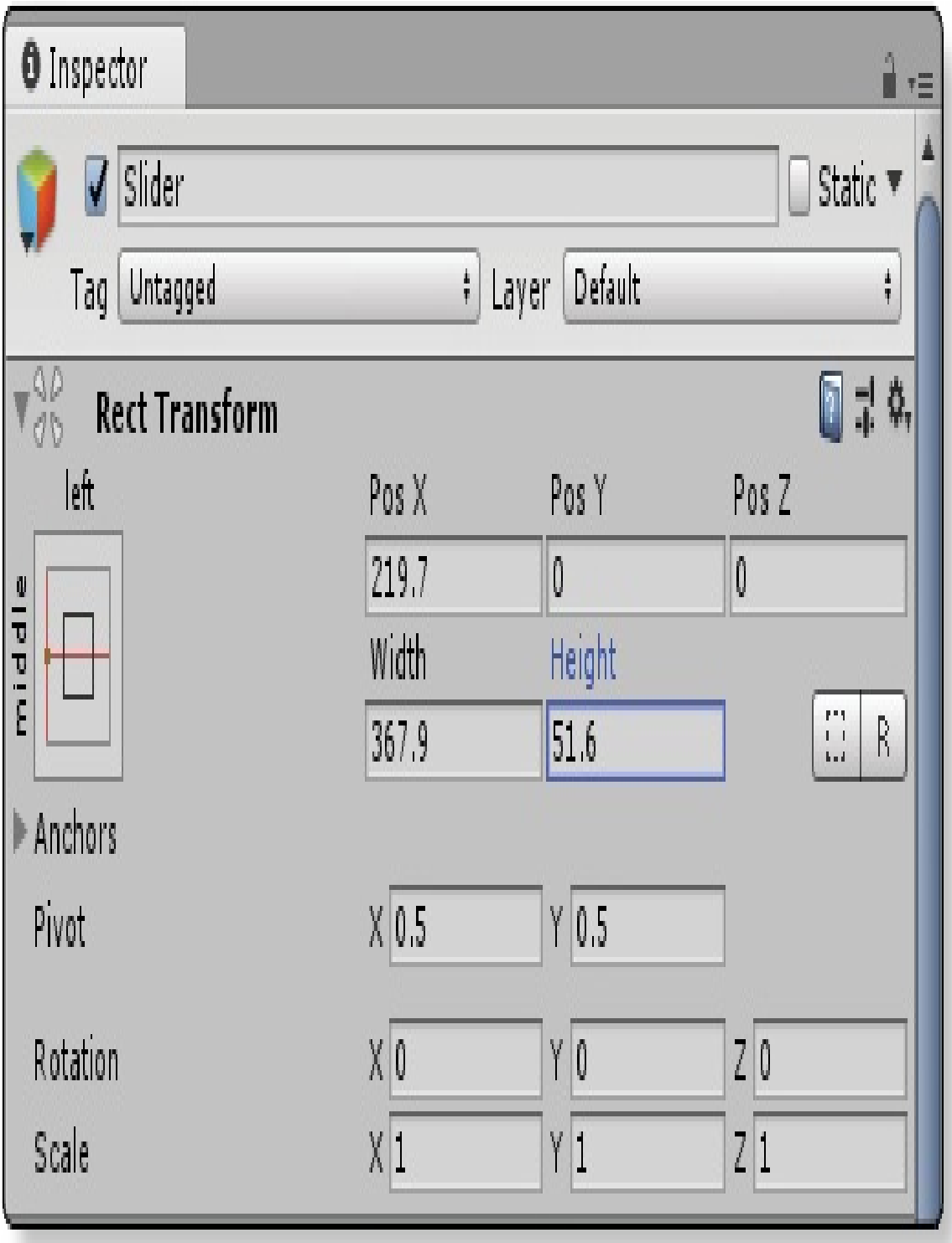
Seleccione “HealthBar”, clic derecho y asigne “UI / Slider”.

Expándalo en “Hierarchy” y elimine el elemento “Handle Size Area” ya que no lo vamos a necesitar.

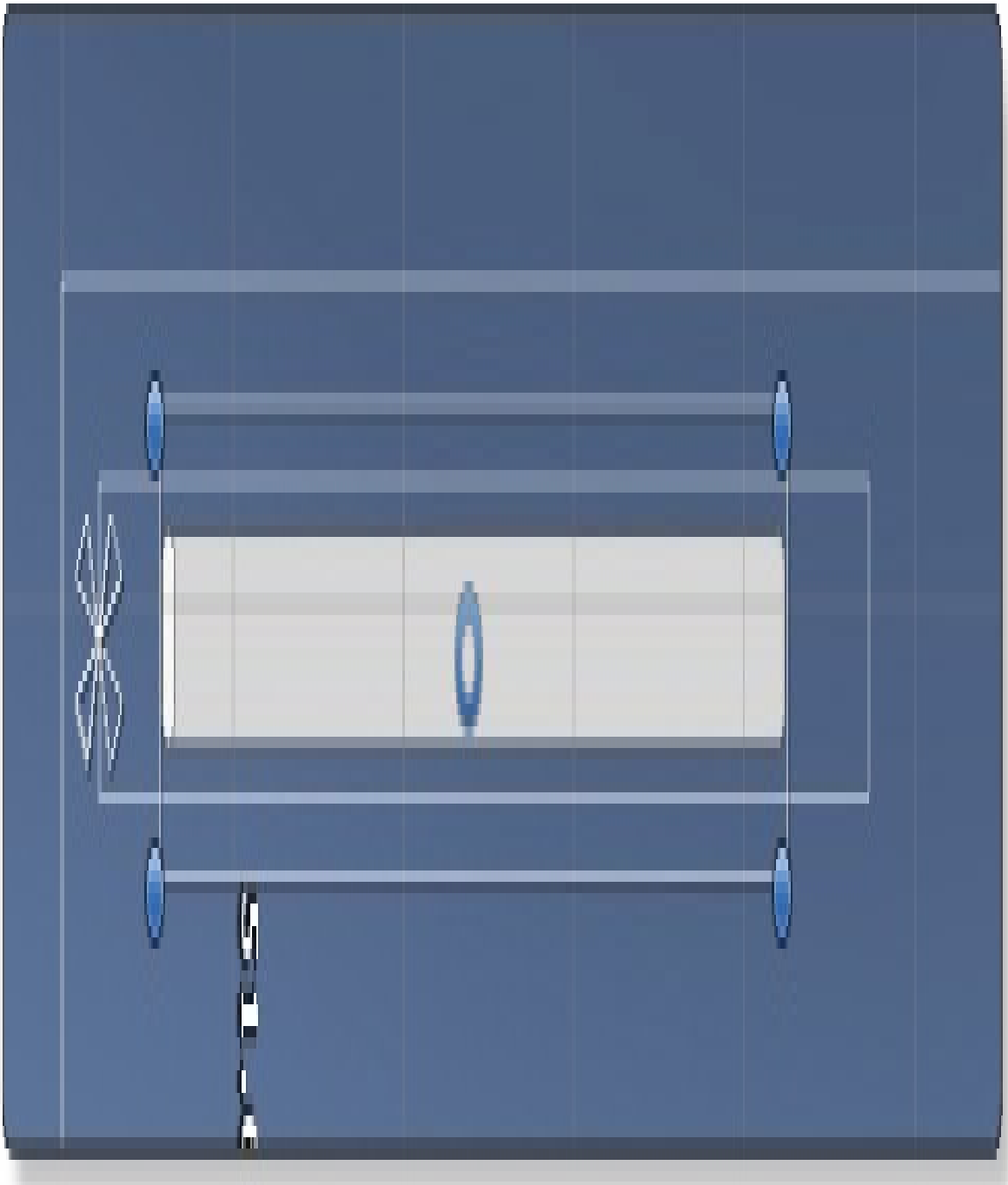
GameManager
EventSystem
GUI
HealthBar
Slider
Background
Fill Area

Expansión de elementos GUI

Seleccione “Slider” y asigne los siguientes valores en “Inspector”.


Inspector
Slider
Static
Tag
Untagged
Layer
Default
Rect Transform
left
middle
Pos X
Pos Y
Pos Z
219.7
0
0
Width
Height
367.9
51.6
R
Anchors
Pivot
X 0.5
Y 0.5
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1

Parámetros para el Slider

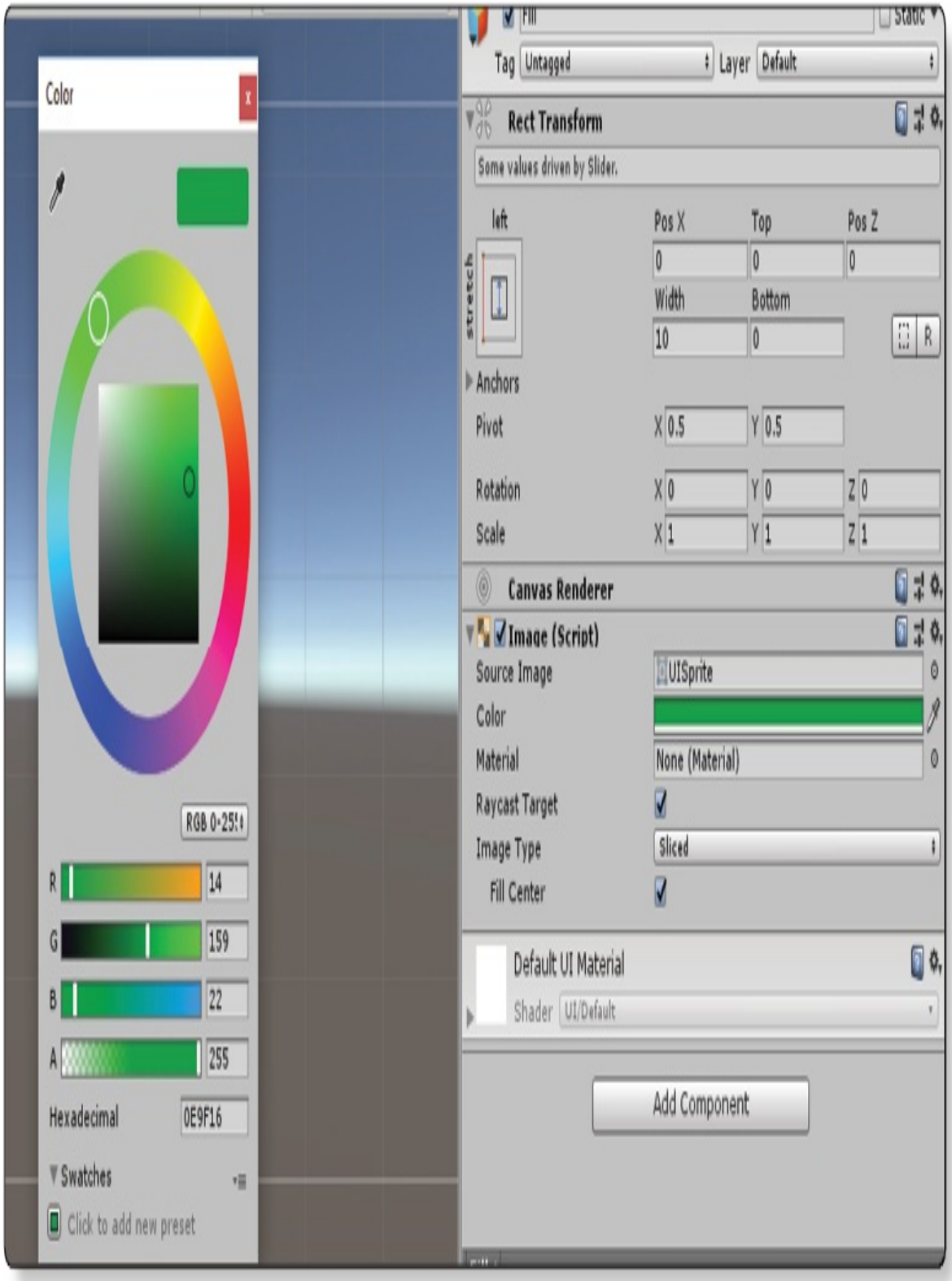


Visualización de Slider

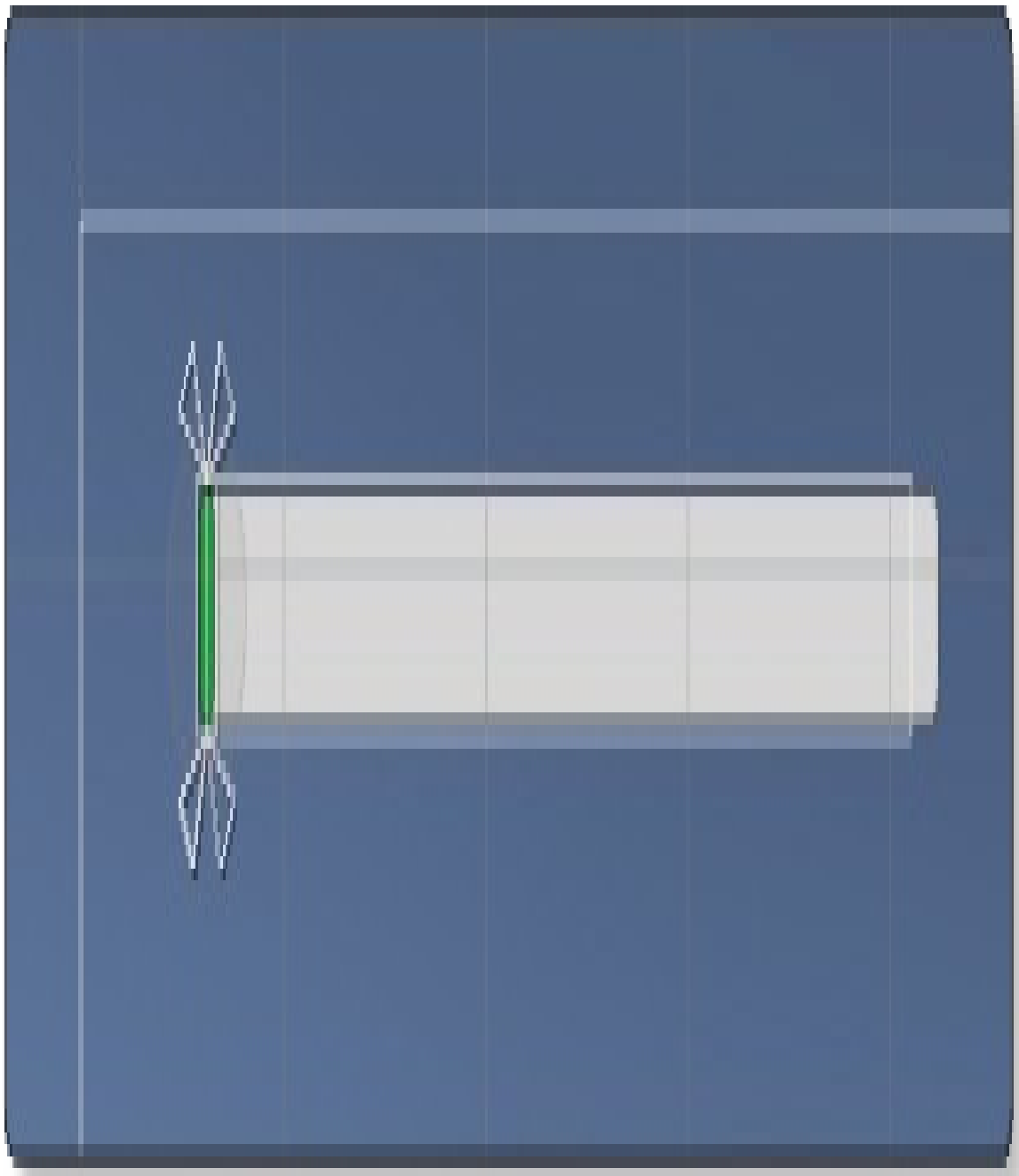
Expanda “Fill Area” en “Hierarchy”, seleccione “Fill” y en “Inspector” cambie el color.

EventSystem
GUI
HealthBar
Slider
Background
Fill Area
Fill

Fill seleccionado


Color
RGB 0-255
R 14
G 159
B 22
A 255
Hexadecimal 0E9F16
Swatches
Click to add new preset
Fill
Static
Tag Untagged
Layer Default
Rect Transform
Some values driven by Slider.
left
stretch
Pos X 0
Top 0
Pos Z 0
Width 10
Bottom 0
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Canvas Renderer
Image (Script)
Source Image UISprite
Color
Material None (Material)
Raycast Target
Image Type Sliced
Fill Center
Default UI Material
Shader UI/Default
Add Component

Parámetros de Fill

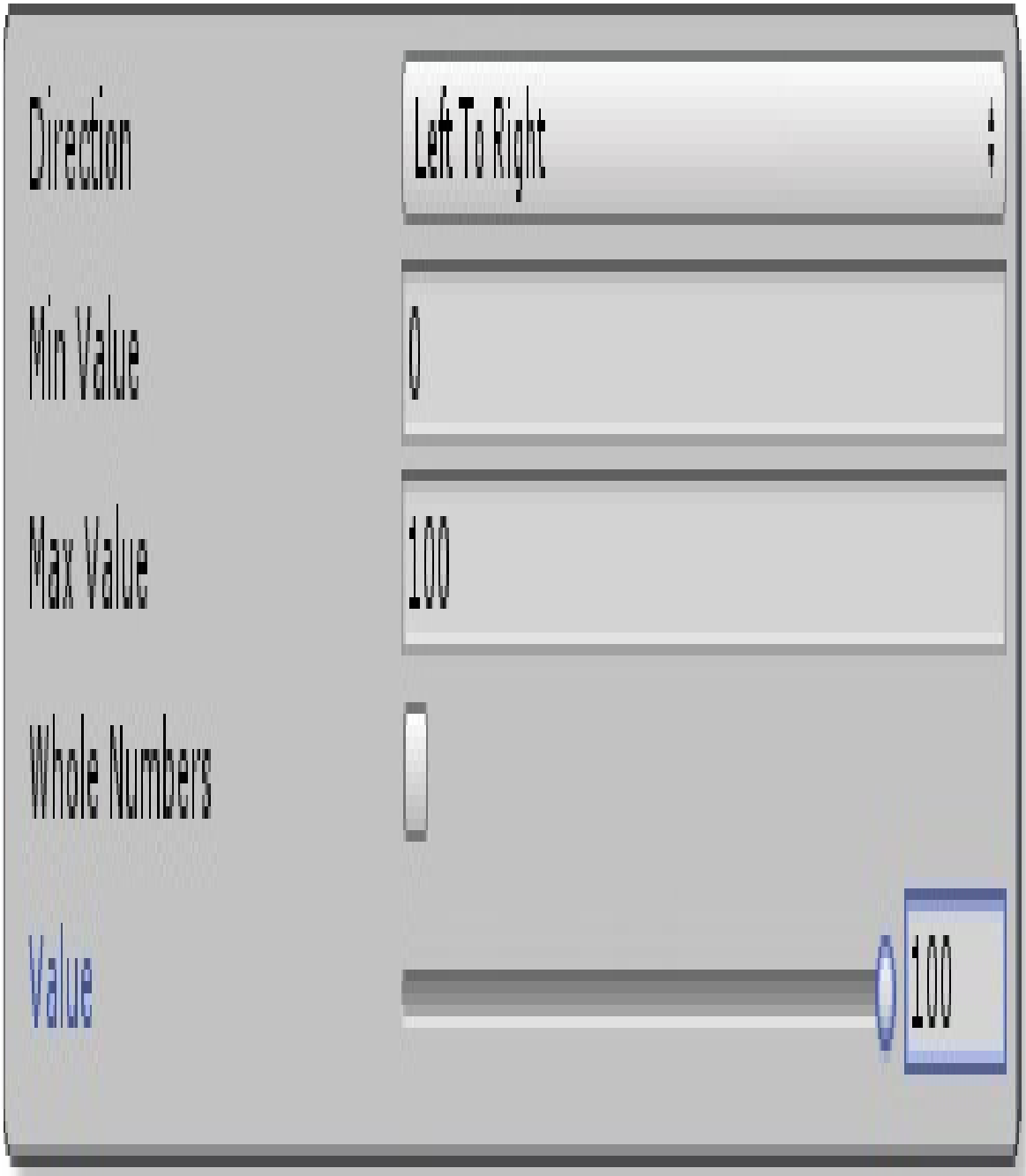


Visualización de Fill

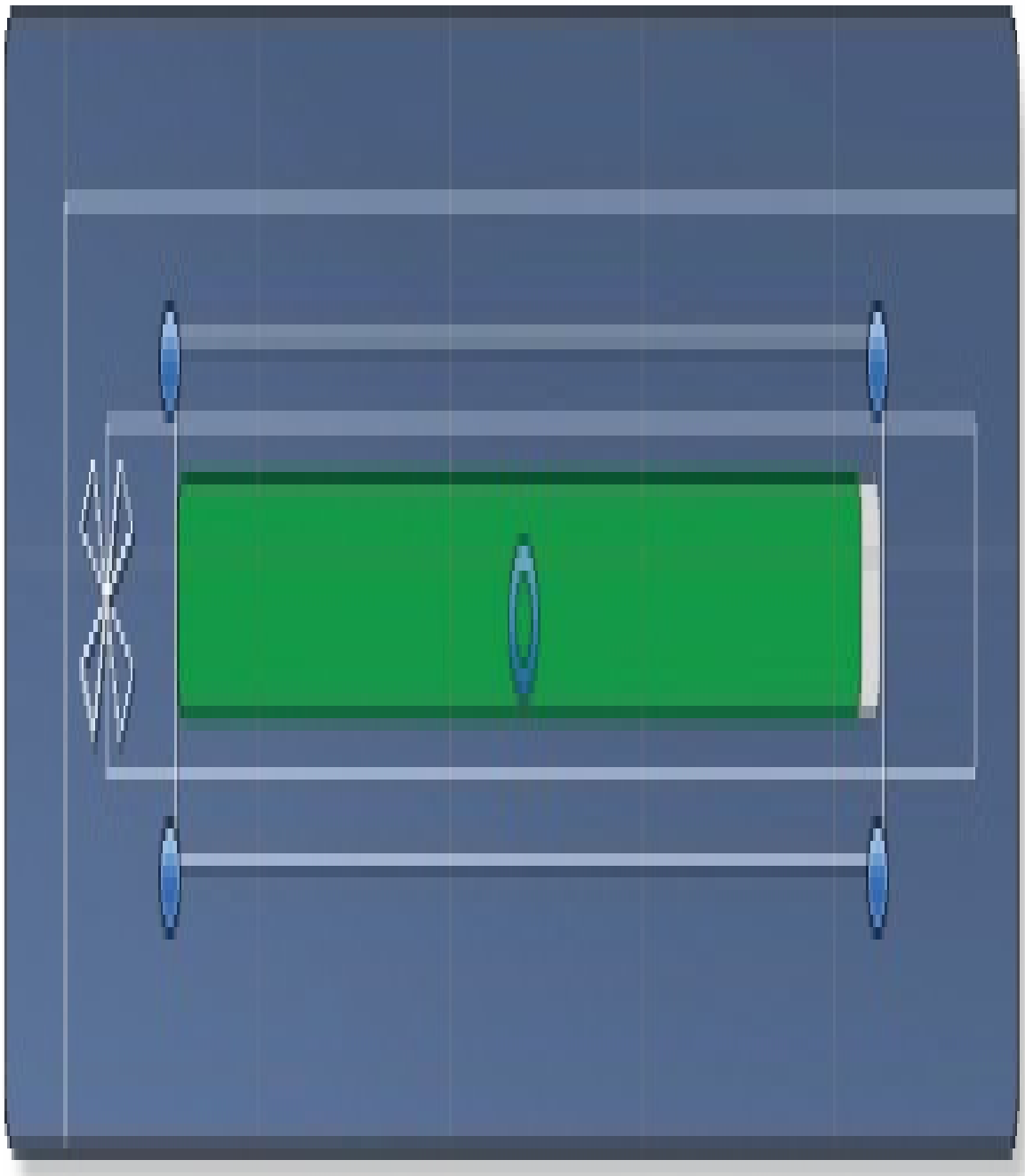
En “Hierarchy” seleccione “Slider” y asigne los siguientes valores.

GUI
HealthBar
Slider
Background
Fill Area
Fill

Slider seleccionado


Direction
Left To Right
Min Value
0
Max Value
100
Whole Numbers
Value
100

Parámetro Value de Slider

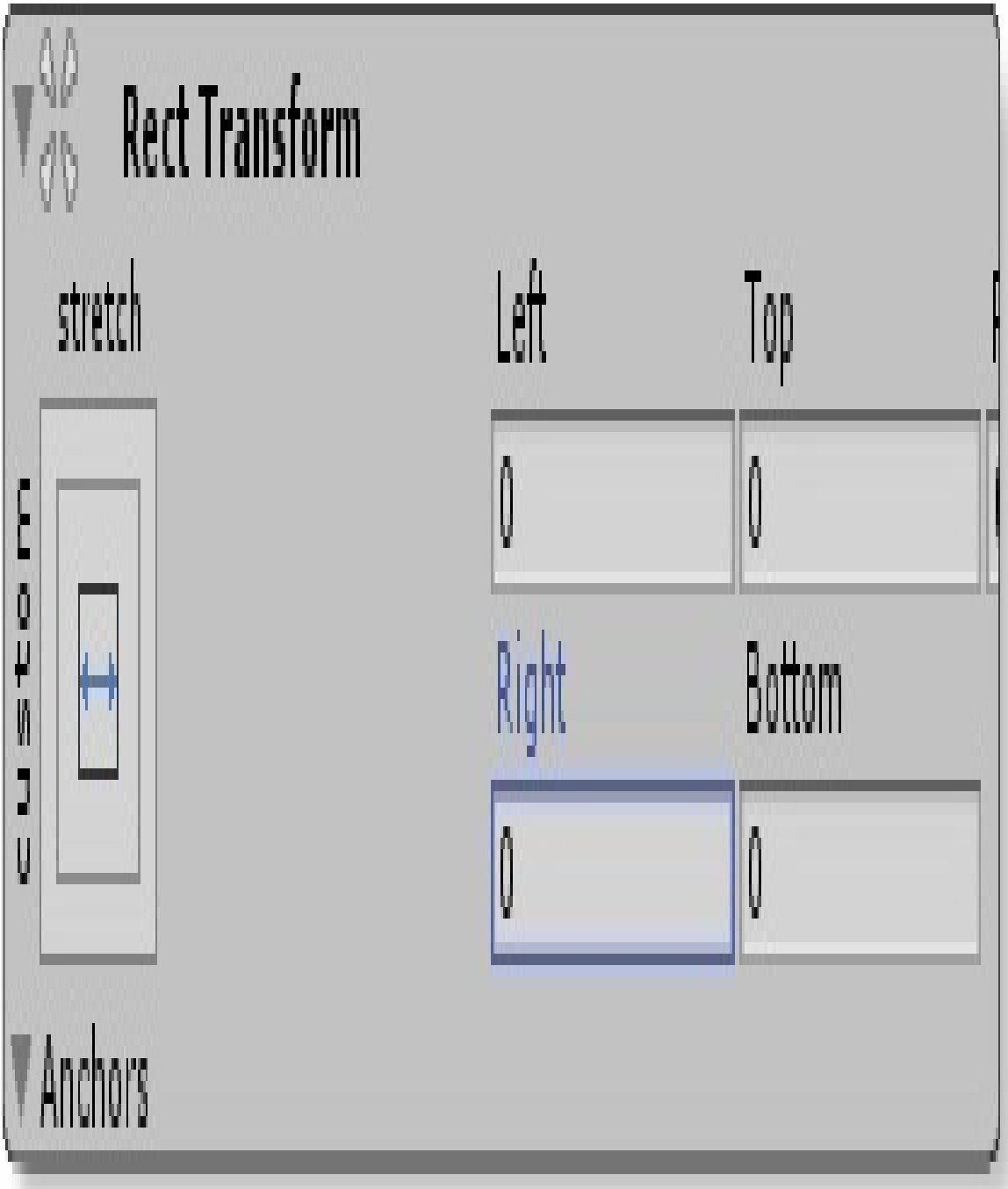


Visualización de Value del Slider

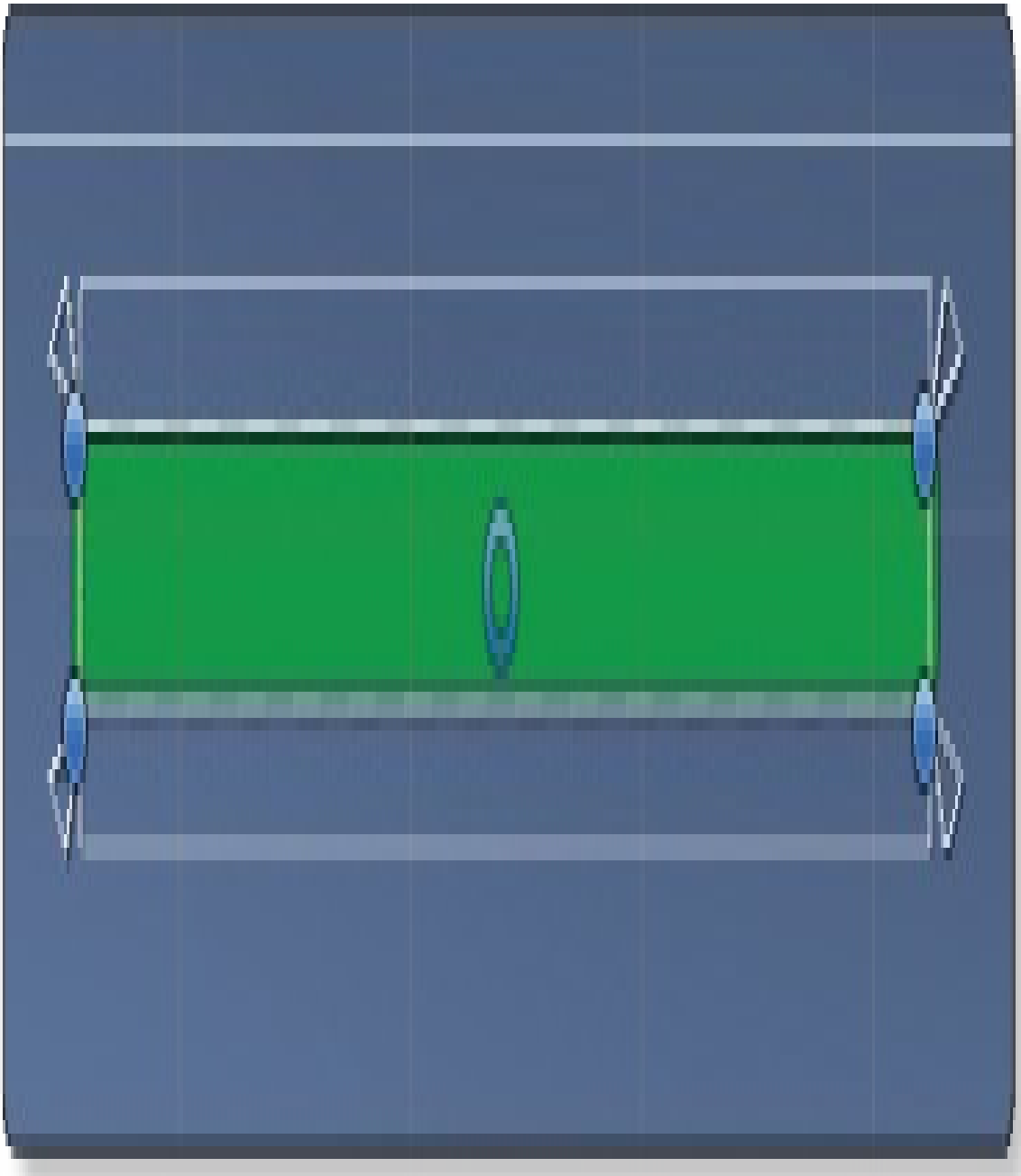
Seleccione “Fill Area” y asigne cero para “Left” y “Right”.

EventSystem
GUI
HealthBar
Slider
Background
Fill Area
Fill

Fill Area seleccionado


Rect Transform
stretch
Left
Top
0
0
Right
Bottom
0
0
Anchors

Parámetros de Fill Area



Visualización de Fill Areal

Vaya al script “PlayerHealth”, mande llamar la librería que se muestra y haga la variable que se indica.

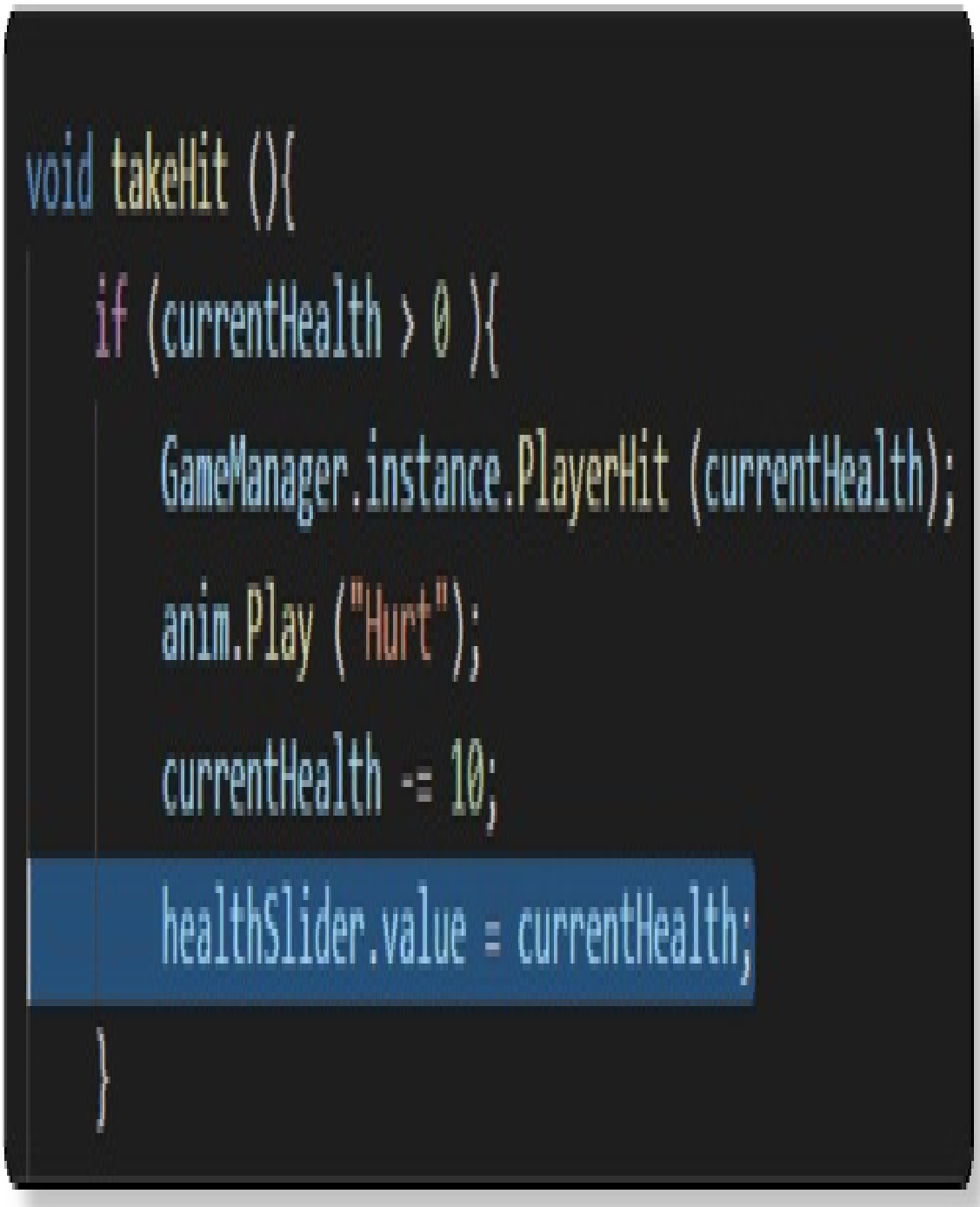
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PlayerHealth : MonoBehaviour {

    [SerializeField] int startingHealth = 100;
    [SerializeField] float timeSinceLastHit = 2.0f;
    [SerializeField] int currentHealth;
    [SerializeField]private float timer = 0f;
    [SerializeField] Slider healthSlider;
    private Animator anim;
```

Creación de variable tipo Slider

Dado que vamos a hacer referencia a elementos tipo UI debemos de mandar llamar la librería correspondiente.

Crearemos la variable tipo “Slider” que obsederá únicamente a objeto tipo “Slider” que le asignemos.

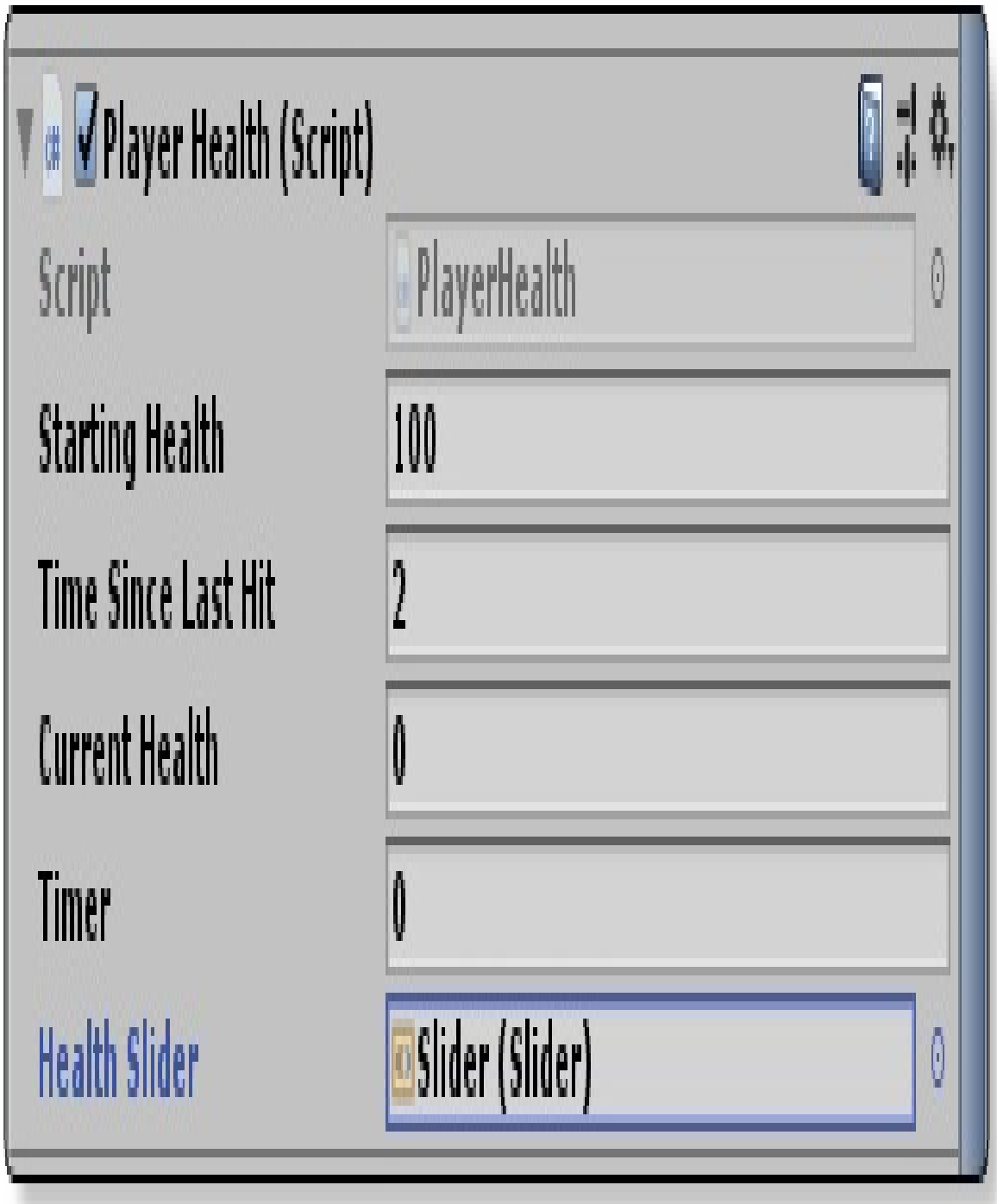
Escriba el siguiente código en la función “TakeHit”.


void takeHit (){
if (currentHealth > 0){
GameManager.instance.PlayerHit (currentHealth);
anim.Play ("Hurt");
currentHealth -= 10;
healthSlider.value = currentHealth;
}

Enlace entre Slider y salud del Player

De esta manera enlazamos la variable “currentHealth” a nuestro slider.

Vaya a Unity y asigne el en script el elemento “Slider”.


Player Health (Script)
Script
PlayerHealth
Starting Health
100
Time Since Last Hit
2
Current Health
0
Timer
0
Health Slider
Slider (Slider)

Asignación de Slider

Seleccione el objeto “GUI” y en “Inspector” vaya a “Canvas Scaler” y asigne los siguientes valores.

Canvas Scaler (Script)
UI Scale Mode
Scale With Screen Size
Reference Resolution
X 800
Y 600
Screen Match Mode
Match Width Or Height
Match
0
Width
Height
Reference Pixels Per Unit
100

Escala para GUI

Esto nos permitirá que nuestros elementos GUI se escalen en medida que se escale nuestra pantalla de juego.

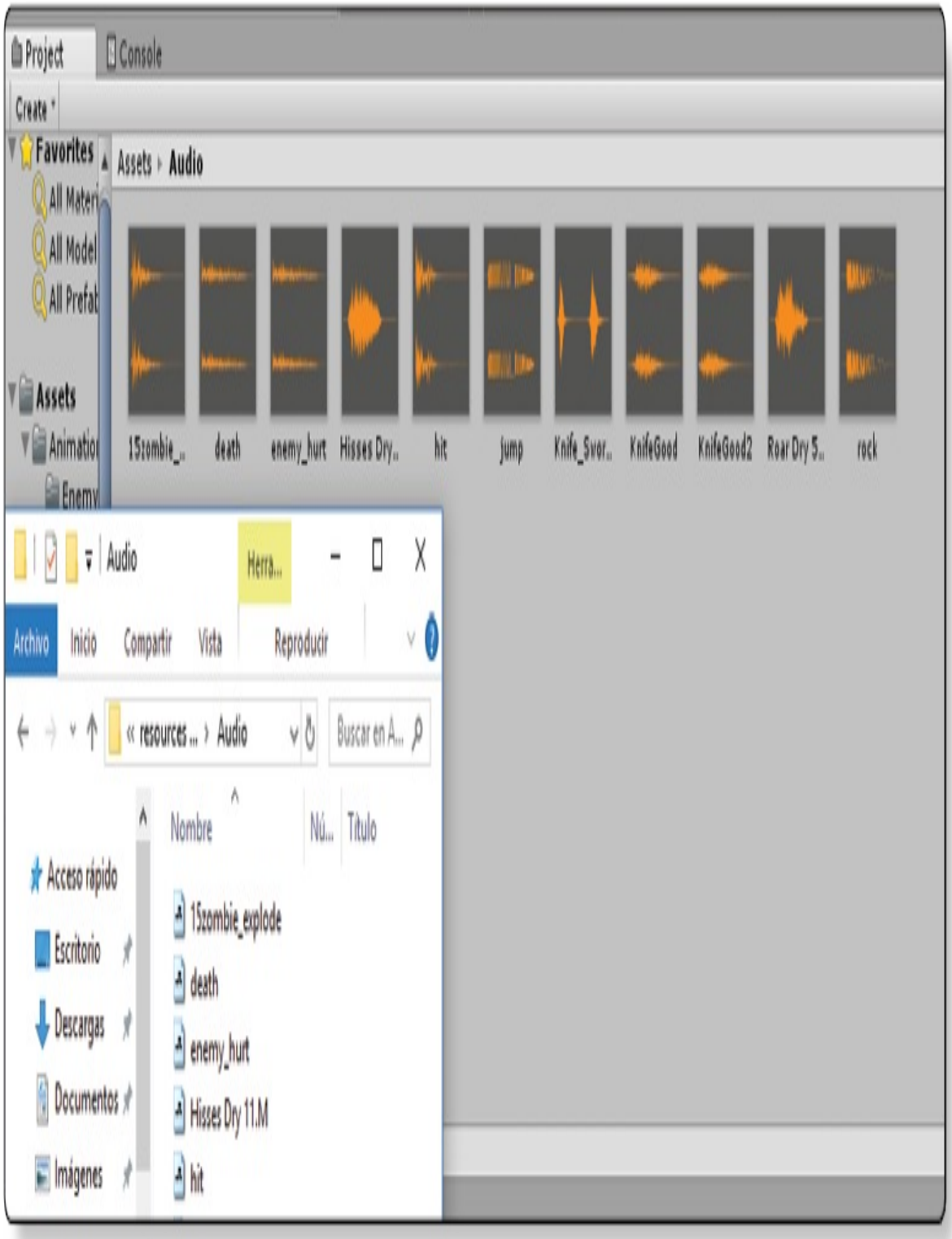
Previsualice el juego para observar como baja el slider conforme a su daño.

HealthBar funcionando correctamente

AÑADIR CLIPS DE AUDIO

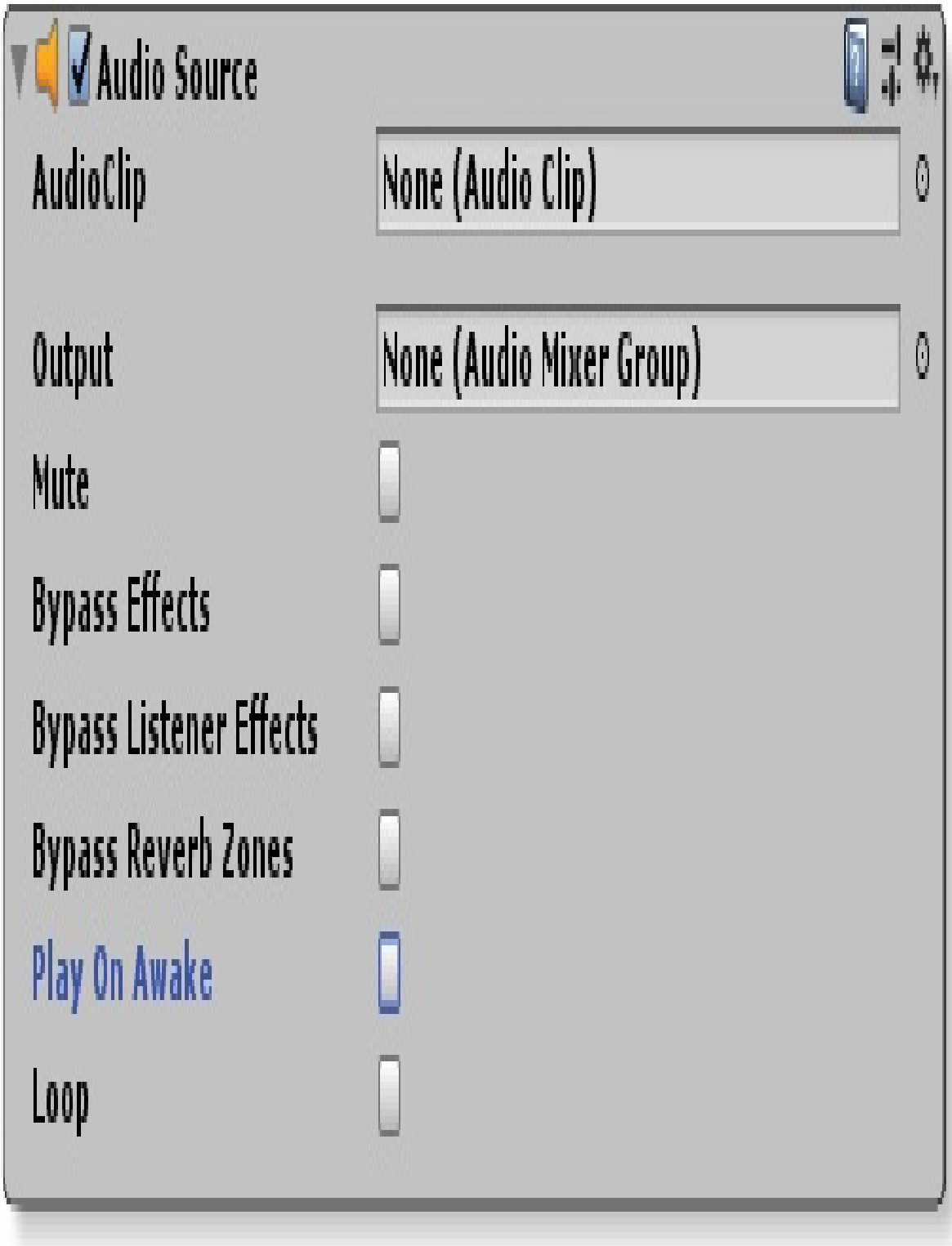
En este apartado aprenderemos de manera básica como añadir audio a nuestras acciones.

En “Project” haga una nueva carpeta, nómbrela “Audio” e importe los clips de audio de la carpeta externa “resources”.


Project
Console
Create
Favorites
Assets
Audio
15zombie_..
death
enemy_hurt
Hisses Dry..
hit
jump
Knife_Swor..
KnifeGood
KnifeGood2
Roar Dry 5..
rock
Audio
Herra...
Archivo
Inicio
Compartir
Vista
Reproducir
« resources ... › Audio
Buscar en A...
Nombre
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
15zombie_explode
death
enemy_hurt
Hisses Dry 11.M
hit

Importar archivos de audio

Seleccione “Player” y añada un componente “Audio Source” y desmarque la casilla “playOnAwake”.


Audio Source
AudioClip
None (Audio Clip)
Output
None (Audio Mixer Group)
Mute
Bypass Effects
Bypass Listener Effects
Bypass Reverb Zones
Play On Awake
Loop

Componente AudioSource

Dado que vamos a programar el audio de daño al “Player” queremos controlar su “momentum” para correr el audio, y por ello no queremos activa la opción “PlayOnAwake”.

Vaya al script “PlayerHealth”, haga las siguientes variables e inicialice.

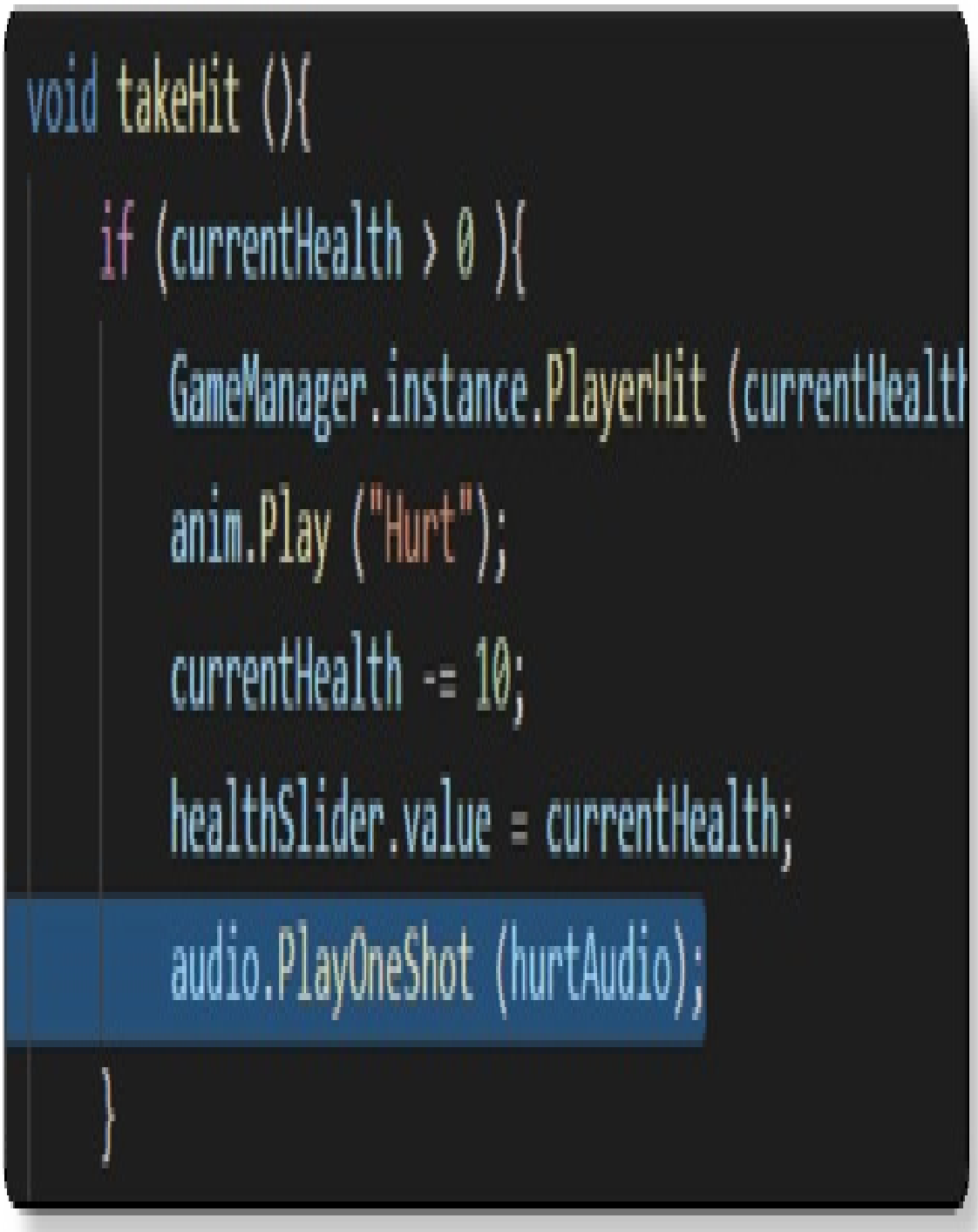
```
private  CharacterMovement characterMovement;

private AudioSource audio;
public AudioClip hurtAudio;

// Use this for initialization
void Start () {

    audio = GetComponent<AudioSource> ();
    anim = GetComponent<Animator> ();
    currentHealth = startingHealth;
    characterMovement = GetComponent<Character
```

Variables para controlar el audio

Con la variable “AudioSource” vamos a mandar llamar el componente “AudioSource” del “Player”.

Con la variable “AudioClip” vamos a asignarle físicamente el clip relativo al audio del daño.

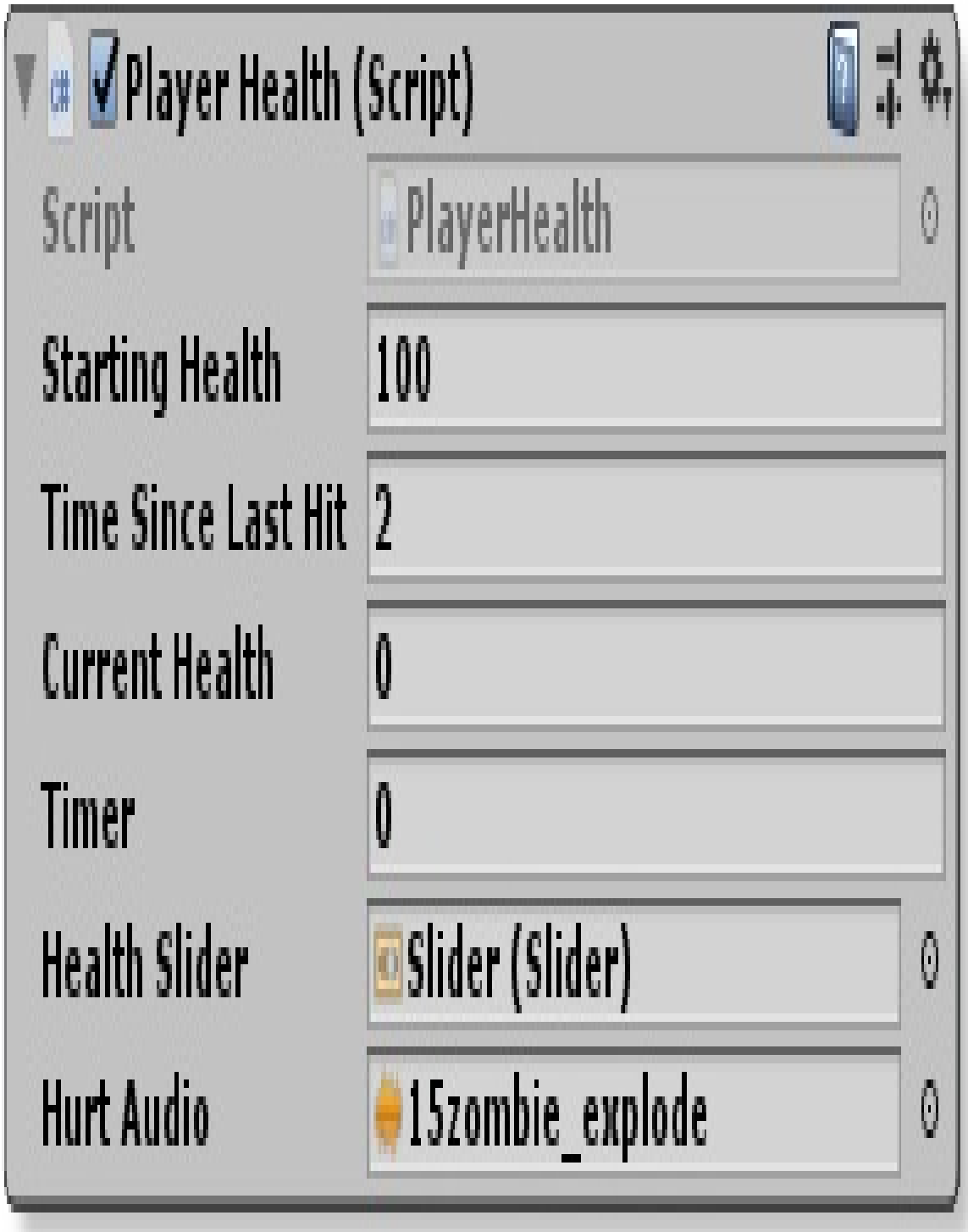
Vaya a la función “TakeHit” y escriba el siguiente código para mandar llamar el clip de audio.


void takeHit (){
if (currentHealth > 0){
GameManager.instance.PlayerHit (currentHealth
anim.Play ("Hurt");
currentHealth -= 10;
healthSlider.value = currentHealth;
audio.PlayOneShot (hurtAudio);
}

Código para llamar el clip de audio

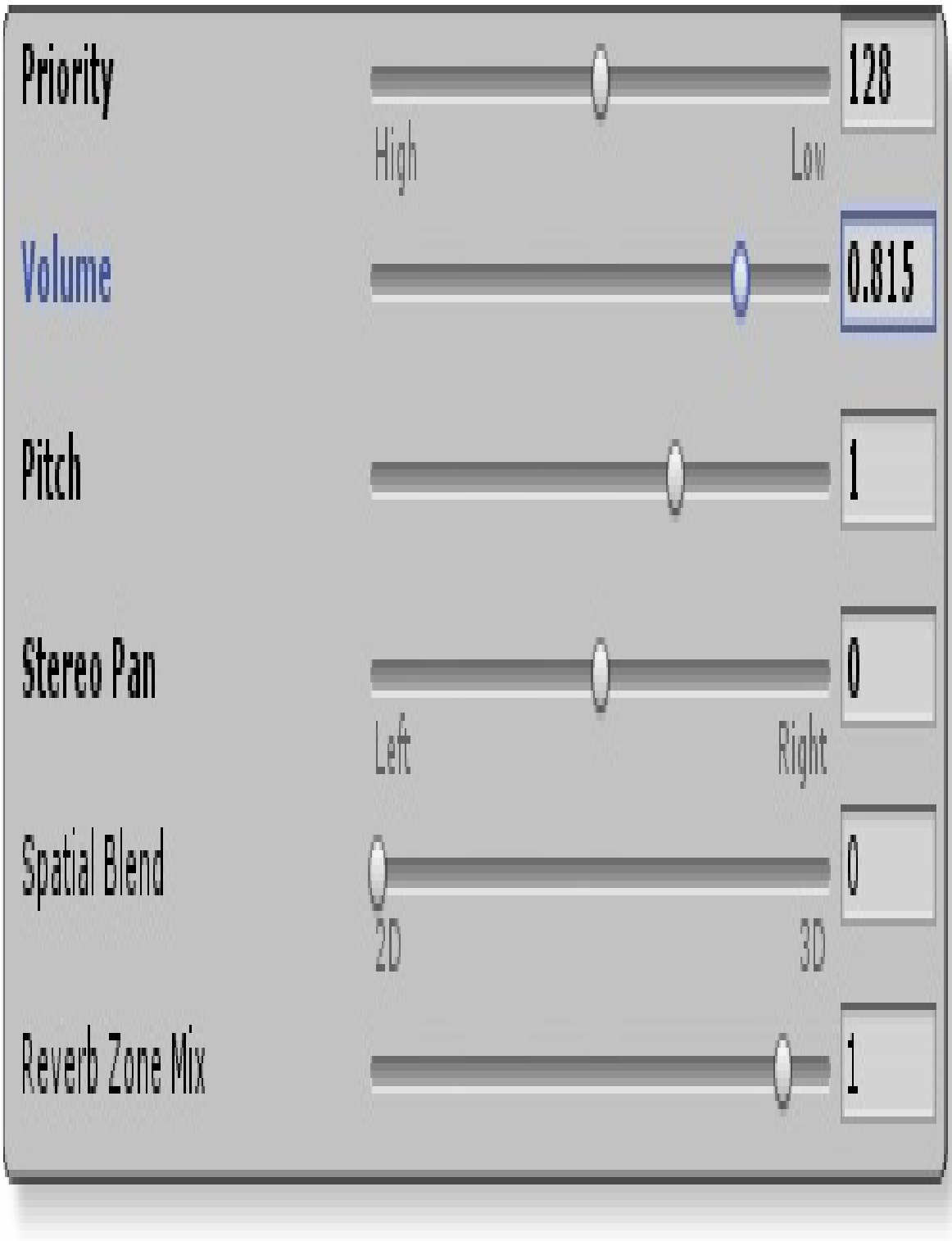
El comando “Play.OneShot” nos permite correr el audio una sola vez en el evento marcado.

En Unity asigne el audio correspondiente al script.


Player Health (Script)
Script
PlayerHealth
Starting Health
100
Time Since Last Hit
2
Current Health
0
Timer
0
Health Slider
Slider (Slider)
Hurt Audio
15zombie_explode

Asignación de clip de audio

Puede modificar los valores del audio en el componente AudioSource.


Priority
128
High
Low
Volume
0.815
Pitch
1
Stereo Pan
0
Left
Right
Spatial Blend
0
2D
3D
Reverb Zone Mix
1

Parámetros para modificar el audio

Previsualice el juego para escuchar el audio cuando el “Player” es lastimado.

RETO AL LECTOR

En este apartado el lector pondrá en práctica algunos de los temas vistos en este capítulo.

Insertar los clips de audio que faltan.

Salto del personaje

Muerte del personaje

Daño al enemigo

Muerte al enemigo

Ataque del Player

CONCLUSIÓN

A través de este capítulo hemos aprendido los siguientes conceptos.

Creación de enemigos básicos

Uso del componente “NavMeshAgent” para la inteligencia artificial

Creación de un “GameManager”

Creación de estados de juego

Programación de funciones “IEnumerator”

Inteligencia artificial para ataques

Animaciones de daño y muerte

Creación y programación básica de elementos GUI

Inserción de clips de audio

TRABAJO CON ENEMIGO SHOOTER

INTRODUCCIÓN

En este capítulo trabajaremos añadiendo un segundo enemigo tipo “Shooter”, es decir que pueda lanzar proyectiles y en este caso será una flecha.

Trabajaremos con su funcionalidad de ataque, sus animaciones y el estudiante podrá poner en práctica los temas vistos en el capítulo anterior.

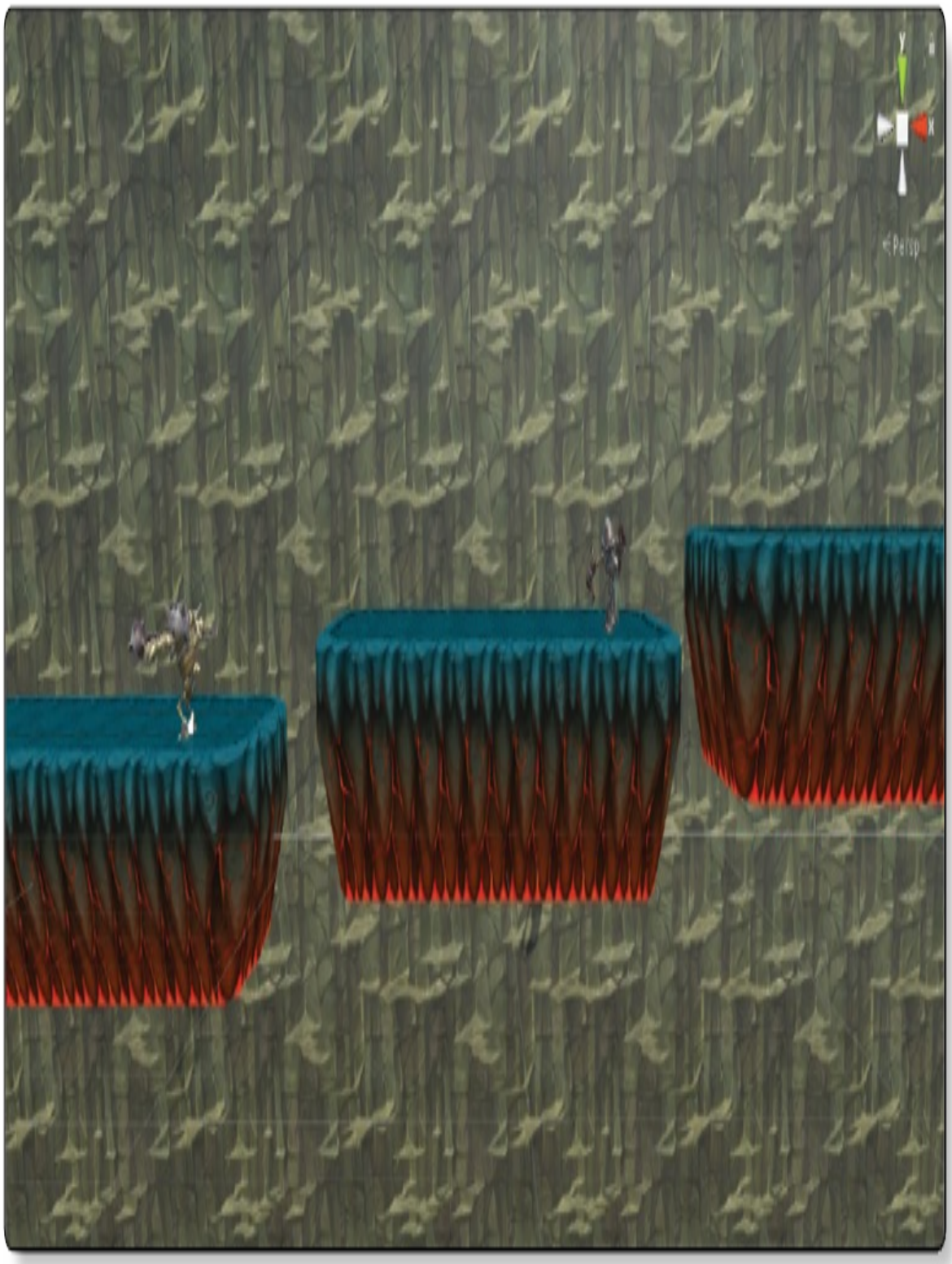
Se verán algunos conceptos nuevos como AnimationEvents y finalizará con un reto donde el estudiante podrá poner en práctica lo aprendido hasta ahora conforme a la inteligencia artificial de los enemigos.

PREPARACIÓN DEL ENEMIGO SHOOTER

En este apartado prepararemos el set para la importación de nuestro nuevo enemigo tipo “Shooter”. Se importará el enemigo, se texturizará y se le asignarán los elementos necesarios para asignar sus animaciones y su programación de comportamientos.

También se preparará el proyectil tipo flecha que disparará el enemigo.

Con los conceptos aprendidos importe, coloque y texturize el modelo “skeleton_archer”, colocándolo en la siguiente plataforma.


y
x
Persp

Colocación del nuevo enemigo

Vista del nuevo enemigo

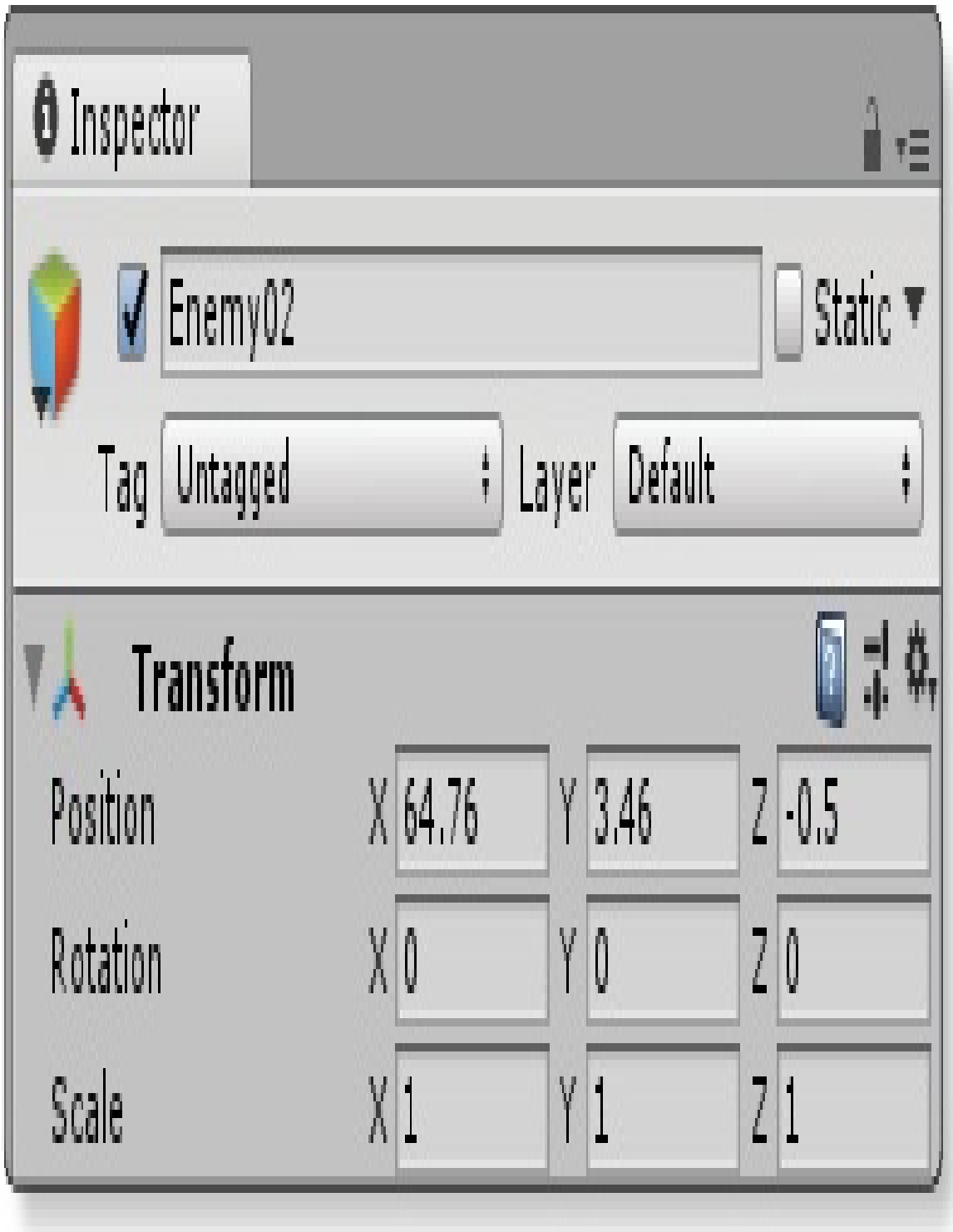
Elimine el componente “Animator” del objeto “skeleton_archer”.

Con los conceptos aprendidos haga un nuevo “empty object”, nómbrelo “Enemy02” colóquelo en relación a “skeleton_archer” y conviértalo en padre.

EventSystem
Enemy02
skeleton_archer

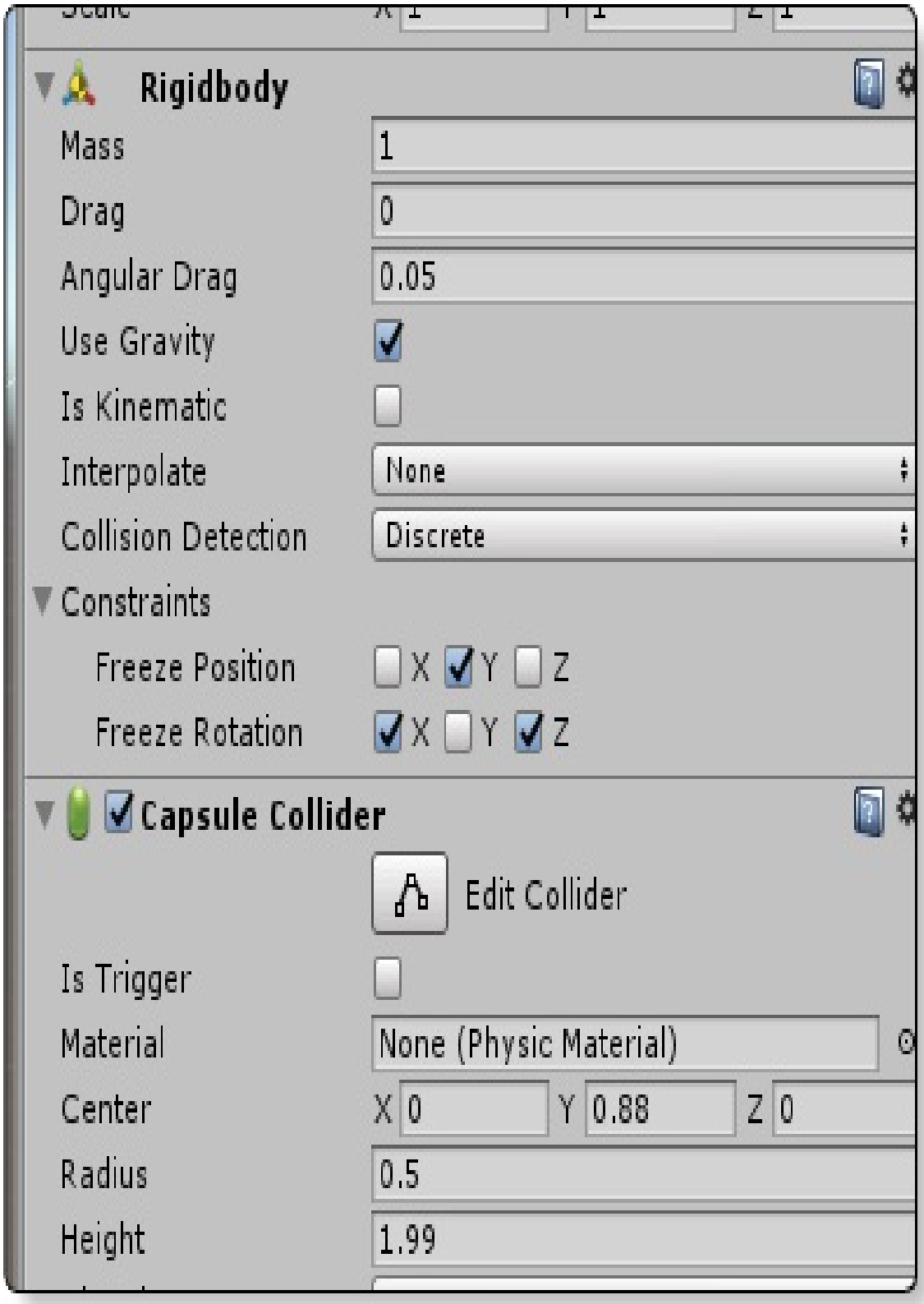
Creación del objeto Enemy02

Posicionamiento del objeto Enemy02


Inspector
Enemy02
Static
Tag
Untagged
Layer
Default
Transform
Position
X 64.76
Y 3.46
Z -0.5
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1

Rotaciones en 0,0,0 del objeto Enemy02

Asigne los componentes “Rigidbody” y “Capsulle Collider” a “Enemy02” con los valores que se muestran.


Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Z
Freeze Rotation
X
Y
Z
Capsule Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 0.88
Z 0
Radius
0.5
Height
1.99

Componentes del objeto Enemy02

Capsule Collider

Asigne un componente “Animator” a “Enemy02”.

De la carpeta “resources” importe el modelo “arrow”, texturice y añádalo a la escena y elimine su componente “Animator”.

Importación del proyectil

Haga un nuevo “Empty Object”, colóquelo en relación a la flecha, renómbrelo “ArrowSpawn” y asigne un “label” color verde.

ArrowSpawn

Colocación de objeto “ArrowSpawn”

Empareje “ArrowSpawn” con “Enemy02”.

Haga un duplicado de “ArrowSpawn” sáquelo de la jerarquía y renómbrelo “Arrow_PFB”.

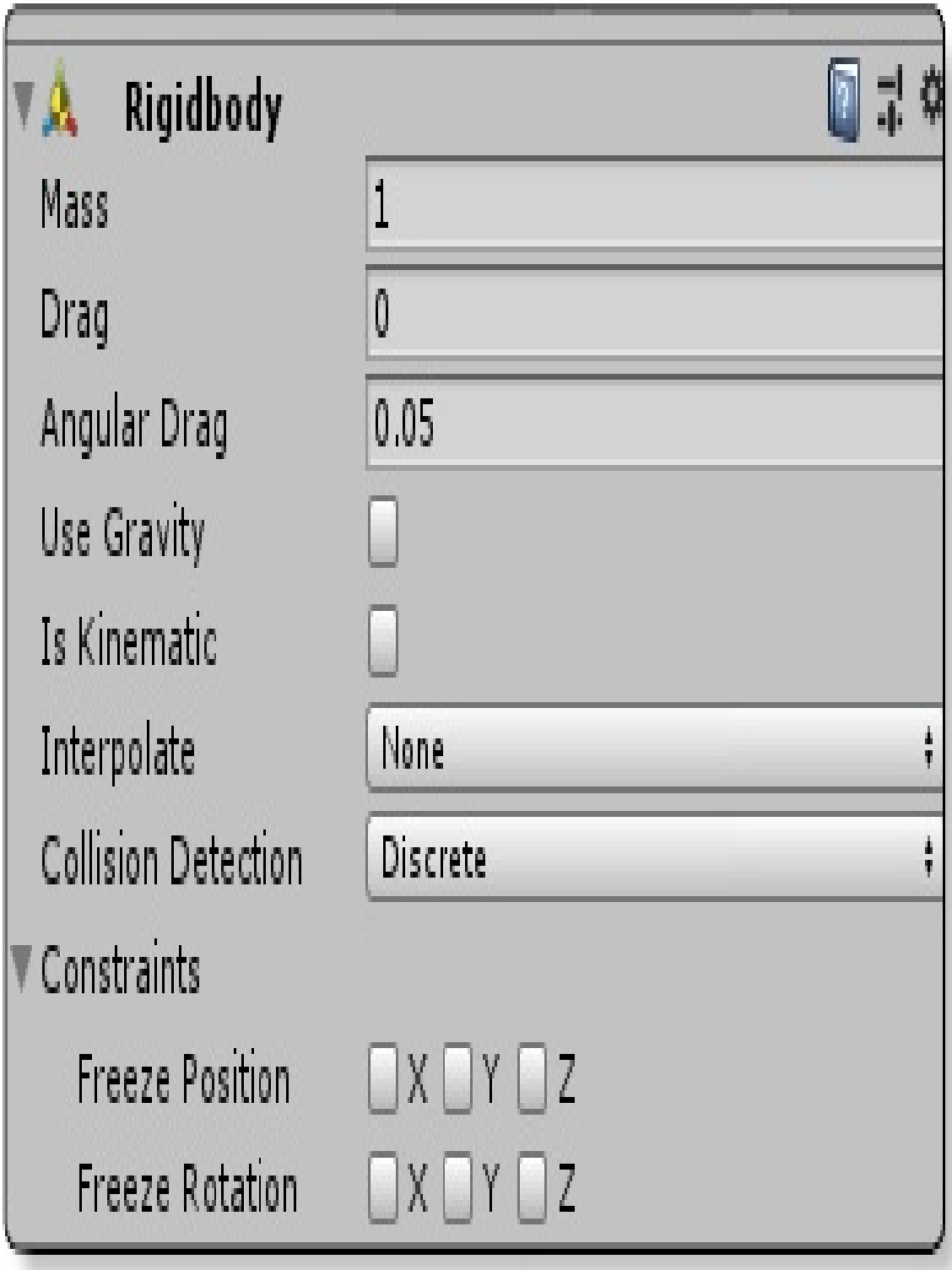
Debe de quedar en la misma posición que “ArrowSpawn”.

Empareje “Arrow” con “Arrow_PFB”.

EventSystem
Enemy02
skeleton_archer
ArrowSpawn
Arrow_PFB
arrow

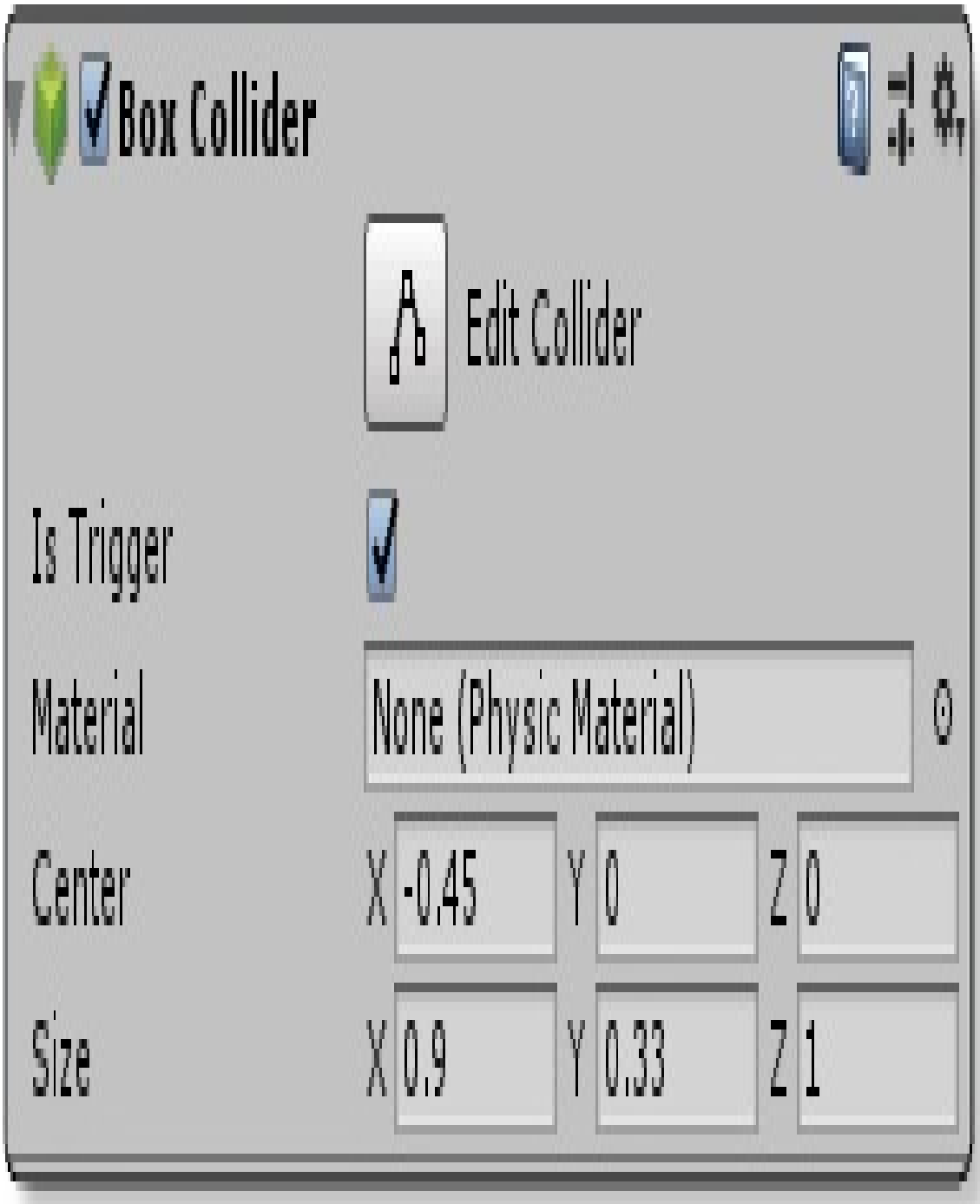
Emparentamiento del proyectil

Seleccione el objeto “Arrow_PFB” y asigne sus componentes “RigidBody” y “BoxCollider” con los valores que se muestran.


Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Z
Freeze Rotation
X
Y
Z

Componente Rigidbody

Desactivemos la casilla “Use Gravity” ya que es un objeto sin necesidad de gravedad.


Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X -0.45
Y 0
Z 0
Size
X 0.9
Y 0.33
Z 1

Componente BoxCollider

Activamos la casilla “Is Trigger” ya que será necesario que el objeto reaccione ante eventos de colisión.

Valores BoxCollider para el proyectil

El Box Collider es ancho para que no tenga ningún problema en colisionar con el Collider del Player conforme a su posición.

Convierta el objeto “Arrow_PFB” en Prefab y elimínelo de la escena.

Vaya al folder de “Animations”, haga un nuevo folder y nómbrelo “Enemy02 Animations”.

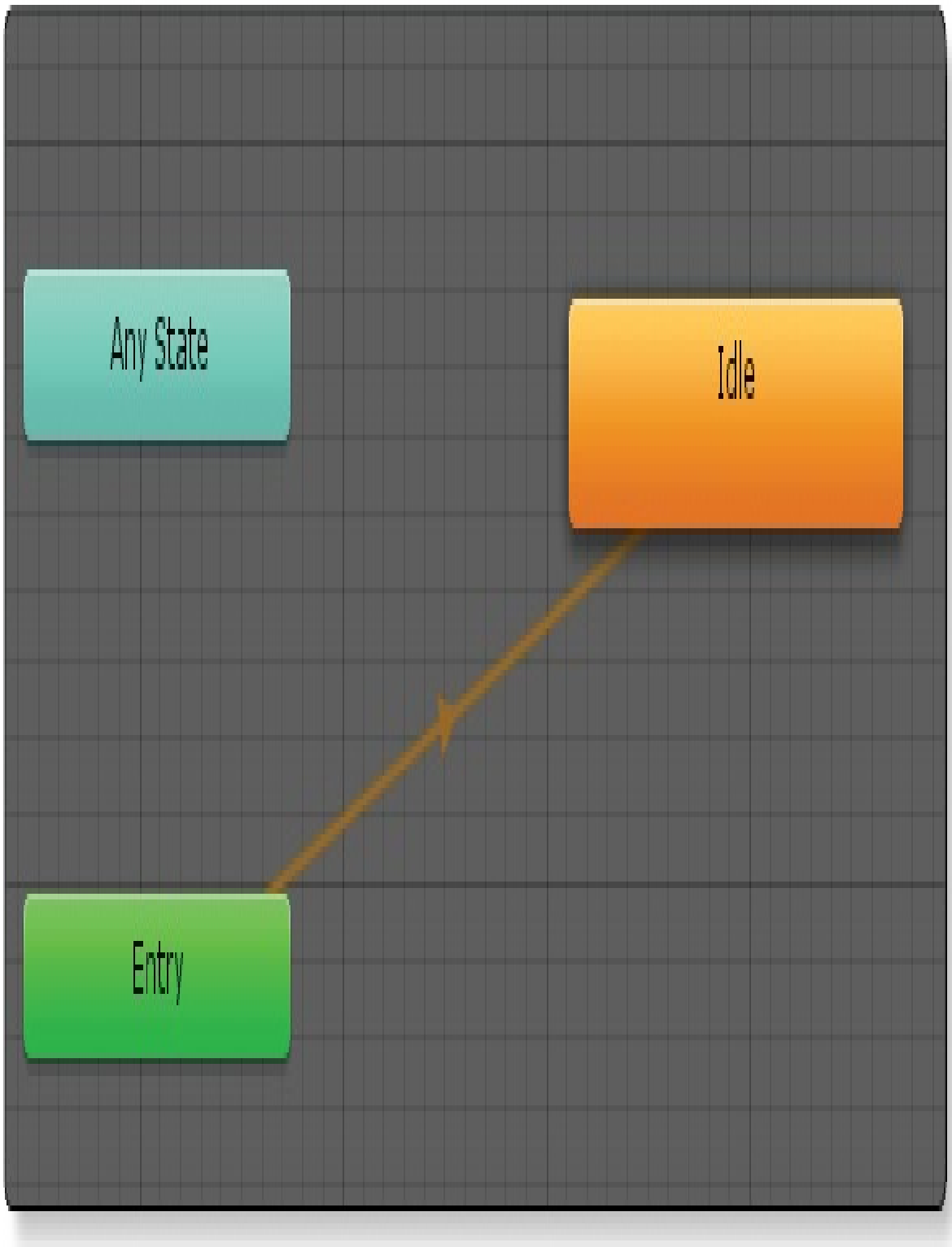
Dentro del nuevo folder haga un nuevo “AnimatorController” y renómbrelo “Enemy02AC”.

En el componente “Animator” de “Enemy02” asigne su controlador y su avatar.

Animator
Controller
Enemy02AC
Avatar
skeleton_archerAvatar
Apply Root Motion
Update Mode
Normal
Culling Mode
Always Animate

Componente Animator de Enemy02

Con los conceptos aprendidos previamente, haga la animación de “Idle” para “Enemy02” y asígnela en la ventana “Animator”.


Any State
Idle
Entry

Ventana Animator

Previsualice el juego para observar lo siguiente:

En “Enemy02” corre su animación de “Idle”.

El proyectil del “Player” (knife) colisiona con “Enemy02” y se destruye el proyectil.

Animación Idle de Enemy02

PROGRAMACIÓN DE ATAQUE DEL ENEMIGO SHOOTER

En este apartado programaremos el comportamiento y la animación de ataque del enemigo tipo “Shooter” en relación a la cercanía o lejanía de nuestro “Player”.

Haga un nuevo C# Script, nómbrelo “ArcherAttack” y asígnelo al objeto “Enemy02”.

Abra el script, haga las siguientes variables e inicialícelas.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ArcherAttack : MonoBehaviour {

    [SerializeField] private float range = 10f;
    [SerializeField] private float timeBetweenAttacks = 1f;

    private Animator anim;
    private GameObject player;
    private bool playerInRange;

    public float arrowSpeed = 600f;
    public Transform arrowSpawn;
    public Rigidbody arrowPrefab;

    Rigidbody clone;

    // Use this for initialization
    void Start () {
        arrowSpawn = GameObject.Find ("ArrowSpawn").transform;
        anim = GetComponent<Animator> ();
        player = GameManager.instance.Player;
    }
```

Script ArcherAtack

Escriba el siguiente código en la función Upddate para comprobar cuando el “Player” se encuentre en rango de ser atacado.

```
// Update is called once per frame
void Update () {

    if (Vector3.Distance (transform.position, player.transform.position) < range) {
        playerInRange = true;
    } else {
        playerInRange = false;
    }
    print ("Player in range: " + playerInRange);
}
```

Código para comprobación de Player en rango

Previsualice el juego para observar que cuando el “Player” se encuentra en el rango de ataque del enemigo, aparece el mensaje en la consola.

Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
2
0
0
Player in range: False
UnityEngine.MonoBehaviour:print(Object)
465
Player in range: True
UnityEngine.MonoBehaviour:print(Object)
190

Comprobación en consola de Player en rango

En Unity asigne los objetos a las variables públicas del script.

Archer Attack (Script)
Script
ArcherAttack
Range
10
Time Between Attacks
1
Arrow Speed
600
Arrow Spawn
ArrowSpawn (Transform)
Arrow Prefab
Arrow_PFB (Rigidbody)

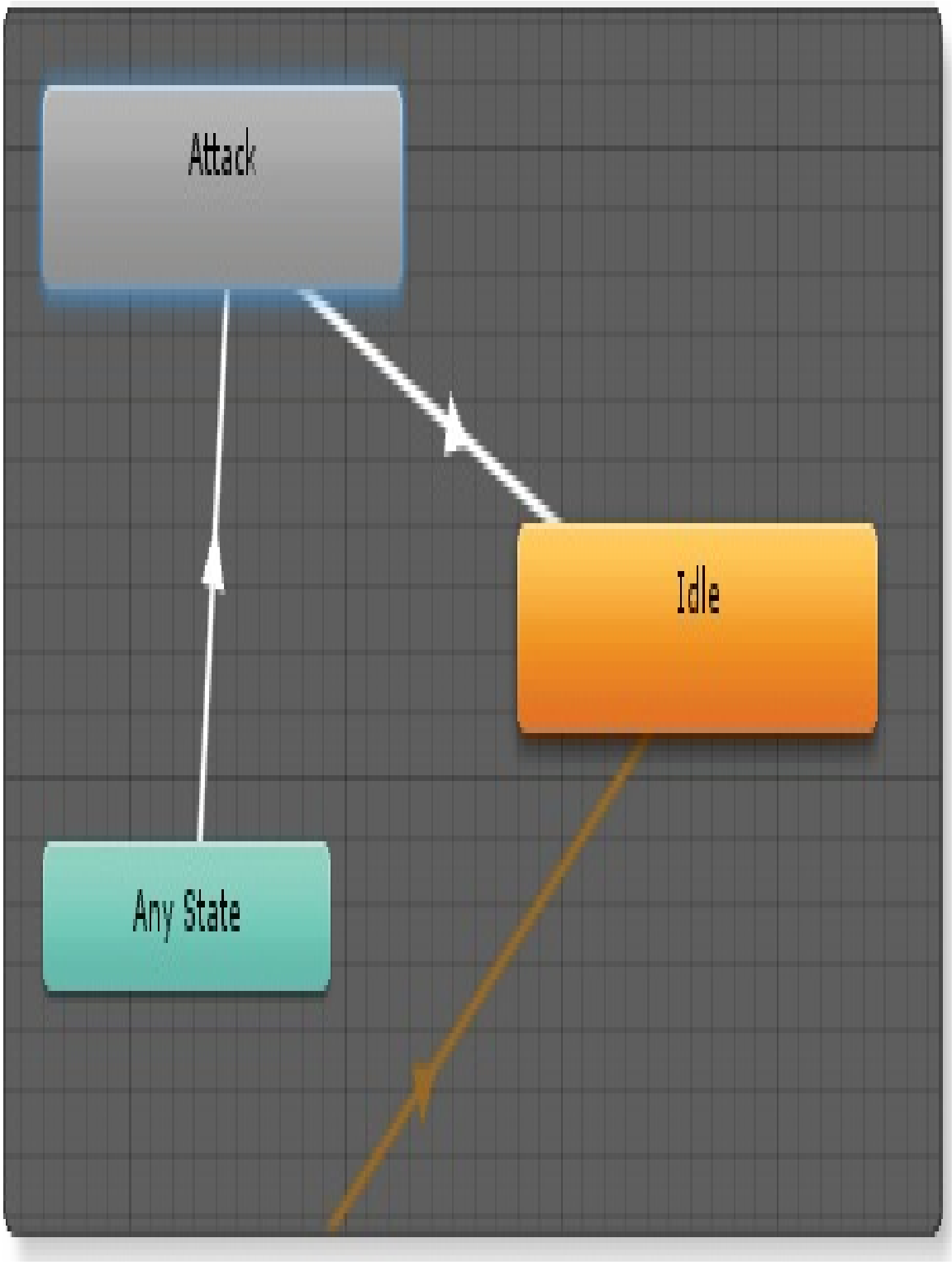
Asignación de elementos en componente Script

Con los conceptos aprendidos haga la animación de ataque del enemigo, nómbrela “Attack” y asígnela en “Animator” con las transiciones indicadas.

frame: 0180-0219: attack.

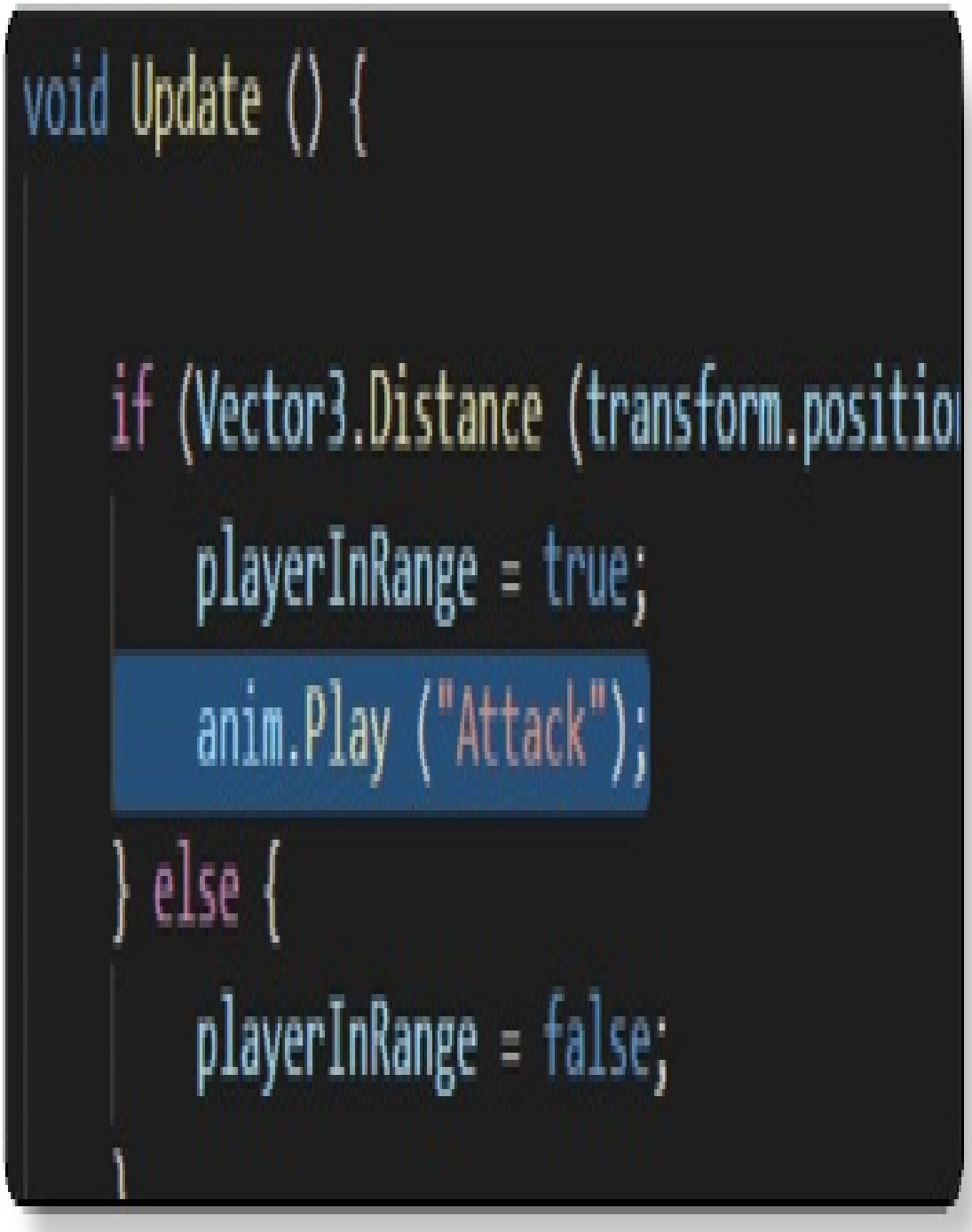
NOTA:

Las transiciones son sin parámetros.


Attack
Idle
Any State

Ventana Animator

Vaya al script de “ArcherAttack” y escriba el siguiente código en la función “Update” para correr la animación de ataque cuando el “Player” se encuentre en rango.


void Update () {
if (Vector3.Distance (transform.positio
playerInRange = true;
anim.Play ("Attack");
} else {
playerInRange = false;
}

Comando para correr la animación de ataque

Con este comando mandamos llamar correr la animación sin ningún parámetro.

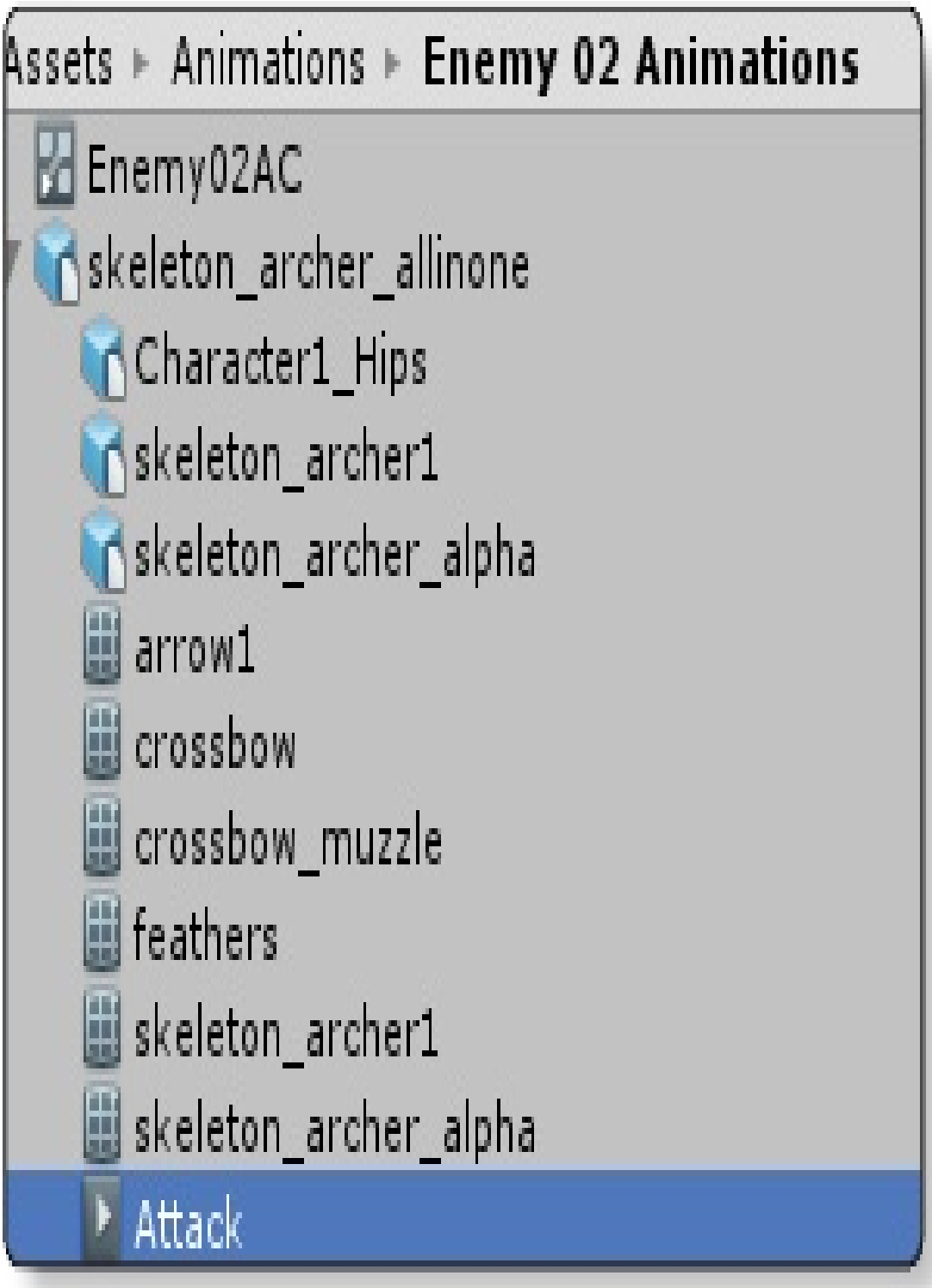
Previsualice el juego para observar que cuando el “Player” se encuentra en rango de ataque, la animación de ataque del enemigo comienza a correr y cuando deja de encontrarse en rango, vuelve a correr la animación “Idle” del enemigo.

Animación de ataque

PROGRAMACIÓN DEL PROYECTIL DEL ENEMIGO SHOOTER

En este apartado vamos a crear un “Animation Event”, a través del cual vamos a asignar la aparición del proyectil, así como su debido movimiento y desaparición en colisión.

En la ventana “Project” localice y seleccione la animación del ataque del enemigo y clic en el botón “Edit” en Inspector.

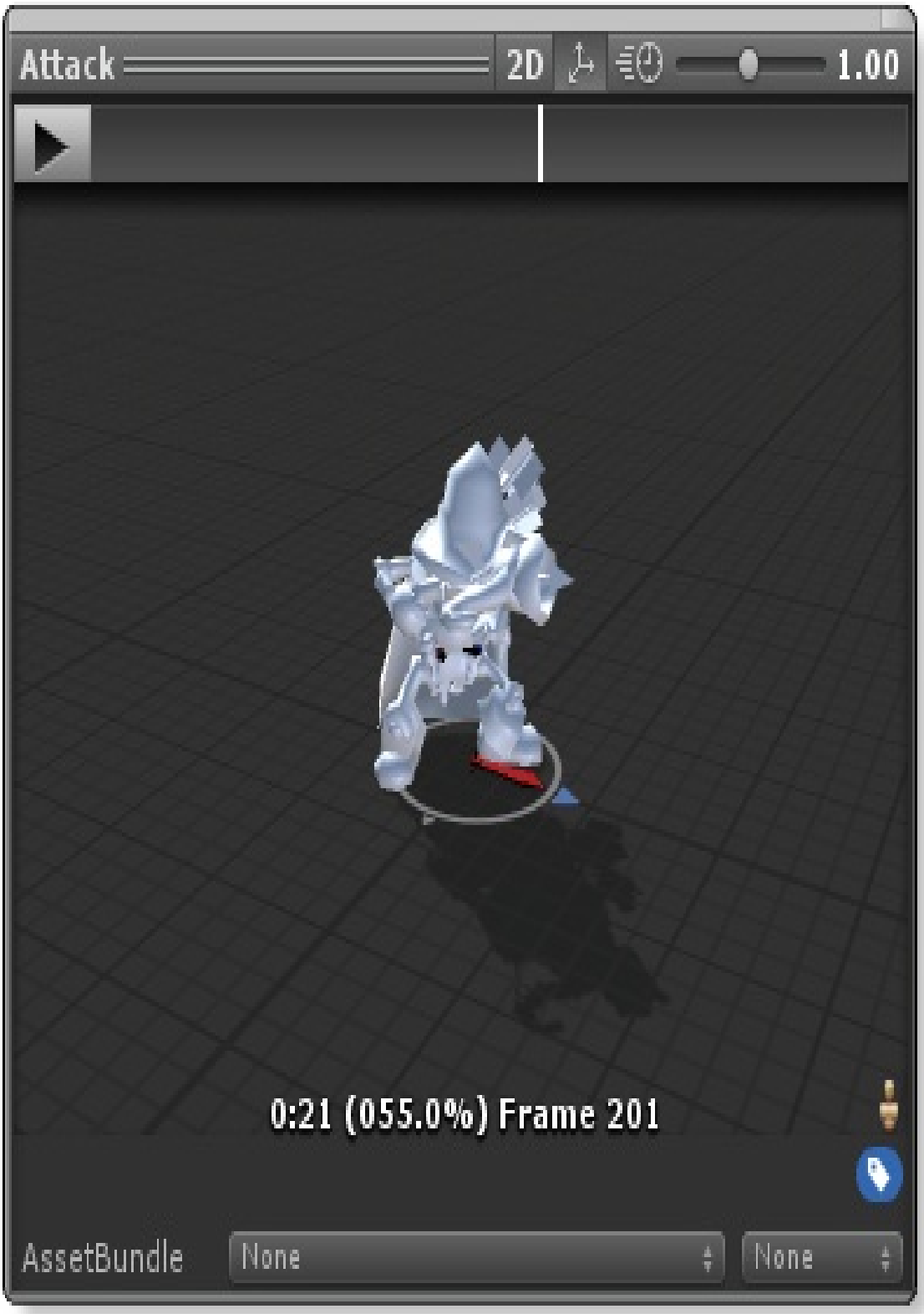

Assets ▸ Animations ▸ Enemy 02 Animations
Enemy02AC
skeleton_archer_allinone
Character1_Hips
skeleton_archer1
skeleton_archer_alpha
arrow1
crossbow
crossbow_muzzle
feathers
skeleton_archer1
skeleton_archer_alpha
Attack

Animación de ataque de Enemy02

Inspector
Attack
Edit...
Length 1.300
30 FPS

Botón de edición de la animación de ataque

En la ventana de previsualización de la animación coloque el tiempo en el momento en el que el enemigo debe de disparar el proyectil.

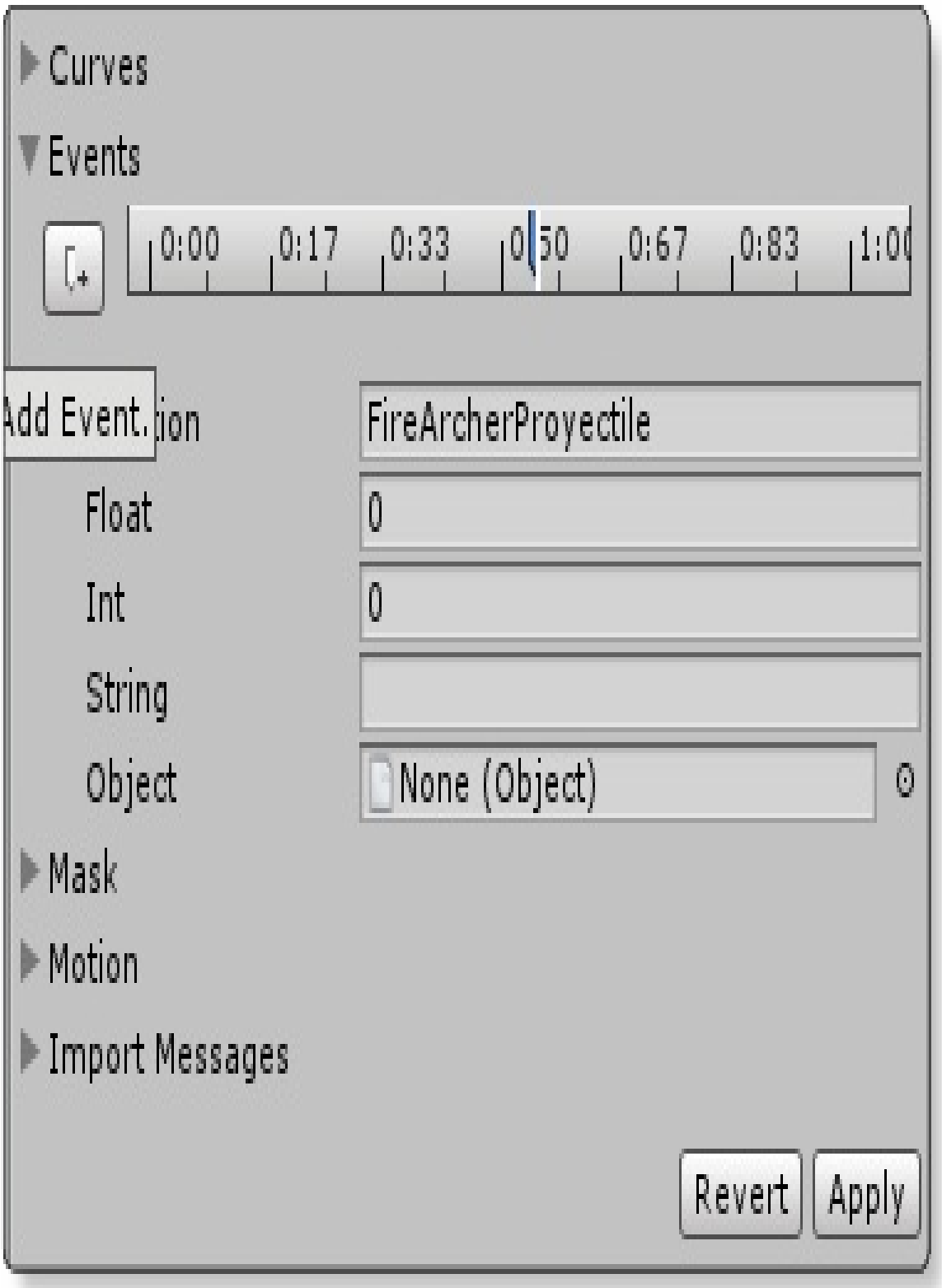

Attack
2D
1.00
0:21 (055.0%) Frame 201
AssetBundle
None
None

Previsualización del momento de ataque

En Inspector expanda la opción de “Events”, clic en el botón “Add Event” para crear un nuevo evento en el tiempo seleccionado y nómbrelo “FireArcherProyectile” y clic en Apply.

NOTA:

Es de suma importancia hacer clic en Apply para que se realice el evento de animación.


Curves
Events
0:00
0:17
0:33
0:50
0:67
0:83
Add Event...
FireArcherProyectile
Float
0
Int
0
String
Object
None (Object)
Mask
Motion
Import Messages
Revert
Apply

Creación de AnimationEvent

El evento aparecerá como un indicador azul.

Vaya al script ArcherAttack y haga la función pública que se muestra.

```
    print ("Player in range: " + playerInRange);
}

public void FireArcherProyectile(){
clone = Instantiate (arrowPrefab, arrowSpawn.position, arrowSpawn.rotation) as Rigidbody;
clone.AddForce (-arrowSpawn.transform.right * arrowSpeed);
}
```

Programación de la función del proyectil

NOTA:

El nombre de la función debe de corresponder exactamente al nombre del evento creado

Dado que dentro del lenguaje no tenemos la definición “left”, por ello iniciamos la definición de “AddForce” con el signo de “menos” (-)

Previsualice el juego para observar como dispara el proyectil el enemigo

Previsualización del proyectil

PROBLEMA: Las instancias de la flecha no se destruyen.

SOLUCIÓN: Ya tenemos un script para la destrucción de proyectiles y es el script “KnifeDestruction”, vamos a asignarlo al Prefab de la flecha.

En la ventana de “Project” localice y seleccione el objeto “Arrow_PFB”.

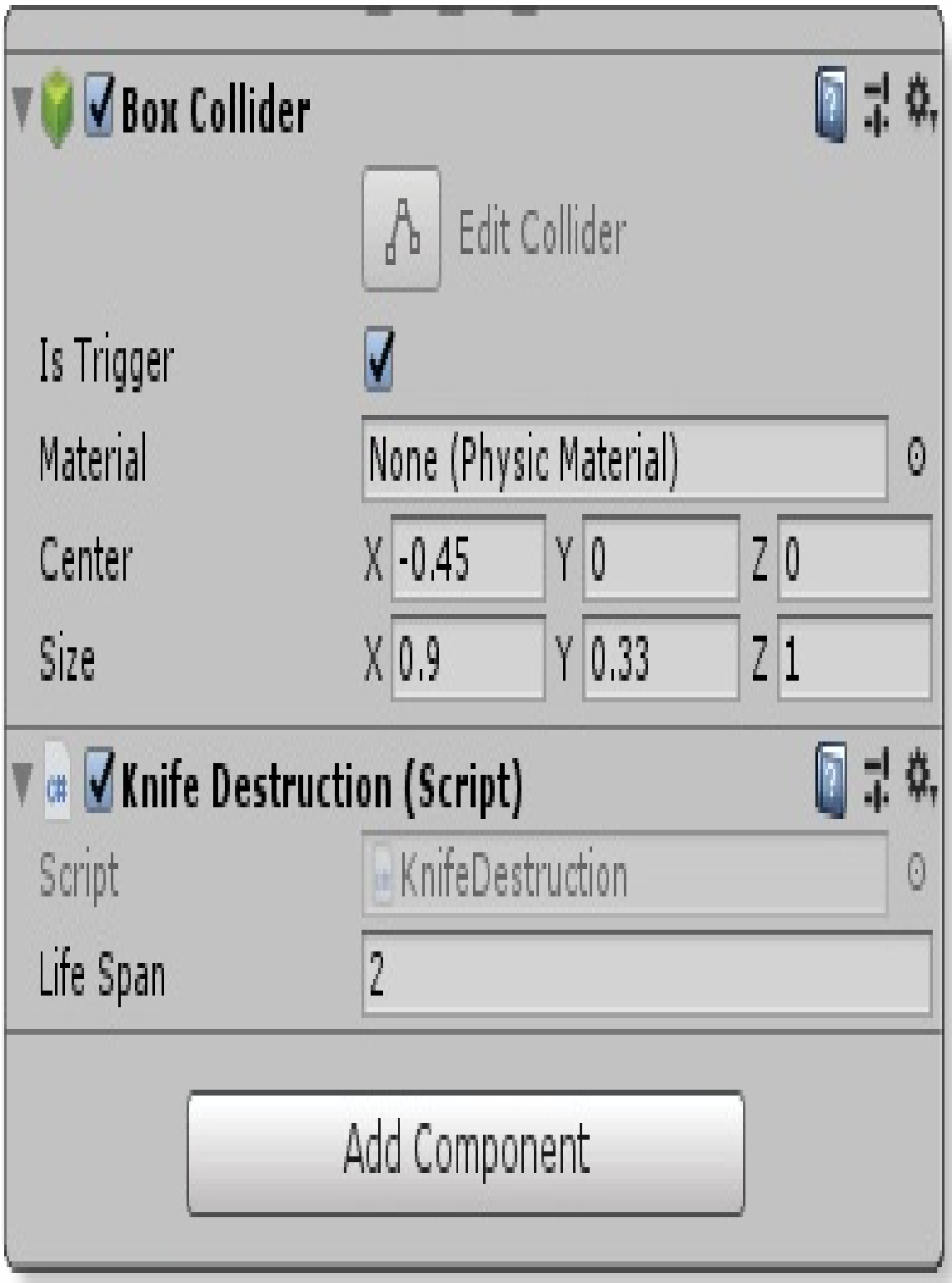
En “Inspector” active la casilla de “Lock” ubicada en la parte superior derecha de “Inspector”

Static

Bloqueo de Inspector

Con esta función bloqueamos la ventana “Inspector” en relación a lo que tenemos en ese momento seleccionado y evitando que cambie la información al seleccionar otro elemento.

Localice el script “KnifeDestruction” y asígnelo a “Arrow_PFB”.


Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X -0.45
Y 0
Z 0
Size
X 0.9
Y 0.33
Z 1
Knife Destruction (Script)
Script
KnifeDestruction
Life Span
2
Add Component

Script KnifeDestruction asignado

Desbloquee “Inspector”.

Previsualice el juego para observar que la flecha se destruye después de 2 segundos y cuando colisiona con un objeto.

PROGRAMACIÓN DEL DAÑOS

En este apartado nos ocuparemos de programar las animaciones de daño y muerte del enemigo, así como de su destrucción al alcanzar su salud a cero.

Asigne el tag “Weapon” al objeto “Arrow_PFB”.

Dado que ya tenemos programado el daño al Player en relación al tag “Weapon”, únicamente debemos de asignarle el tag al proyectil y le hará daño al Player.

Haga la animación de muerte de Enemy02, nómbrela “Die”.

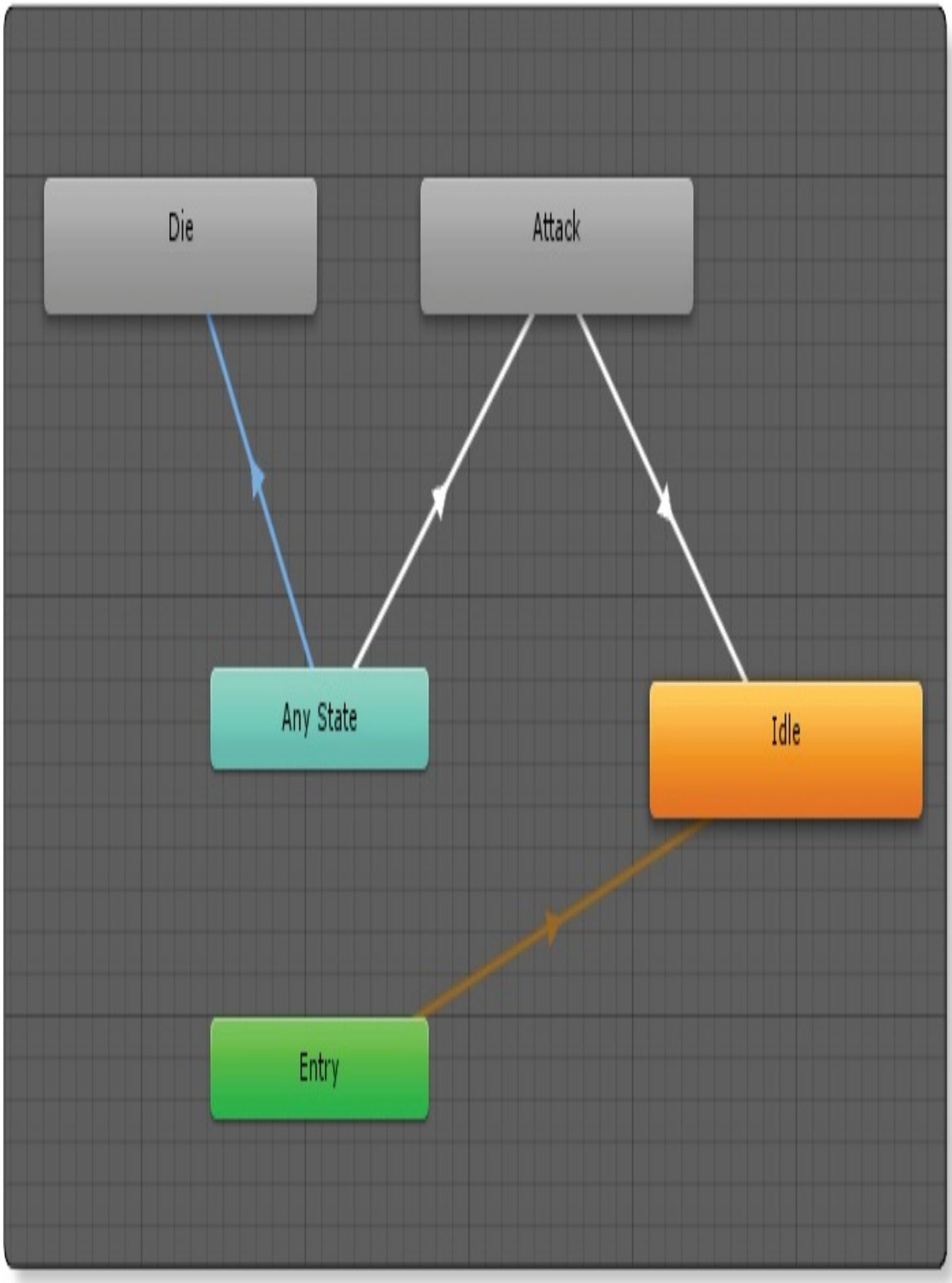
frame: 0220-0249: die

Haga la animación de daño de Enemy02” y nómbrela “Hurt”.

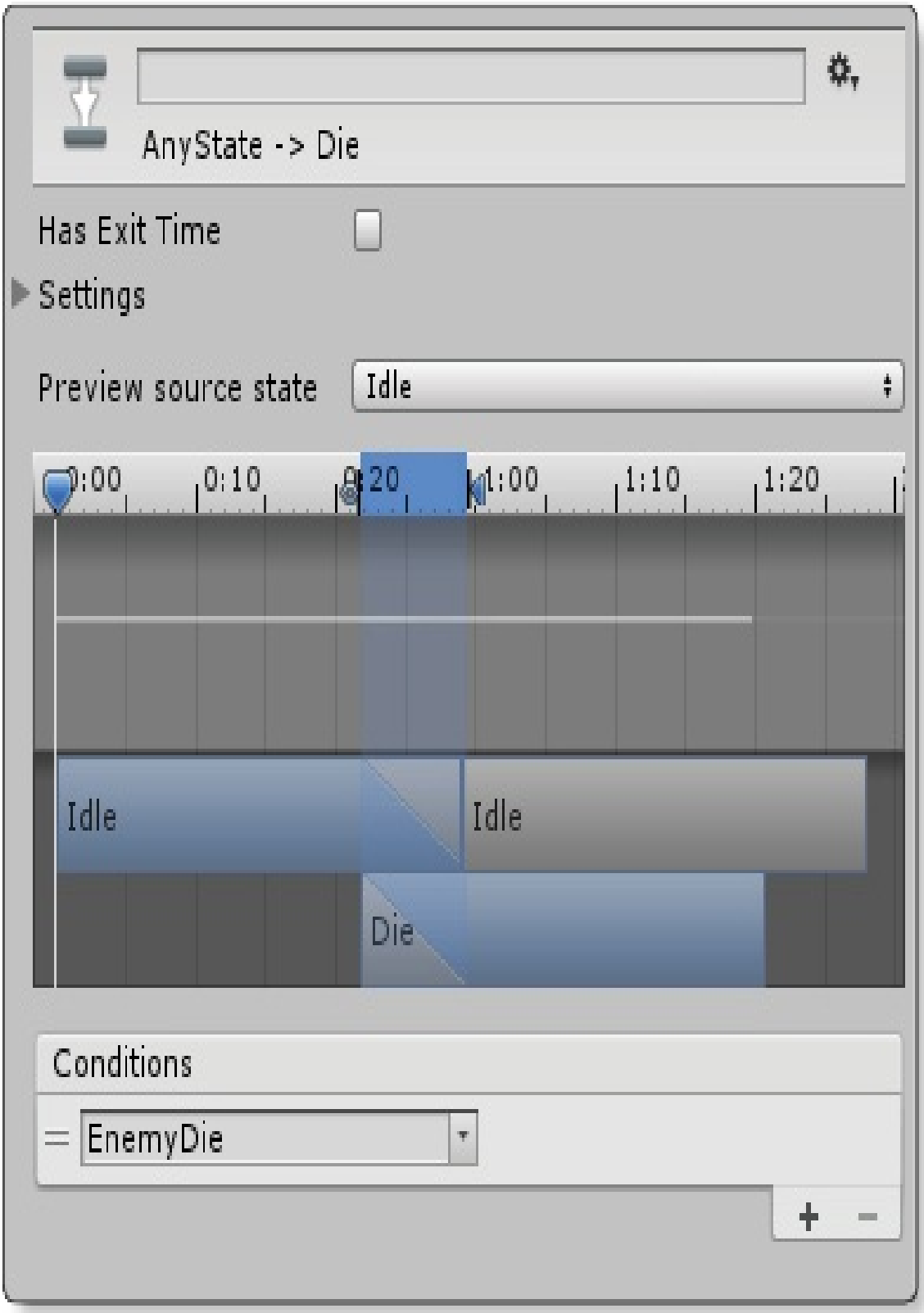
frame: 0325-0354: gethit from front.

En Animator coloque la animación, haga un parámetro tipo “Trigger”, nómbrelo

“EnemyDie y asígnelo en la transición.


Die
Attack
Any State
Idle
Entry

Ventana Animator

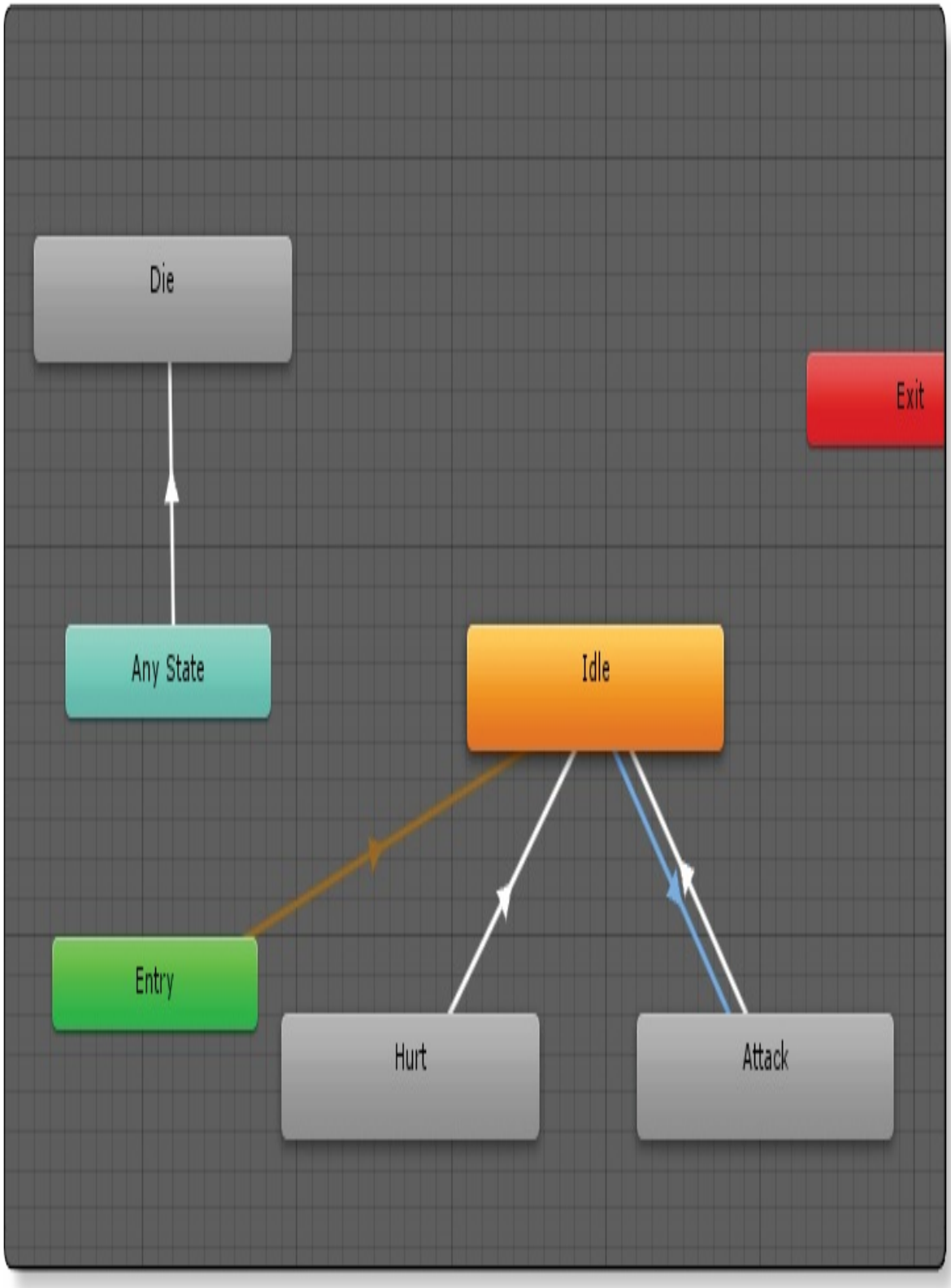

AnyState -> Die
Has Exit Time
Settings
Preview source state
Idle
0:00
0:10
0:20
1:00
1:10
1:20
Idle
Idle
Die
Conditions
EnemyDie

Parámetros de transición

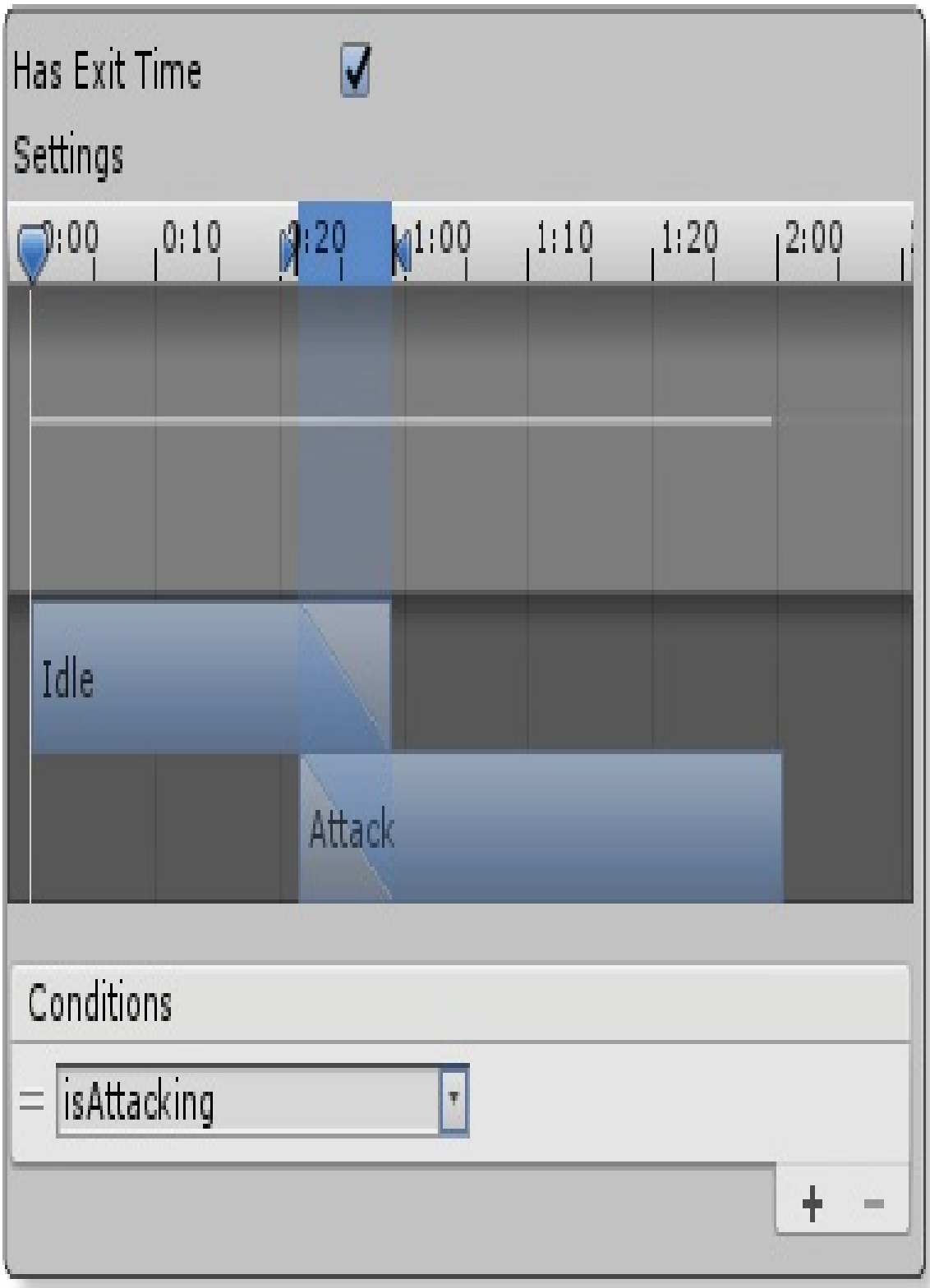
Asigne la animación “Hurt” con su transición y sin parámetro.

Reacomode las animaciones.

Haga un nuevo parámetro tipo “Trigger”, nómbrelo is Attacking y asígnelo a la transición seleccionada.


Die
Exit
Any State
Idle
Entry
Hurt
Attack

Reasignación de animaciones y parámetros


Has Exit Time
Settings
0:00
0:10
0:20
1:00
1:10
1:20
2:00
Idle
Attack
Conditions
= isAttacking
+
-

Parámetros de transición

Haga un nuevo script, nómbrelo Enemy02Health, asígnelo al enemigo.

Abra el script y escriba las siguientes variables y el Getter que se muestra.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class Enemy02Health : MonoBehaviour {

    [SerializeField] private int startingHealth = 20;
    [SerializeField] private float timeSinceLastHit = 0.5f;
    [SerializeField] private float dissapearSpeed = 2f;
    [SerializeField] private int currentHealth;


    private float timer = 0f;
    private Animator anim;
    private bool isAlive;
    private Rigidbody rigidbody;
    private CapsuleCollider capsuleCollider;
    private bool dissapearEnemy = false;
    private AudioSource audio;
    public AudioClip hurtAudio;
    public AudioClip killAudio;


    public bool IsAlive {
        get {return isAlive; }
    }
```

Variables de script “Enemy02Health”

Inicialice las variables que se muestran e inicialice el “timer” en la función “Update”.

```
// Use this for initialization
void Start () {

    rigidbody = GetComponent <Rigidbody> ();
    capsuleCollider = GetComponent <CapsuleCollider> ();
    anim = GetComponent <Animator> ();
    isAlive = true;
    currentHealth = startingHealth;
    audio = GetComponent<AudioSource> ();
}

// Update is called once per frame
void Update () {

    timer += Time.deltaTime;
}
```

Inicialización de variables

Escriba el siguiente código para programar lo siguiente:

El daño que le hará el proyectil del Player al enemigo

La animación de “Hurt” cuando sea tocado por el proyectil del Player

La funcionalidad de la muerte del enemigo cuando su salud llegue a cero

```
    void OnTriggerEnter (Collider other) {

        if (timer >= timeSinceLastHit && !GameManager.instance.GameOver) {
            if (other.tag == "PlayerWeapon") {
                takeHit ();
                timer = 0f;
            }
        }
    }

    void takeHit () {

        if (currentHealth > 0) {
            anim.Play ("Hurt");
            currentHealth -= 10;
            audio.PlayOneShot (hurtAudio);
        }

        if (currentHealth <= 0) {
            isAlive = false;
            KillEnemy ();
        }
    }

    void KillEnemy () {

        capsuleCollider.enabled = false;
        anim.SetTrigger ("EnemyDie");
        rigidbody.isKinematic = true;
        audio.PlayOneShot (killAudio);
    }

}
```

Código para los daños

Vaya al script “ArcherAttack” y en la función Update de ArcherAttack edite la siguiente línea de código

```
void Update () {

    if (Vector3.Distance (transform.position, player
        playerInRange = true;
        anim.SetTrigger("isAttacking");
    } else {
        playerInRange = false;
    }
}
```

Arranque de la animación

Previsualice el juego para observar las animaciones de Hurt y de muerte del enemigo.

Previsualización del daño

Previsualización de la muerte

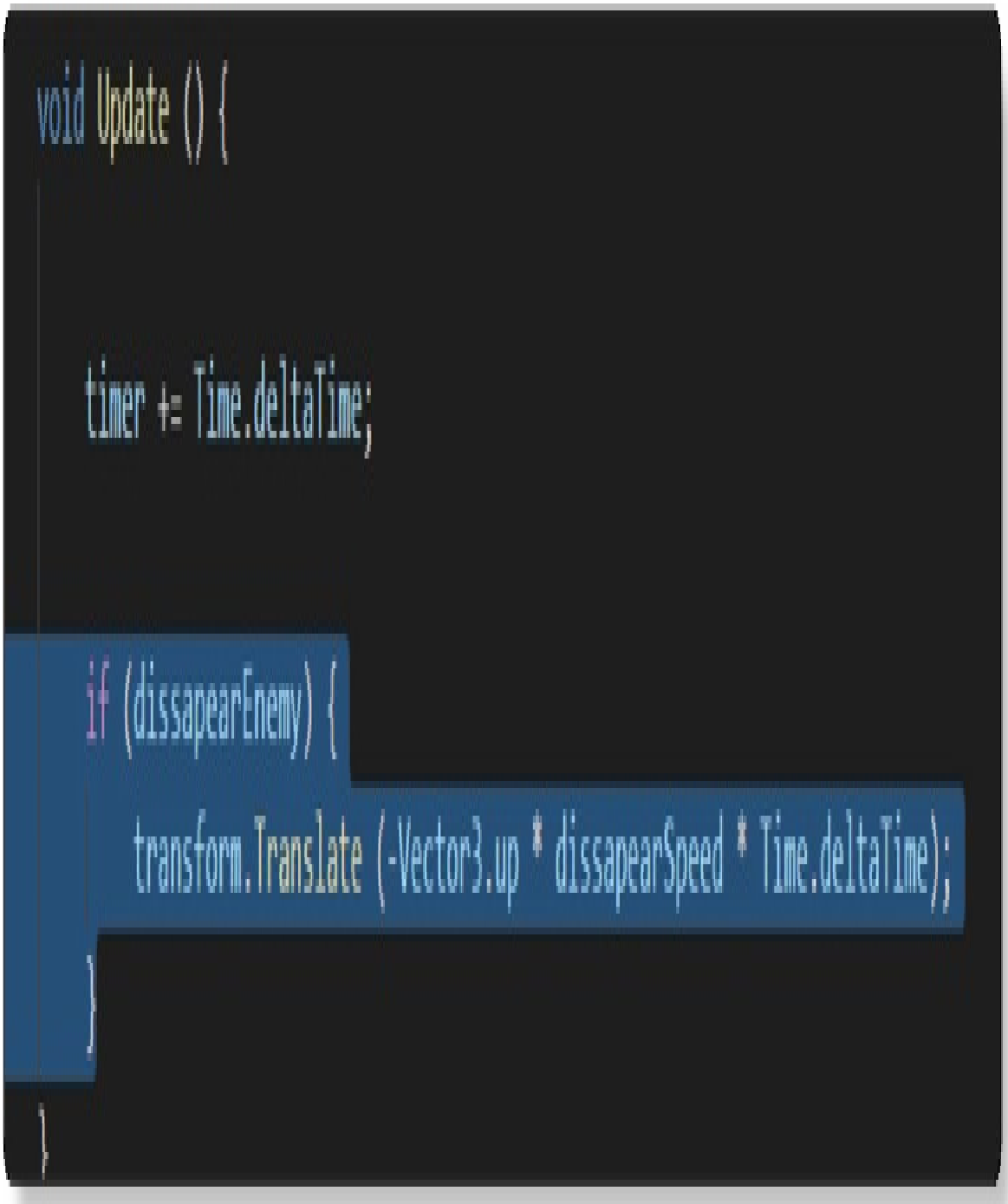
A continuación, queremos que el personaje desaparezca de la misma manera en la que desaparece en primer enemigo.

Vaya al script Enemy02Healh y escriba el siguiente código para esperar un tiempo para asignar el valor a la variable “dissapearEnemy” y otro tiempo para que el objeto se destruya.

```
void KillEnemy () {

    capsuleCollider.enabled = false;
    anim.SetTrigger ("EnemyDie");
    rigidbody.isKinematic = true;
    audio.PlayOneShot (killAudio);

    StartCoroutine (removeEnemy ());
}

IEnumerator removeEnemy() {

    yield return new WaitForSeconds (2f);
    dissapearEnemy = true;
    yield return new WaitForSeconds (1f);
    Destroy (gameObject);
}
```

IEnumerator removeEnemy

Vaya a la función Update y escriba el siguiente código para hundir al enemigo cuando la variable “dissapearEnemy” sea verdadera.


void Update () {
timer += Time.deltaTime;
if (dissapearEnemy) {
transform.Translate (-Vector3.up * dissapearSpeed * Time.deltaTime);
}
}

Código para hundir al enemigo

Previsualice el juego para observar como se hunde y como se destruye el enemigo.

Haga un Prefab de Enemy02.

RETO AL ESTUDIANTE

Con la finalidad de que el estudiante pueda repasar lo visto en este capítulo y en el previo, deberá de crear un nuevo enemigo con el personaje brindado en la carpeta “resources” referido a “Orc_Ranger”.

A este nuevo enemigo deberá de programar las siguientes funcionalidades.

Debe de estar en su modo Idle al inicio

Cuando el Player se encuentre a cierto rango que el enemigo comience a caminar hacia él

Cuando el Player se encuentre más cerca, el enemigo deberá de disparar un proyectil

Programar los daños necesarios para el Player y el enemigo

Asignar sonidos a las animaciones de ataque, daño y muerte del enemigo

ITEMS

INTRODUCCIÓN

En este capítulo trabajaremos con la creación de dos tipos de Items, uno tipo Sprite 3D y otro basado en un modelo 3D.

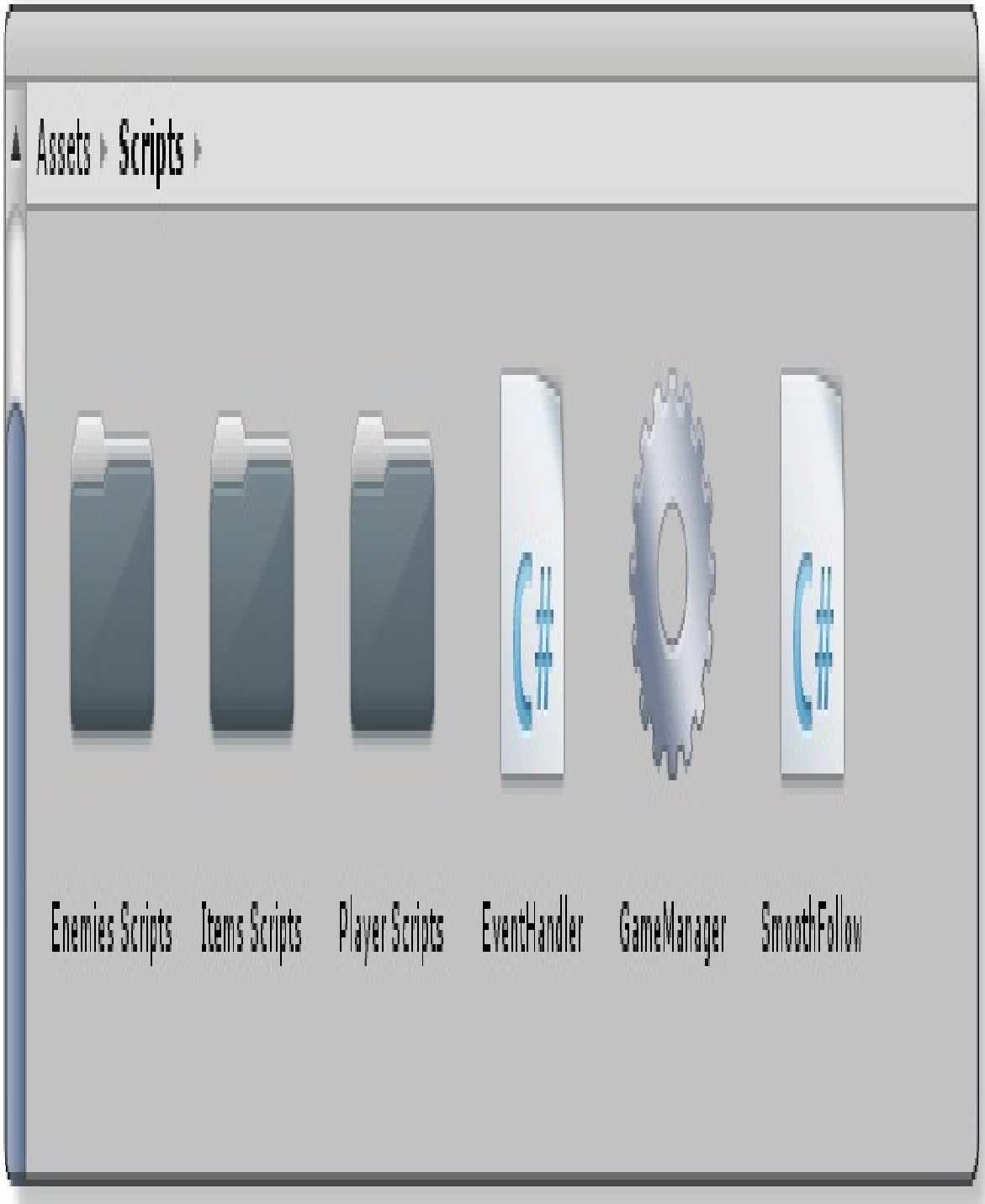
Programaremos el primero con la funcionalidad de añadir salud a nuestro Player y el segundo con la funcionalidad de invisibilidad.

El estudiante podrá poner en práctica lo aprendido previamente y se abarcarán temas nuevos como sistemas de partículas, efectos desde una librería externa, matrices entre otros.

CREACIÓN DE UN ITEM 2D DE SALUD

En este apartado crearemos y programaremos un “ítem” tipo 2D que le añada puntos de salud al “Player” cuando sea obtenido.

Para reorganizar los scripts de manera funcional, vaya al folder de scripts, haga las carpetas que se muestran y coloque dentro los scripts correspondientes a cada carpeta.


Assets ▸ Scripts ▸
Enemies Scripts
Items Scripts
Player Scripts
EventHandler
GameManager
SmoothFollow

Organización de los scripts

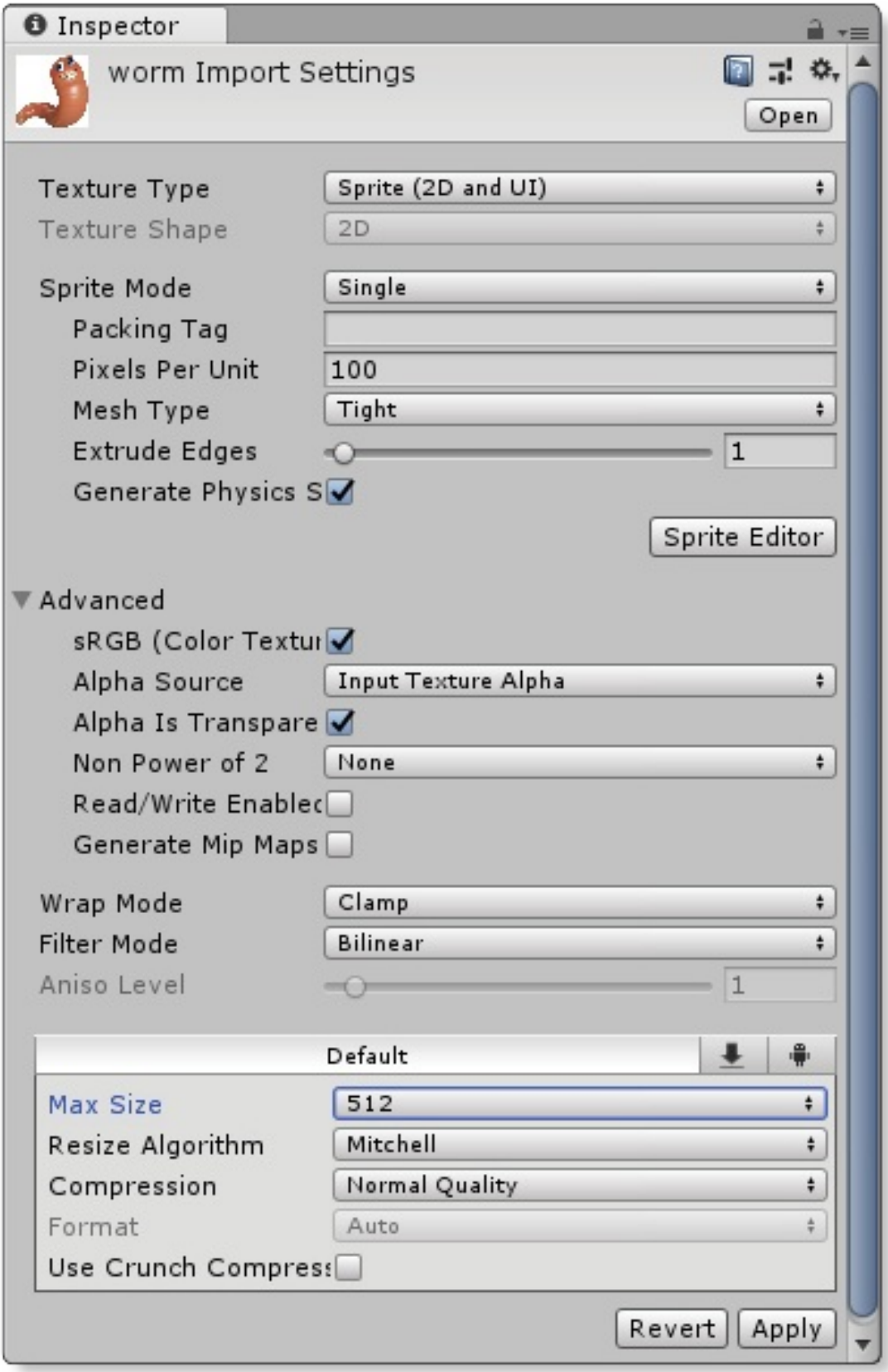
Haga un nuevo folder en “Assets” y nómbrelo “Sprites”.

Importe dentro del folder “Sprites” la imagen “worm” que se encuentra en la carpeta de “resources” de la lección.

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
Audio
Materials
Models
Prefabs
Scenes
Scripts
Sprites
Textures
Assets ▸ Sprites
worm

Imagen 2D

Convierta la imagen en textura tipo “Sprite” con los valores que se muestran.

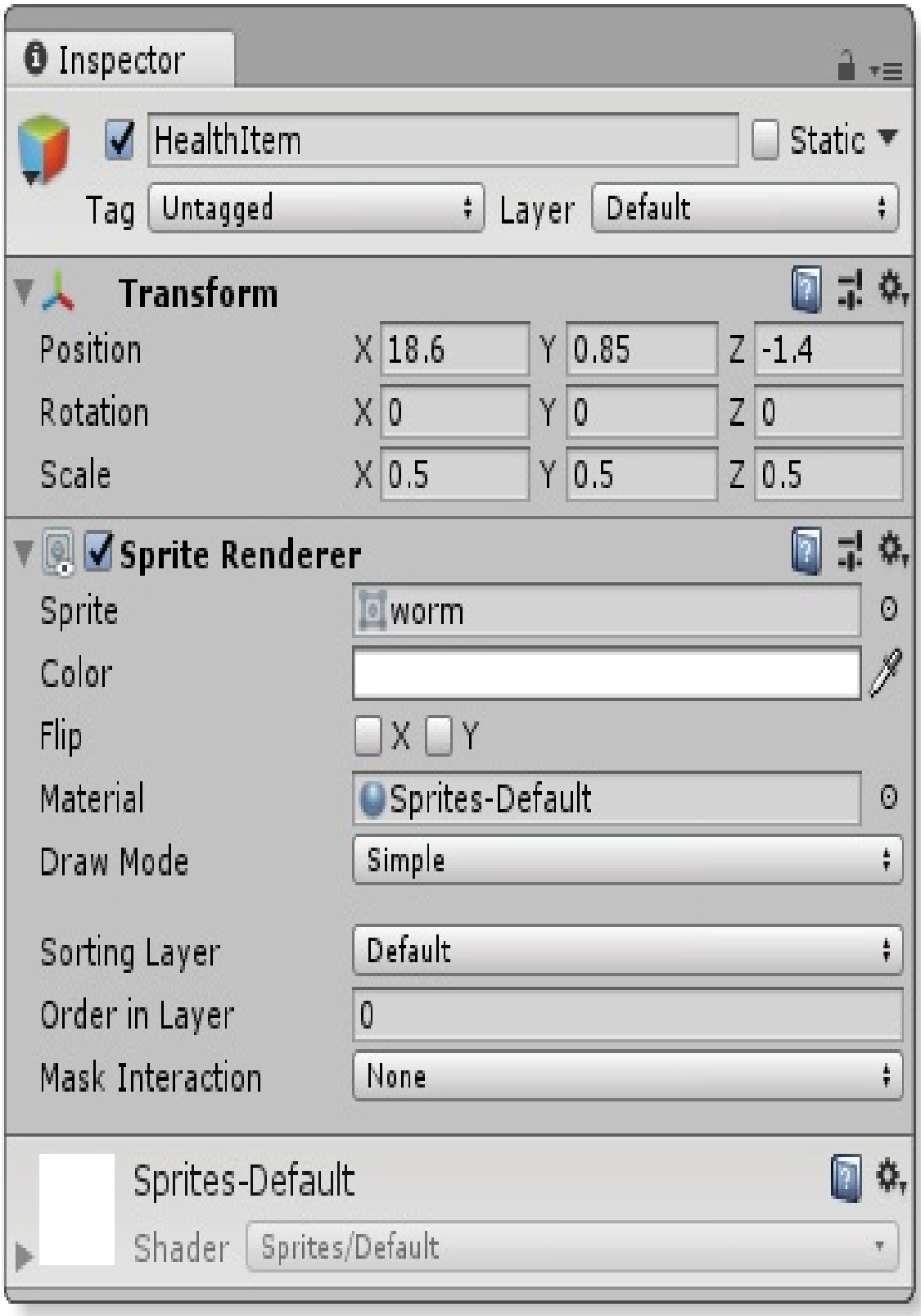

Inspector
worm Import Settings
Open
Texture Type
Sprite (2D and UI)
Texture Shape
2D
Sprite Mode
Single
Packing Tag
Pixels Per Unit
100
Mesh Type
Tight
Extrude Edges
1
Generate Physics S
Sprite Editor
Advanced
sRGB (Color Textur
Alpha Source
Input Texture Alpha
Alpha Is Transpare
Non Power of 2
None
Read/Write Enablec
Generate Mip Maps
Wrap Mode
Clamp
Filter Mode
Bilinear
Aniso Level
1
Default
Max Size
512
Resize Algorithm
Mitchell
Compression
Normal Quality
Format
Auto
Use Crunch Compress
Revert
Apply

Valores para Textura tipo Sprite 2D

Al tratarse de una “sprite” que nos va a abarcar una pequeña parte de la pantalla, podemos asignar un tamaño máximo (Max Size) de 512 sin tener que sacrificar calidad.

Haga un nuevo “EmptyObjet” y nómbrelo “HealthItem”.

Añada un componente “SpriteRenderer” y asigne como imagen el sprite “Worm” como se muestra y colóquelo en la escena como se indica.


Inspector
HealthItem
Static
Tag
Untagged
Layer
Default
Transform
Position
X 18.6
Y 0.85
Z -1.4
Rotation
X 0
Y 0
Z 0
Scale
X 0.5
Y 0.5
Z 0.5
Sprite Renderer
Sprite
worm
Color
Flip
X
Y
Material
Sprites-Default
Draw Mode
Simple
Sorting Layer
Default
Order in Layer
0
Mask Interaction
None
Sprites-Default
Shader
Sprites/Default

Componente SpriteRenderer

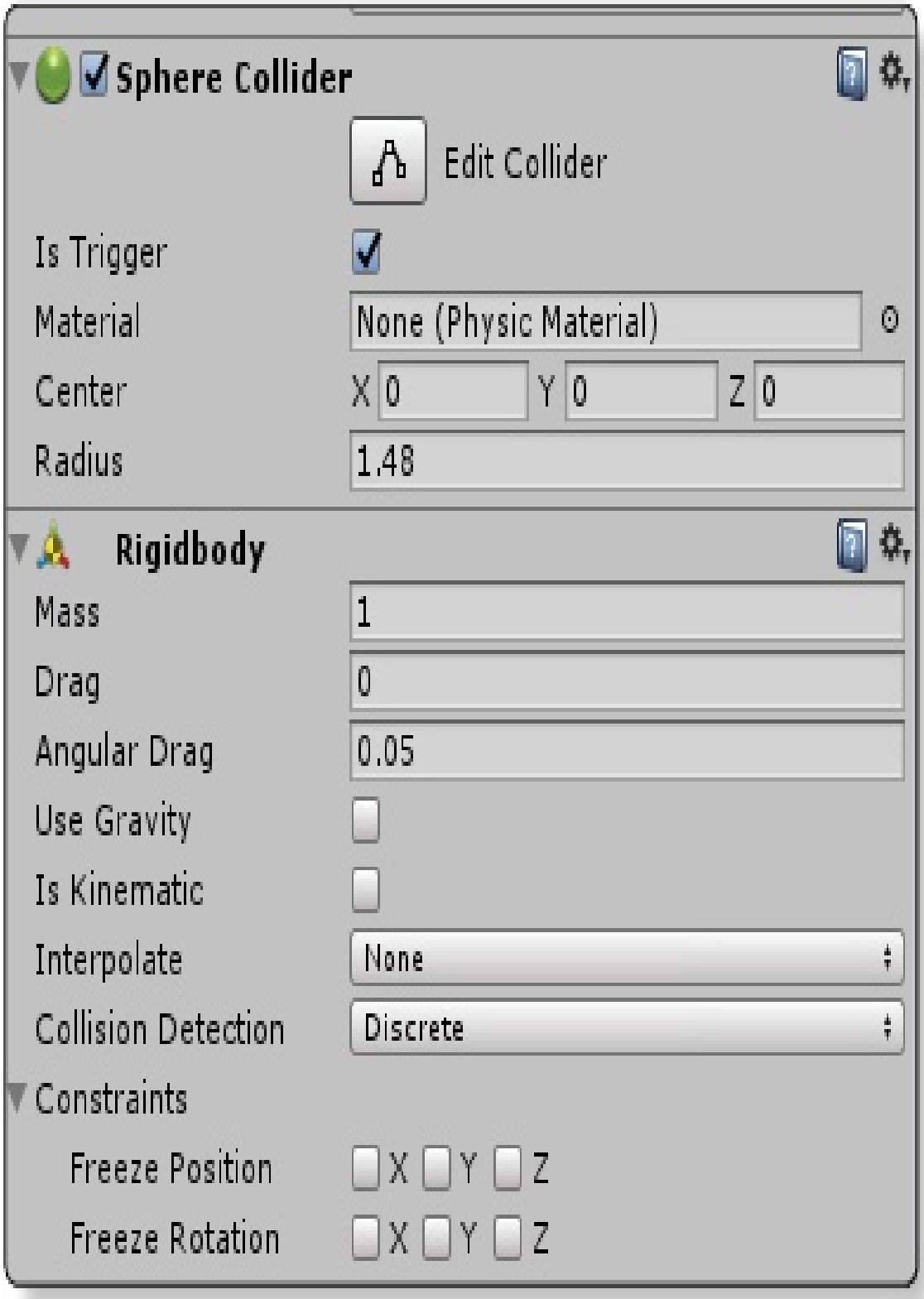
NOTA:

La escala la asignamos a 0.5 en los tres vectores

KnifeSpawn

Previsualización de item

Añada a “HealthItem” un “SphereCollder” y un “RigidBody” con los valores que se muestran.


Sphere Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 0
Z 0
Radius
1.48
Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Z
Freeze Rotation
X
Y
Z

Componentes de HealthItem

SphereCollider

Haga un nuevo script, nómbrelo “HealthItemScript” y asígnelo al objeto “HealthItem”.

Abra el script, escriba las siguientes variables e inicialícelas como se muestra.

```
public class HealthItemScript : MonoBehaviour {

    private GameObject player;
    private PlayerHealth playerHealth;

    // Use this for initialization
    void Start () {

        player = GameManager.instance.Player;
        playerHealth = player.GetComponent<PlayerHealth> ();

    }
```

Variables de script HealthItemScript

Queremos mandar llamar un elemento del script “PlayerHealth”, por ello creamos la variable del tipo objeto “PlayerHealth” y la inicializamos indicando que vamos a tomar un elemento existente en el objeto “Player” dentro de su componente “PlayerHealth”.

Para ello vamos a ir al script “PlayerHealth” y a crear un “Getter” y un “Setter” para las variables que necesitamos compartir sus valores

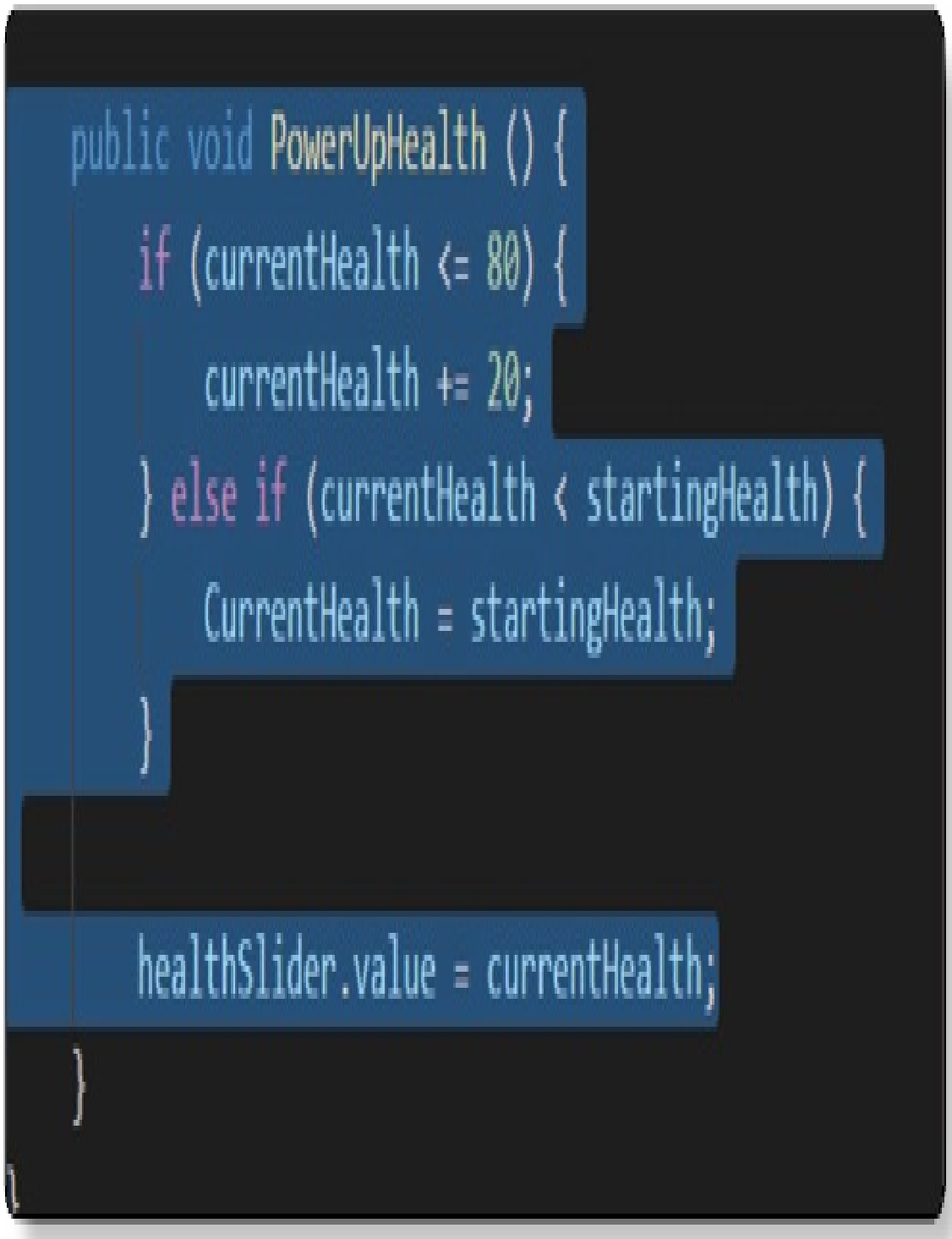
Vaya al script “PlayerHealth” y haga el siguiente “Getter” y “Setter” para la variable “CurrentHealth”.

```
    public AudioClip hurtAudio;
    public AudioClip deadAudio;


    public int CurrentHealth {
        get { return currentHealth; }
        set {
            if (value < 0)
                currentHealth = 0;
            else
                currentHealth = value;
        }
    }


    // Use this for initialization
    void Start () {
```

Getter & Setter

En donde si el valor actual es menor que cero, la variable “currentHealth” será igual a cero, si esta condición no se cumple, entonces el valor de “currentHealth” será igual al valor actual.

En el mismo script (“PlayerHealth”) escribimos la siguiente función que se correrá una vez que el “Player” haga recogido el “ítem”..



```
    public void PowerUpHealth () {
        if (currentHealth <= 80) {
            currentHealth += 20;
        } else if (currentHealth < startingHealth) {
            CurrentHealth = startingHealth;
        }

        healthSlider.value = currentHealth;
    }
}
```

Función PowerUpHealth

En donde si el valor de “currentHealth” es menor o igual a 80, entonces se sumarán 20 puntos a la salud, si no se cumple esta condición, entonces si el valor de “currentHealth” es menor que “startingHealth” (100), (pero mayor que 80 por la condición anterior), entonces el valor de “CurrentHealth” será igual al valor de “startingHealth” (100).

Y finalizamos asignando el nuevo valor de “currentHealth” al objeto “Slider” de la salud.

Vaya al script “HealthItem” y escriba la siguiente función para el momento en el que colinde con el objeto “Player”.

```
// Update is called once per frame
void Update () {

}

void OnTriggerEnter (Collider other) {
    if (other.gameObject == player) {
        playerHealth.PowerUpHealth ();
        Destroy (gameObject);
    }
}
```

Función OnTriggerEnter

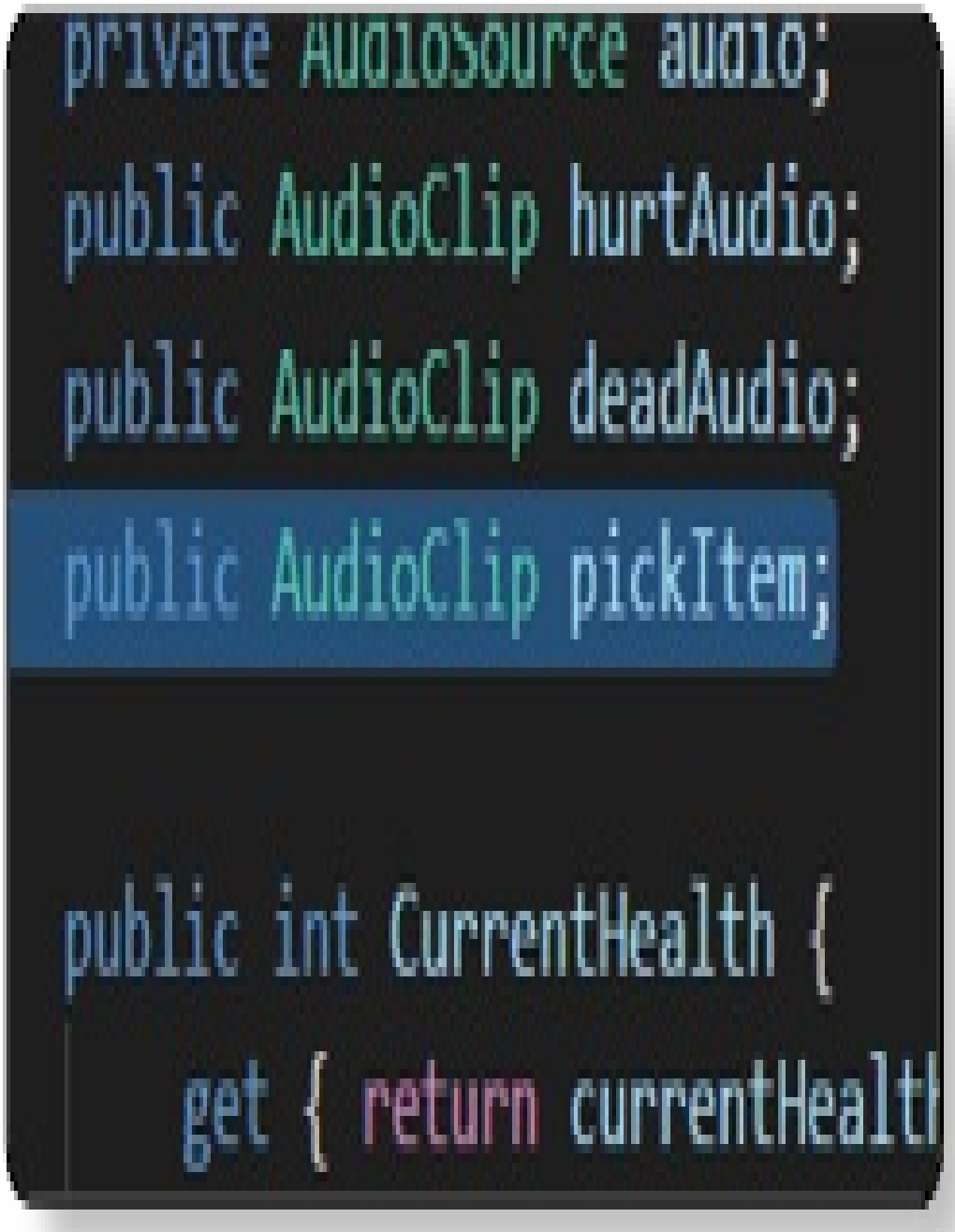
Donde se mandará llamar la función “PowerUpHealth” creada en el script “PlayerHealth” y en seguida se destruirá el objeto.

Previsualice el juego para comprobar el funcionamiento del “ítem”.

A continuación, se crearán los elementos necesarios para añadir un clip de audio para el momento en el que se toma el “ítem”.

En la carpeta de “Audio” importe el sonido “healthsound” de la carpeta de “resources”.

Vaya al script de “PlayerHealth”, haga la siguiente variable y asígnela como se muestra en la función “PowerUpHealth”.


private AudioSource audio;
public AudioClip hurtAudio;
public AudioClip deadAudio;
public AudioClip pickItem;
public int CurrentHealth {
get { return currentHealt

Variable tipo AudioClip

```
public void PowerUpHealth () {
    if (currentHealth <= 80) {
        currentHealth += 20;
    } else if (currentHealth < startingHealth) {
        CurrentHealth = startingHealth;
    }

    healthSlider.value = currentHealth;
    audio.PlayOneShot (pickItem);
}
```

Reproducción de audio

Vaya a Unity y asigne el elemento correspondiente en el componente del script.

Player Health (Script)
Script
PlayerHealth
Starting Health
100
Time Since Last Hit
2
Current Health
50
Health Slider
Slider (Slider)
Hurt Audio
15zombie_explode
Dead Audio
Hisses Dry 11.M
Pick Item
healthSound

Asignación de audio

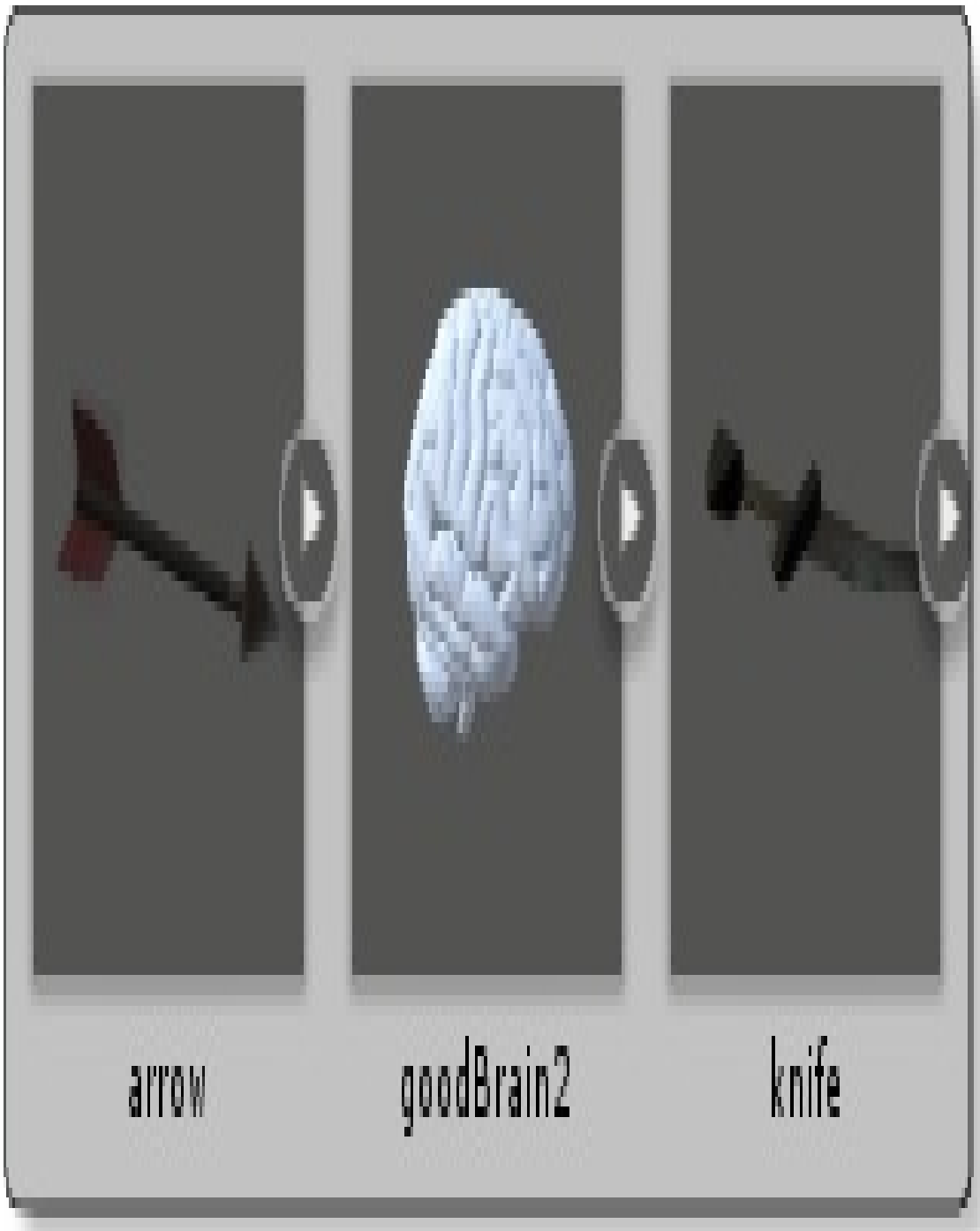
Previsualice el juego para observar el sonido correspondiente.

Convierta el ítem en “Prefab” y bórrelo de la escena.

CREACIÓN DE UN ITEM 3D DE INVENCIBILIDAD

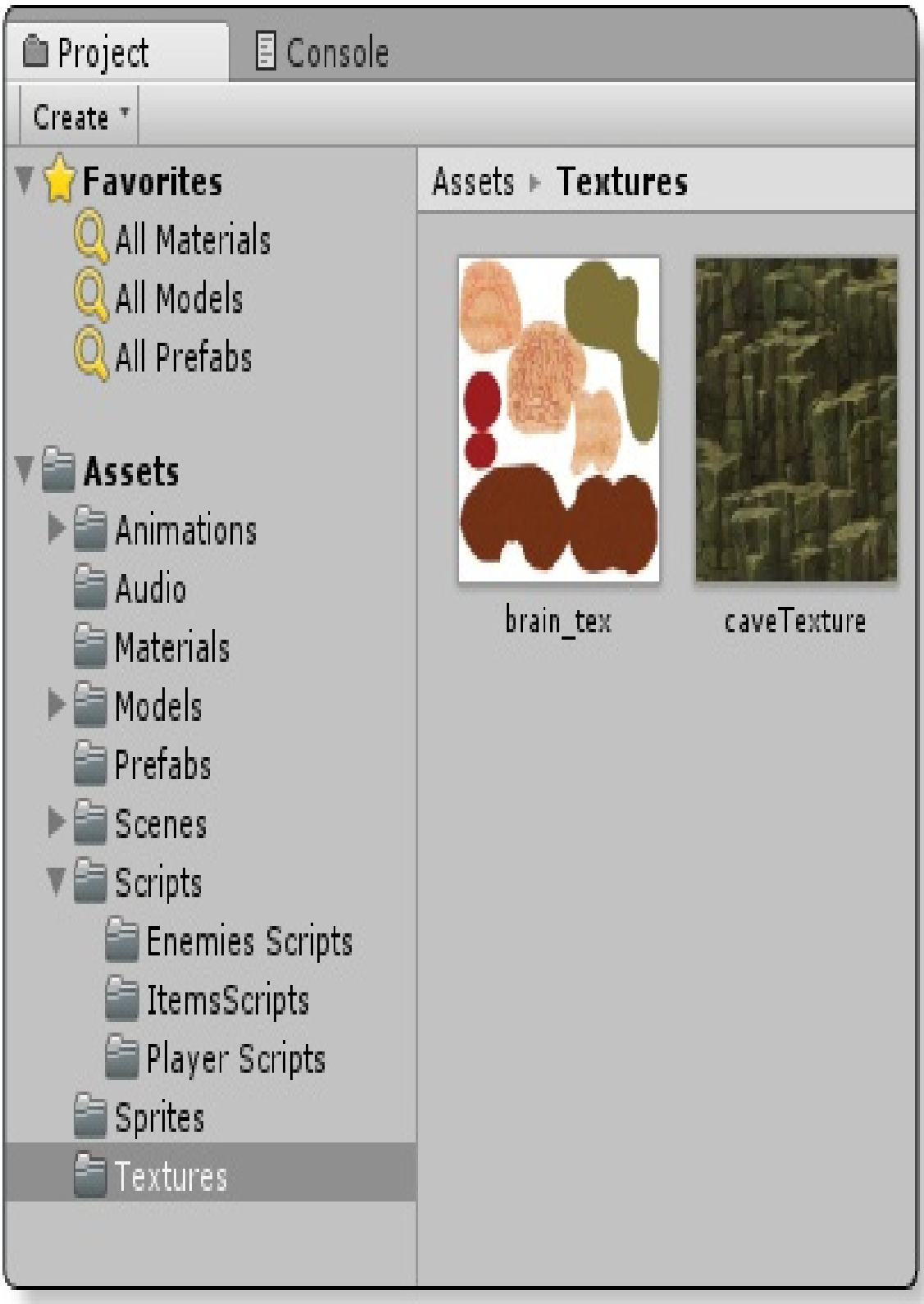
En este apartado crearemos y programaremos un “ítem” tipo 3D que haga invencible al “Player” durante una fracción de tiempo cuando sea obtenido, veremos la funcionalidad de un sistema de partículas y programaremos la funcionalidad del Item conforme a la invencibilidad del Player.

Importe de la carpeta de “resources” el modelo “goodBrain2” a la carpeta de “Models” de Unity”.


arrow
goodBrain2
knife

Modelo 3D

Importe la textura correspondiente al folder “Textures” de Unity.

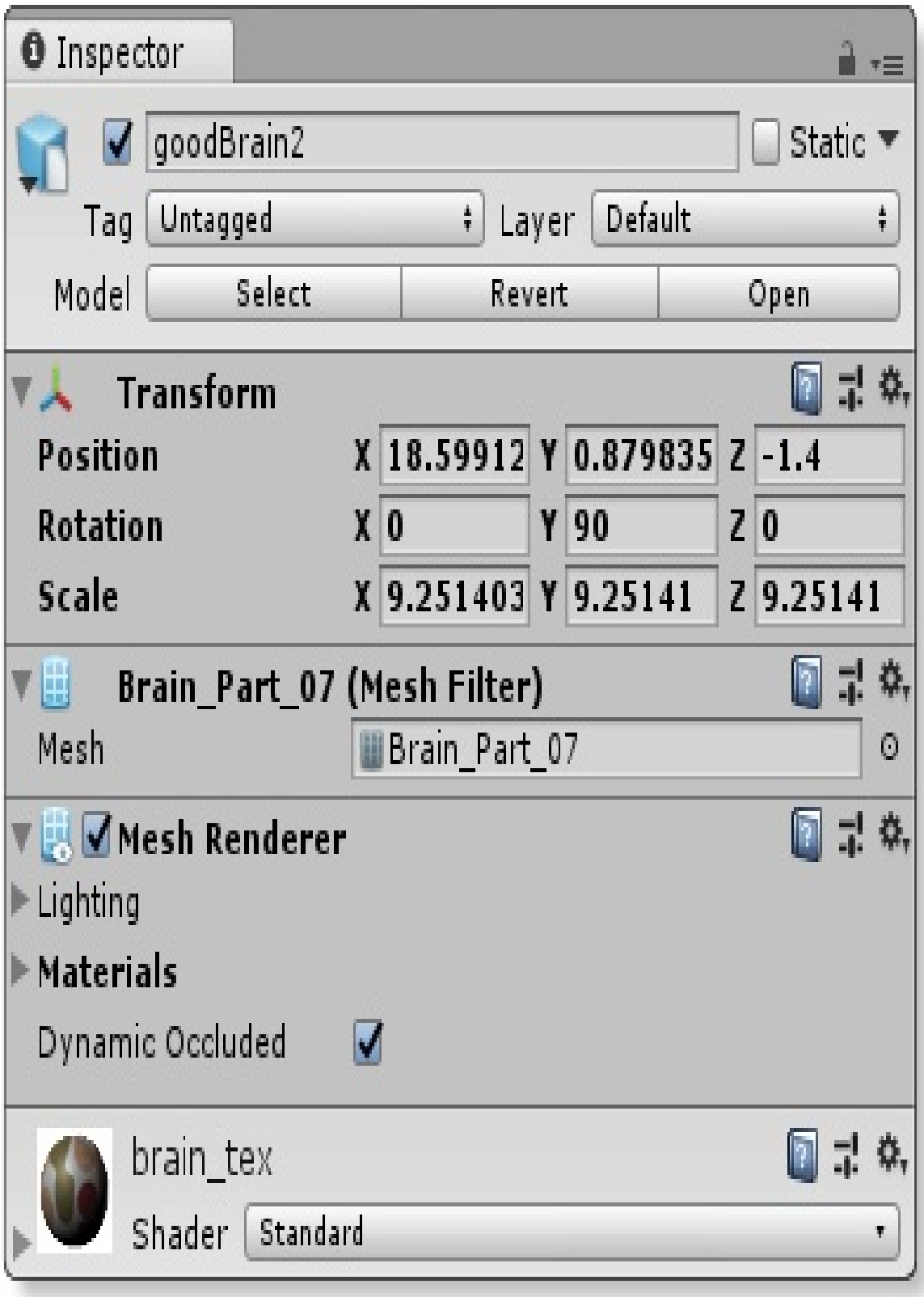

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
Audio
Materials
Models
Prefabs
Scenes
Scripts
Enemies Scripts
ItemsScripts
Player Scripts
Sprites
Textures
Assets ▸ Textures
brain_tex
caveTexture

Textura del modelo 3D

Coloque el modelo en la escena, asigne los valores que se muestran de transformación y texturice.

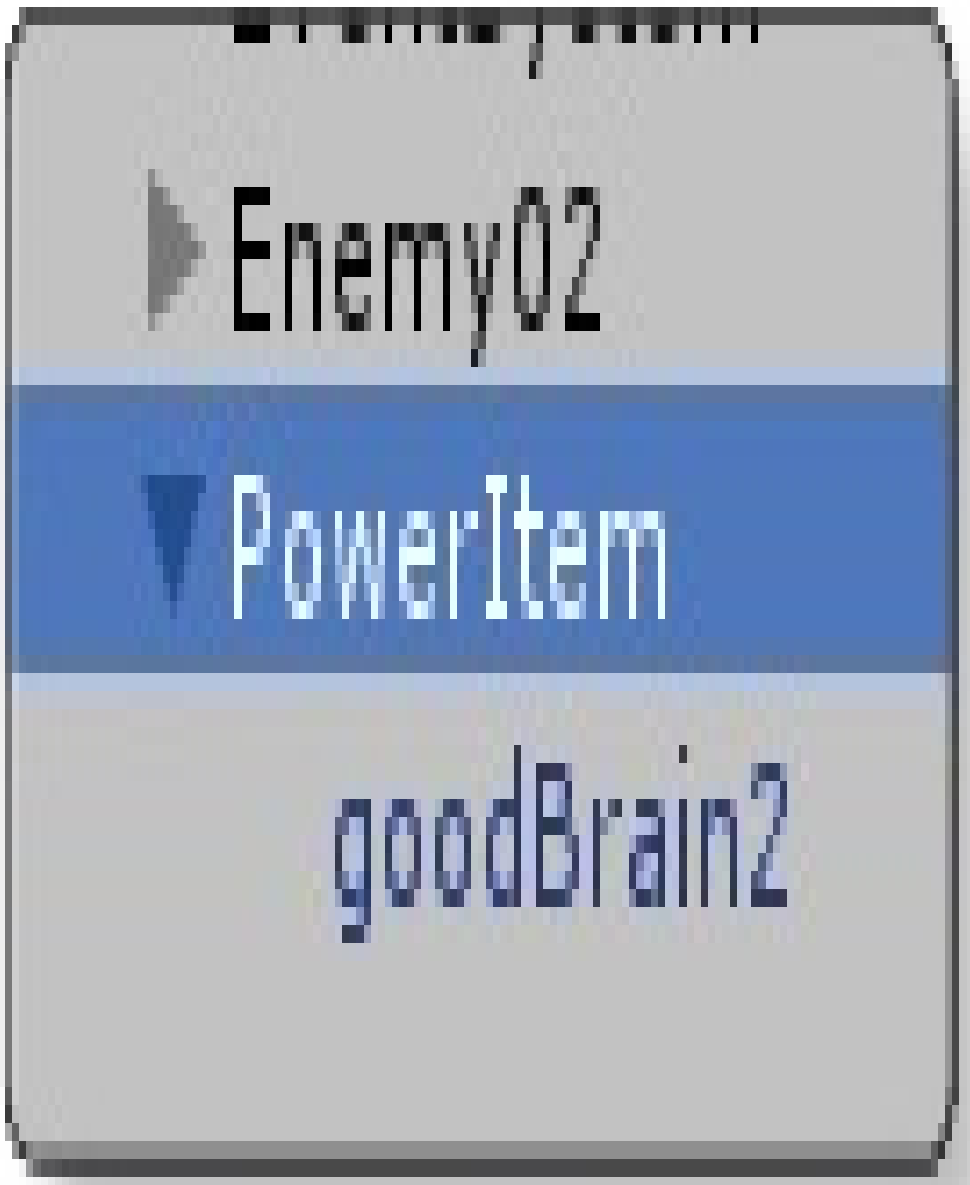
KnifeSpawn

Modelo texturizado

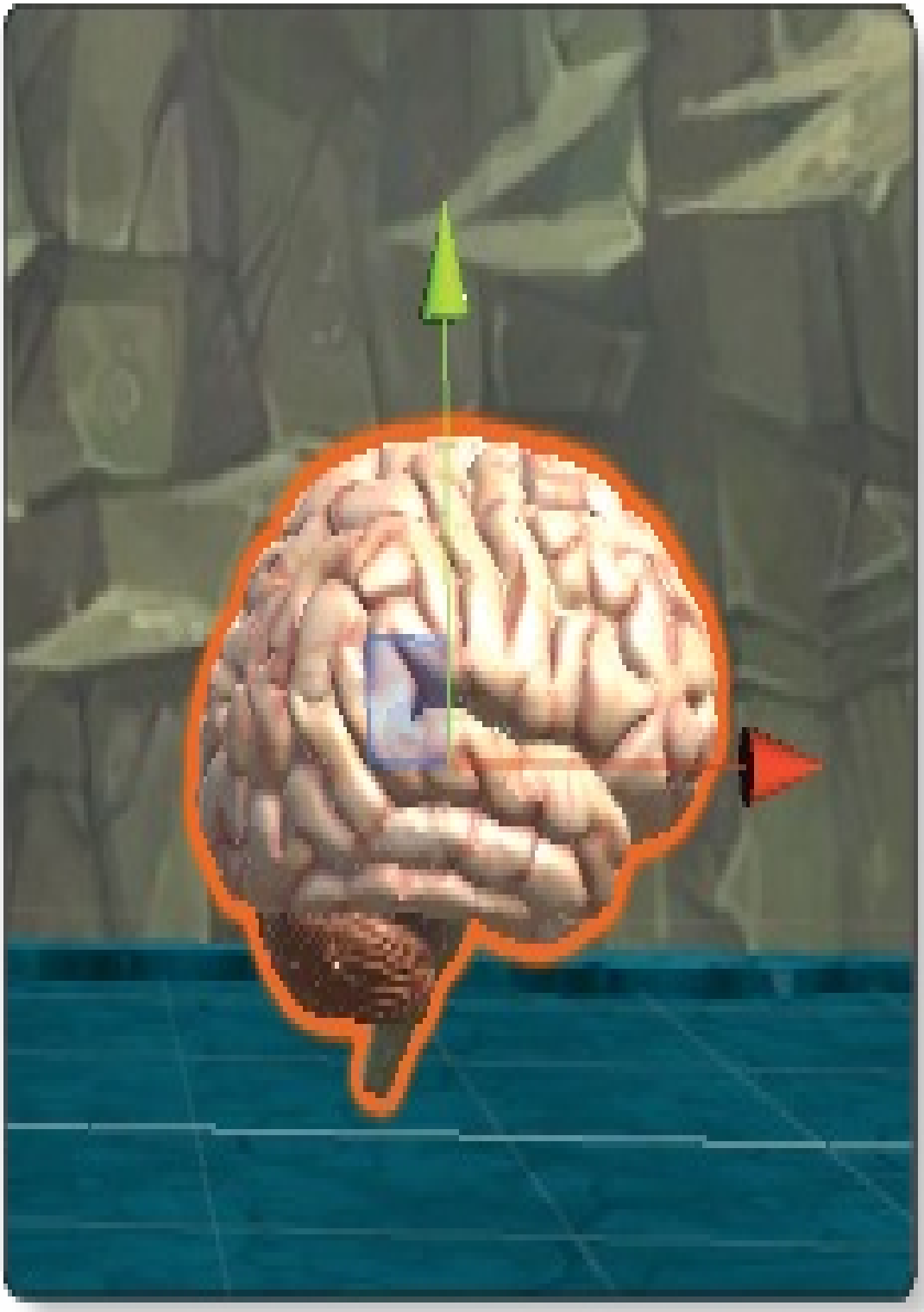

Inspector
goodBrain2
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 18.59912
Y 0.879835
Z -1.4
Rotation
X 0
Y 90
Z 0
Scale
X 9.251403
Y 9.25141
Z 9.25141
Brain_Part_07 (Mesh Filter)
Mesh
Brain_Part_07
Mesh Renderer
Lighting
Materials
Dynamic Occluded
brain_tex
Shader
Standard

Valores del modelo 3D

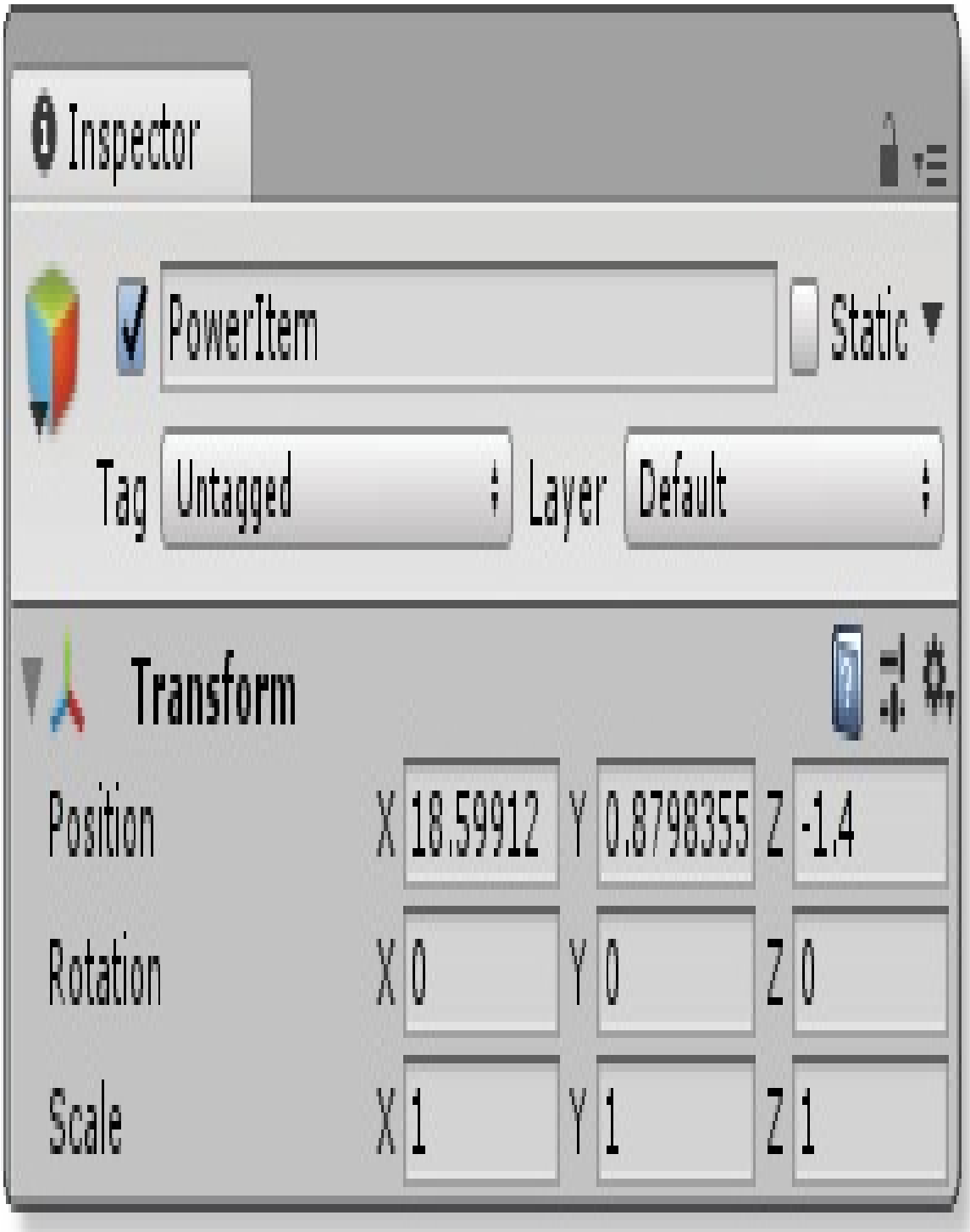
Haga un nuevo “EmptyObject”, renómbrelo “PowerItem”, colóquelo en relación al cerebro y emparéjelo.


Enemy02
PowerItem
goodBrain2

Emparentamiento con objeto “PowerItem”



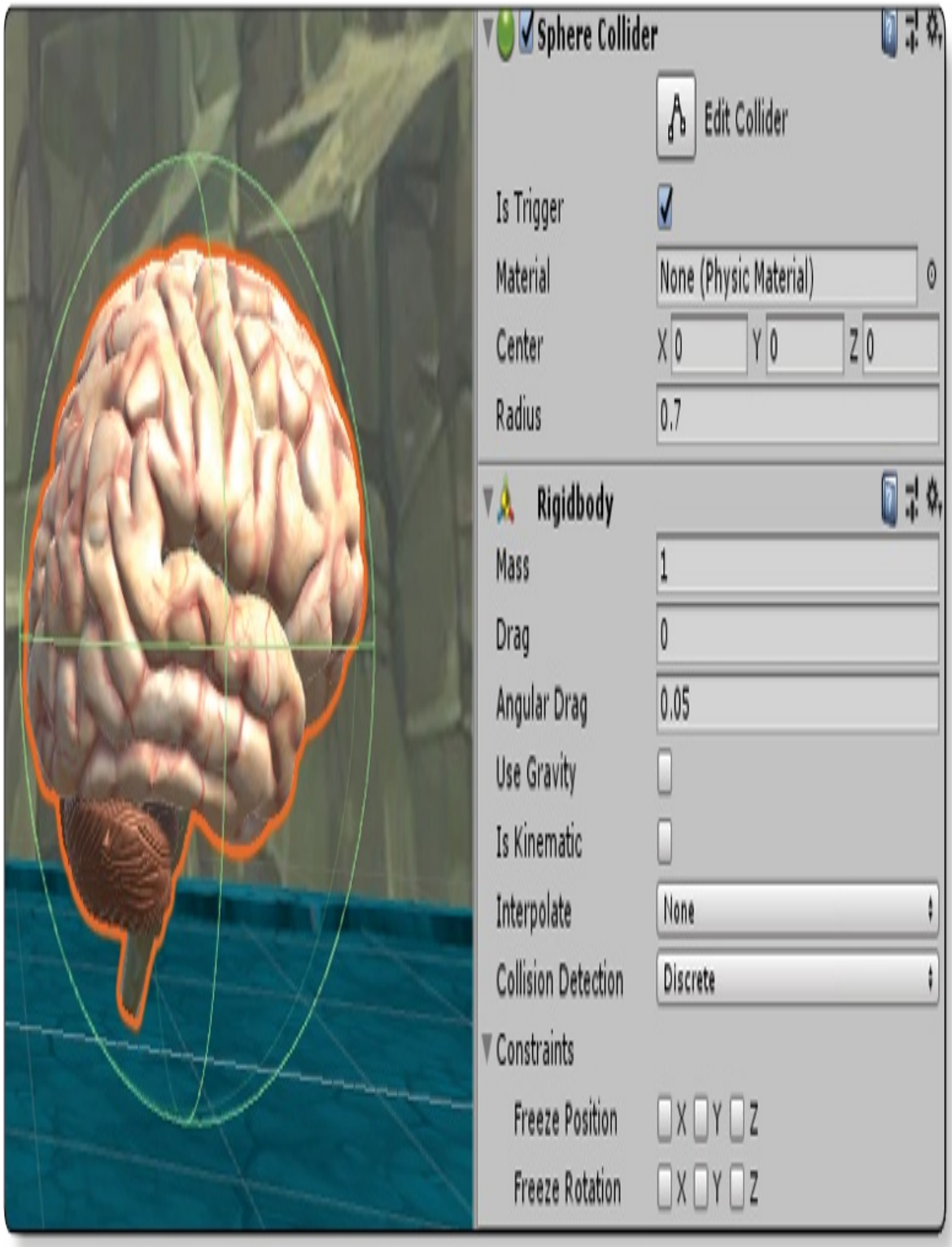
Colocación de objeto PowerItem


Inspector
PowerItem
Static
Tag
Untagged
Layer
Default
Transform
Position
X 18.59912
Y 0.8798355
Z -1.4
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1

Valores de transformación de PowerItem

Haga su componente “SphereCollider” con la casilla “IsTrigger” activa.

Haga su componente “RigidBody” sin gravedad activa.

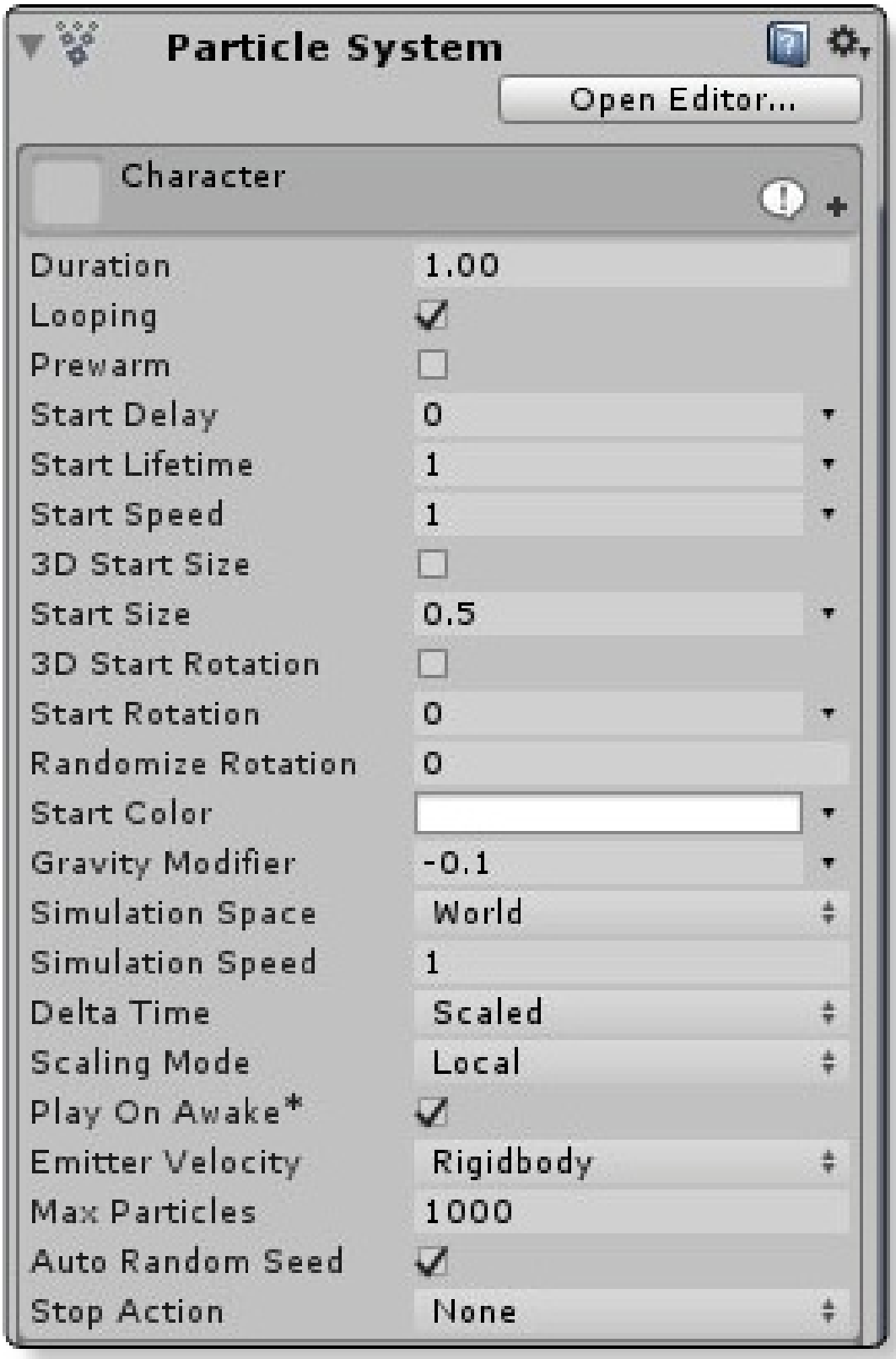

Sphere Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 0
Z 0
Radius
0.7
Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X Y Z
Freeze Rotation
X Y Z

Componentes de PowerItem

Haga un nuevo C# Script, nómbrelo PowerItemScript y asígnelo al objeto “PowerItem”.

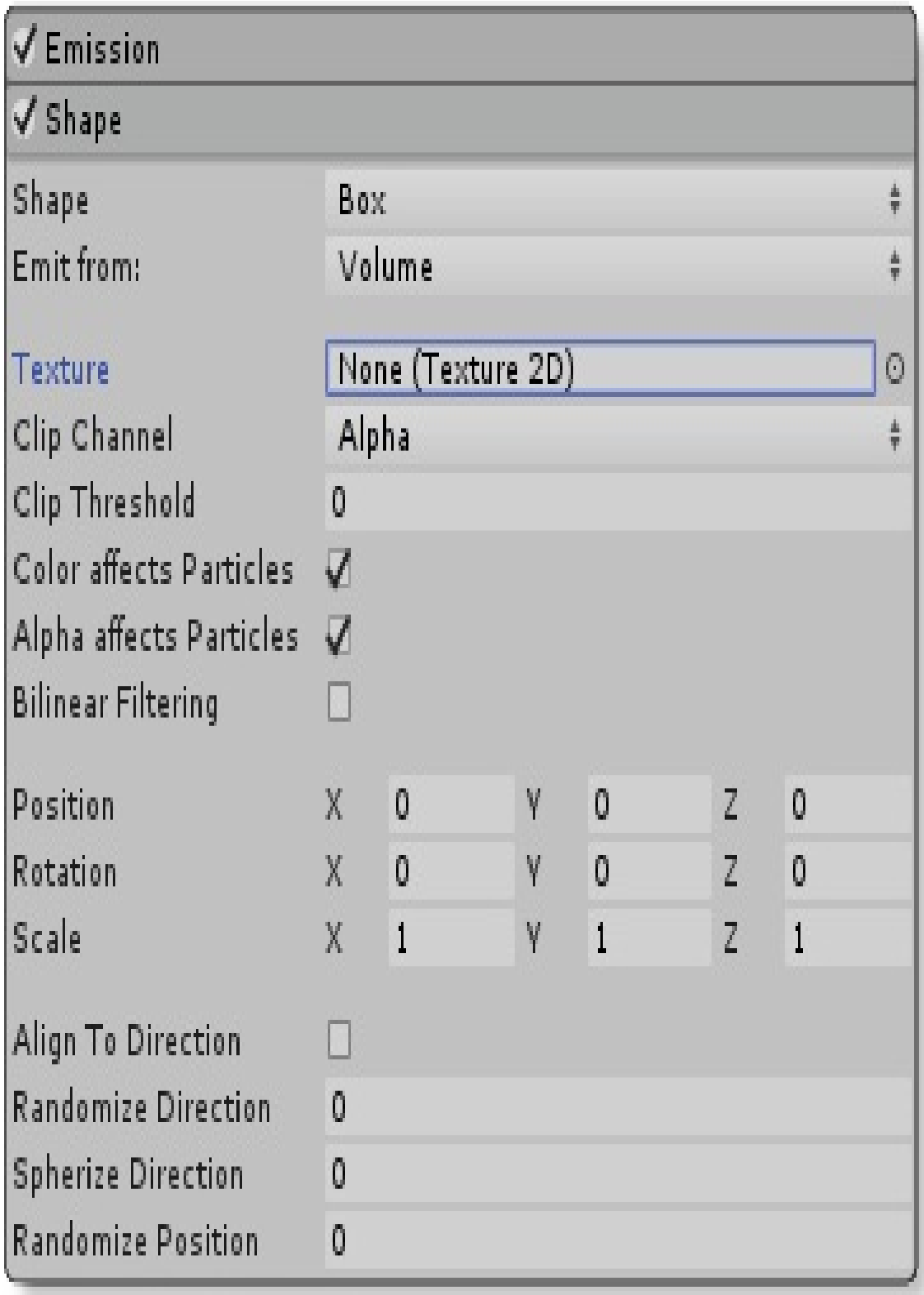
A continuación, vamos a asignar un sistema de partículas para el “PowerItem”.

Seleccione “PowerItem” y asigne un componente “Effects/ ParticleSystem” y añada los valores que se muestran.


Particle System
Open Editor...
Character
Duration
1.00
Looping
Prewarm
Start Delay
0
Start Lifetime
1
Start Speed
1
3D Start Size
Start Size
0.5
3D Start Rotation
Start Rotation
0
Randomize Rotation
0
Start Color
Gravity Modifier
-0.1
Simulation Space
World
Simulation Speed
1
Delta Time
Scaled
Scaling Mode
Local
Play On Awake*
Emitter Velocity
Rigidbody
Max Particles
1000
Auto Random Seed
Stop Action
None

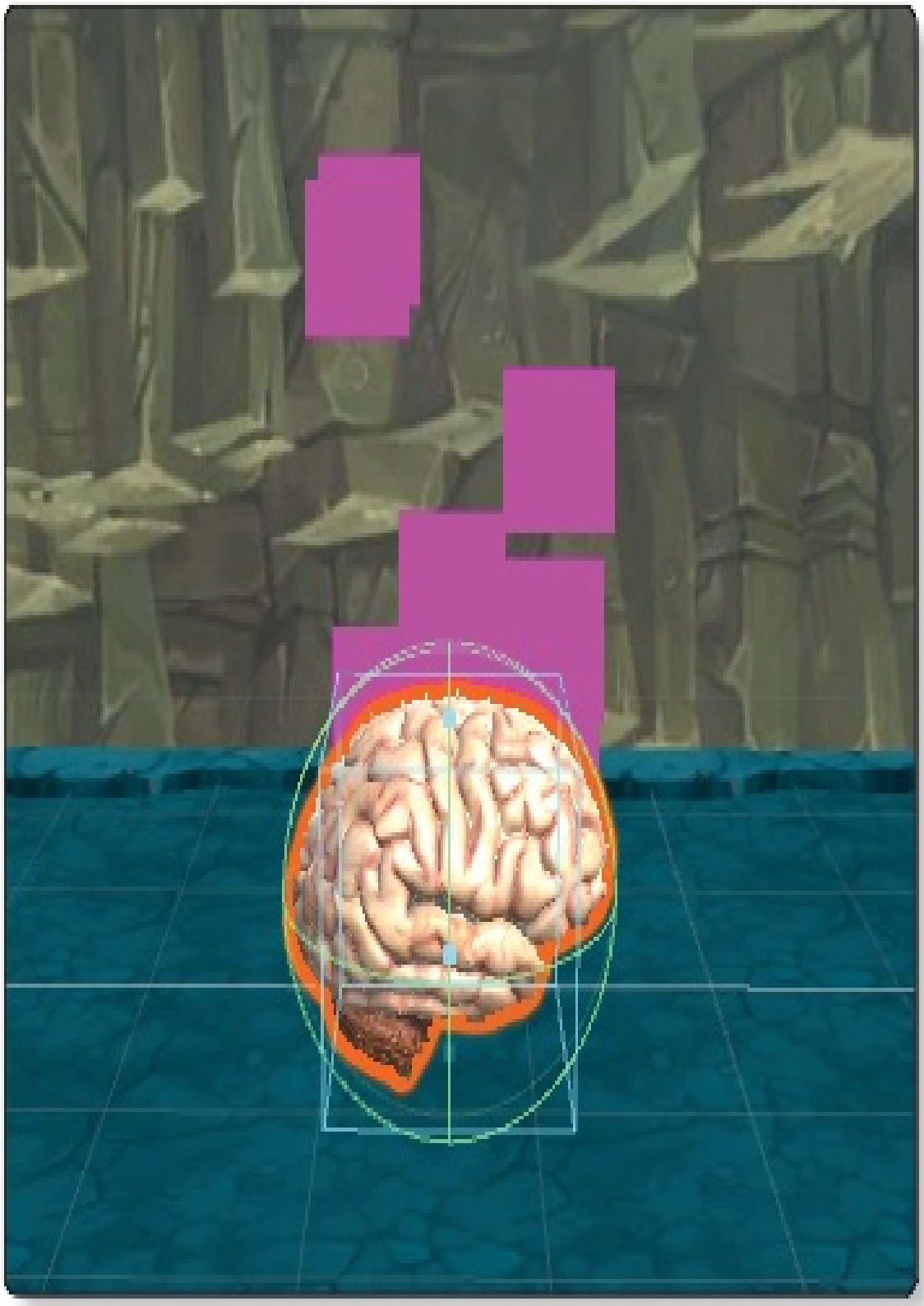
Pestaña Charater del sistema de partículas

La pestaña “Character” nos brinda control sobre aspectos de duración, repetición, vida, color, gravedad, simulación, velocidad, máximas partículas entre otros de las partículas.


Emission
Shape
Shape
Box
Emit from:
Volume
Texture
None (Texture 2D)
Clip Channel
Alpha
Clip Threshold
0
Color affects Particles
Alpha affects Particles
Bilinear Filtering
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Align To Direction
Randomize Direction
0
Spherize Direction
0
Randomize Position
0

Pestaña Shape del sistema de partículas

La pestaña “Shape” nos brinda control sobre el tipo de emisor, la textura (que todavía no creamos), canales de transformación entre otras de las partículas.



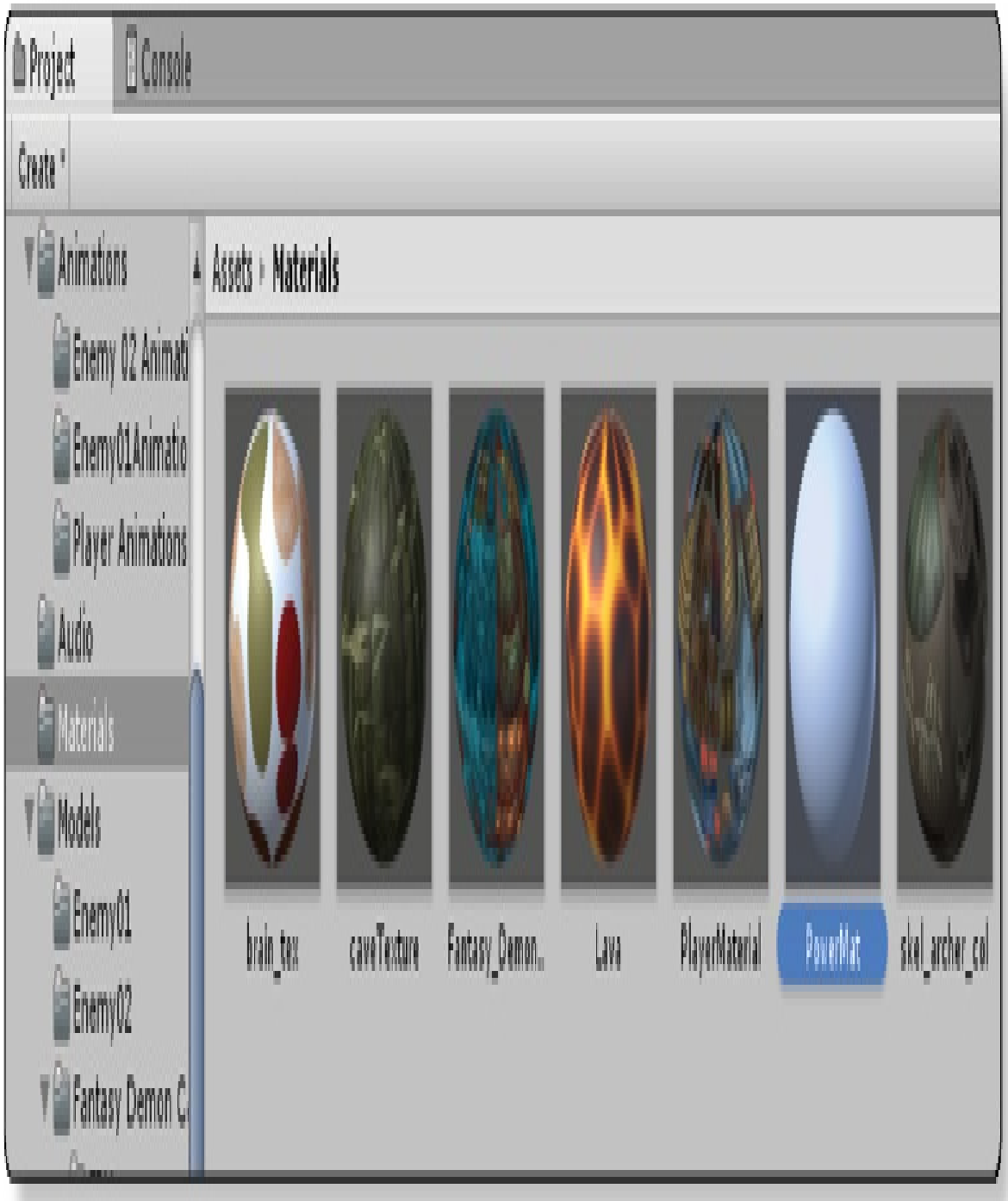
Previsualización de partículas

Particle Effect
Pause
Restart
Stop
Playback Speed
1.00
Playback Time
57.54
Particles
10
Speed Range
1.0 - 3.2
Simulate Layers
Nothing
Resimulate
Show Bounds

Controlador de efectos de partícula

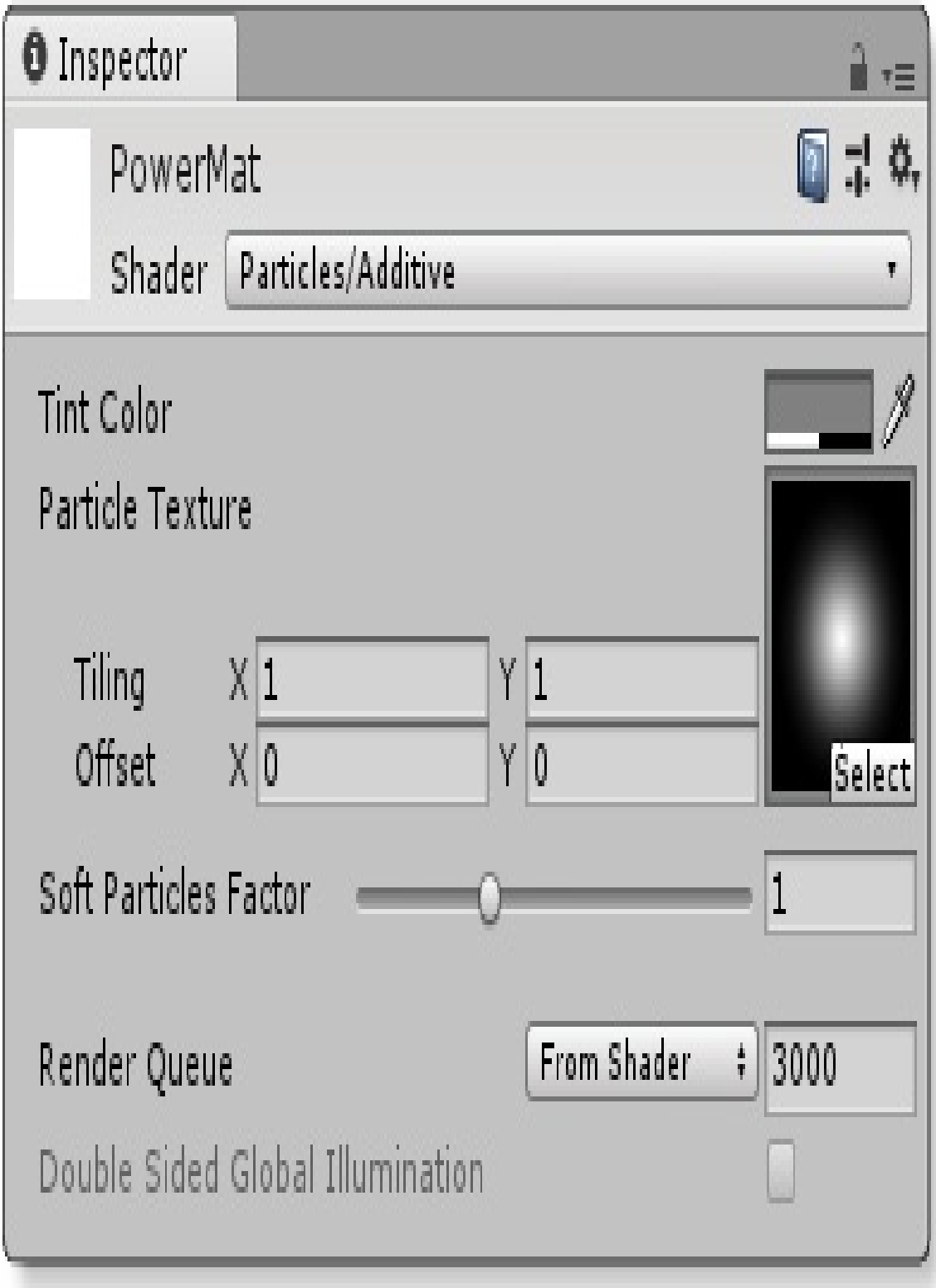
Cuando seleccionamos el objeto que contiene el sistema de partículas, podemos observar su previsualización con este controlador.

Vaya al folder “Materials”, haga un nuevo material y renómbrelo “PowerMat”.


Project
Console
Create
Animations
Enemy 02 Animati
Enemy01Animatio
Player Animations
Audio
Materials
Models
Enemy01
Enemy02
Fantasy Demon C
Assets ▸ Materials
brain_tex
caveTexture
Fantasy_Demon...
Lava
PlayerMaterial
PowerMat
skel_archer_col

Nuevo material

En “Inspector” asigne los valores que se muestran para el material.


Inspector
PowerMat
Shader
Particles/Additive
Tint Color
Particle Texture
Tiling
X 1
Y 1
Offset
X 0
Y 0
Select
Soft Particles Factor
1
Render Queue
From Shader
3000
Double Sided Global Illumination

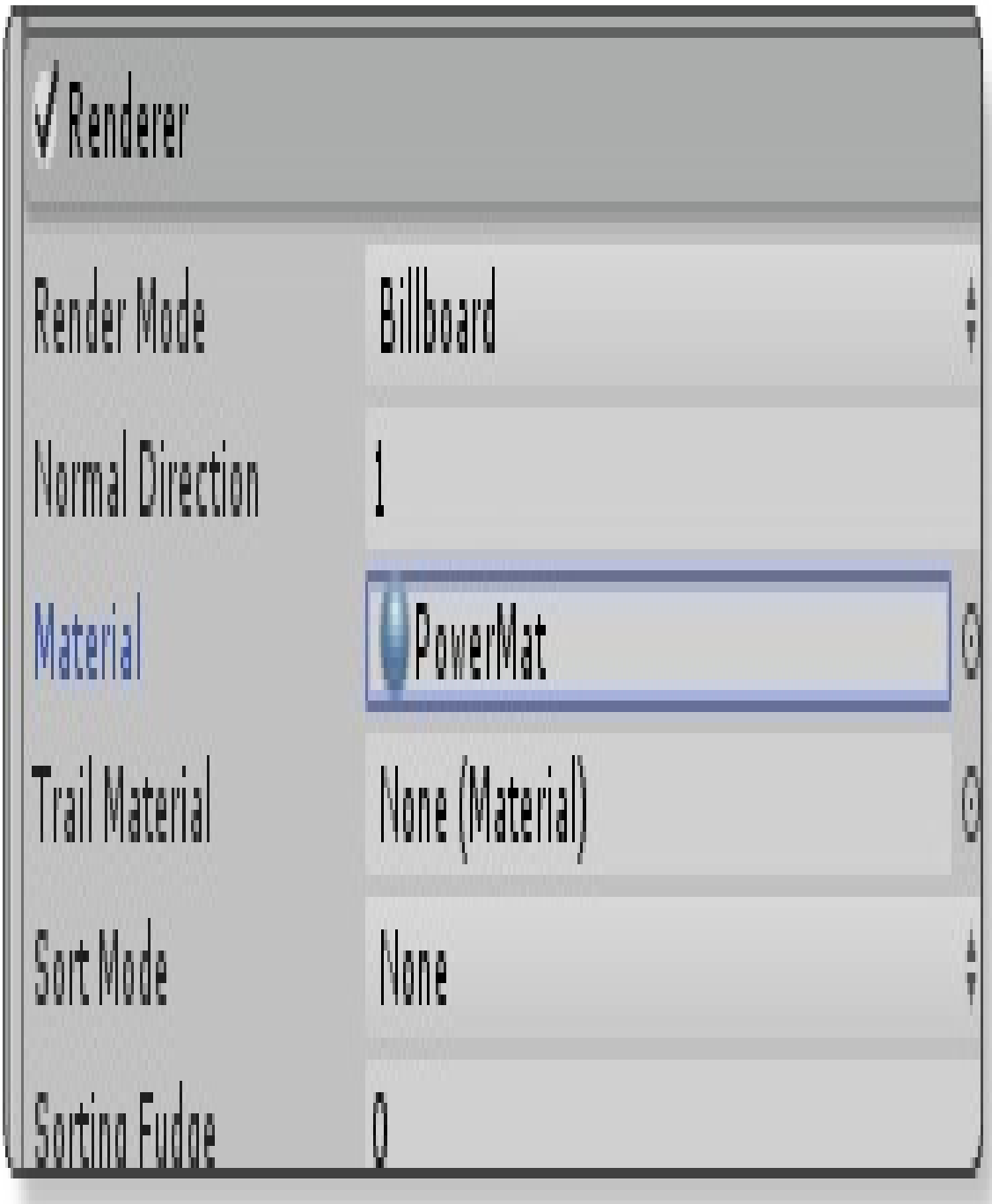
Valores de material “PowerMat”

En la previsualización pulse el botón “Select” y asigne el tipo “Particle Default”.

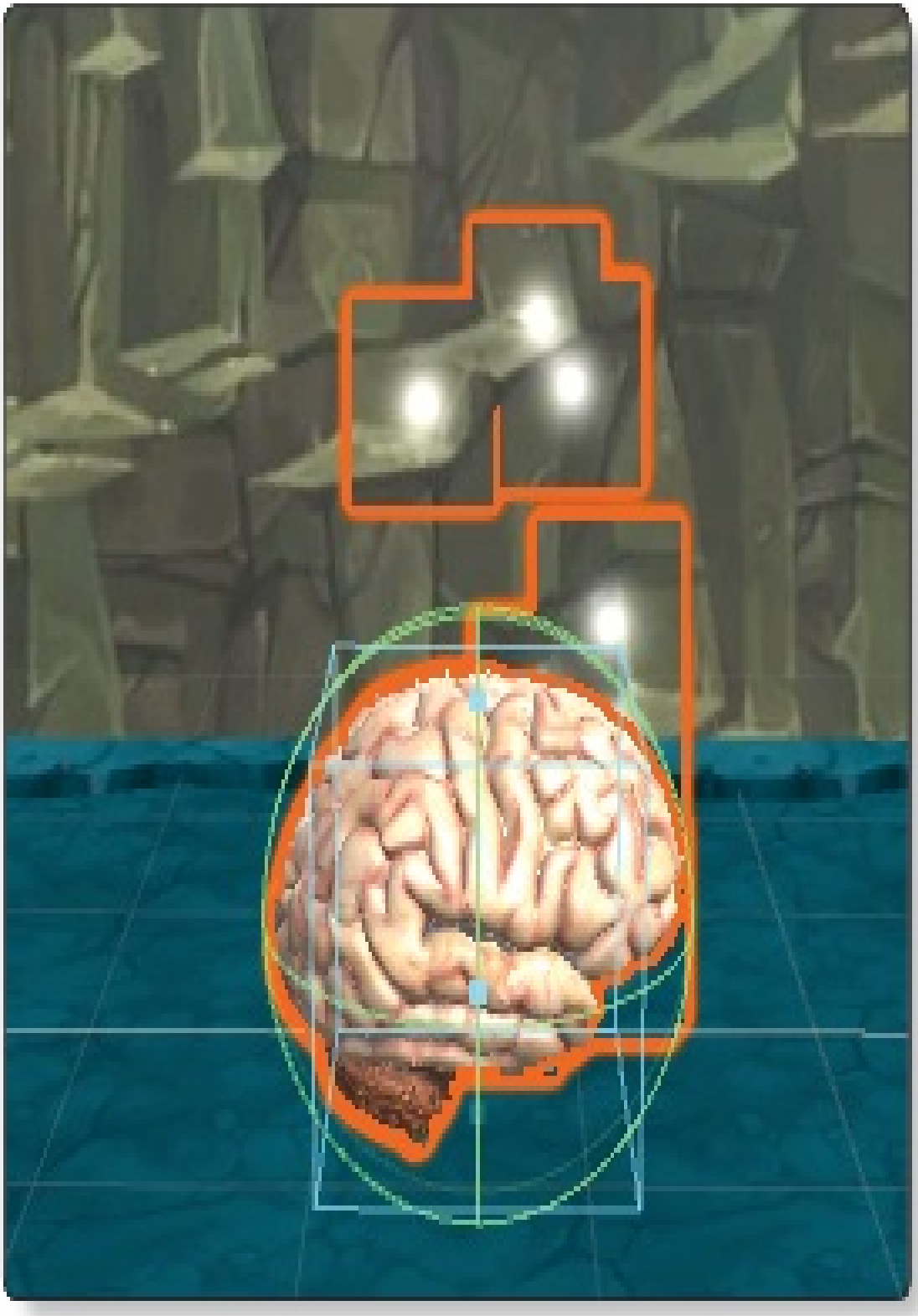
PowerMat

Previsualización de material PowerMat

Seleccione “PowerItem”, expanda la pestaña de “Renderer” del sistema de partículas y asigne el material creado.

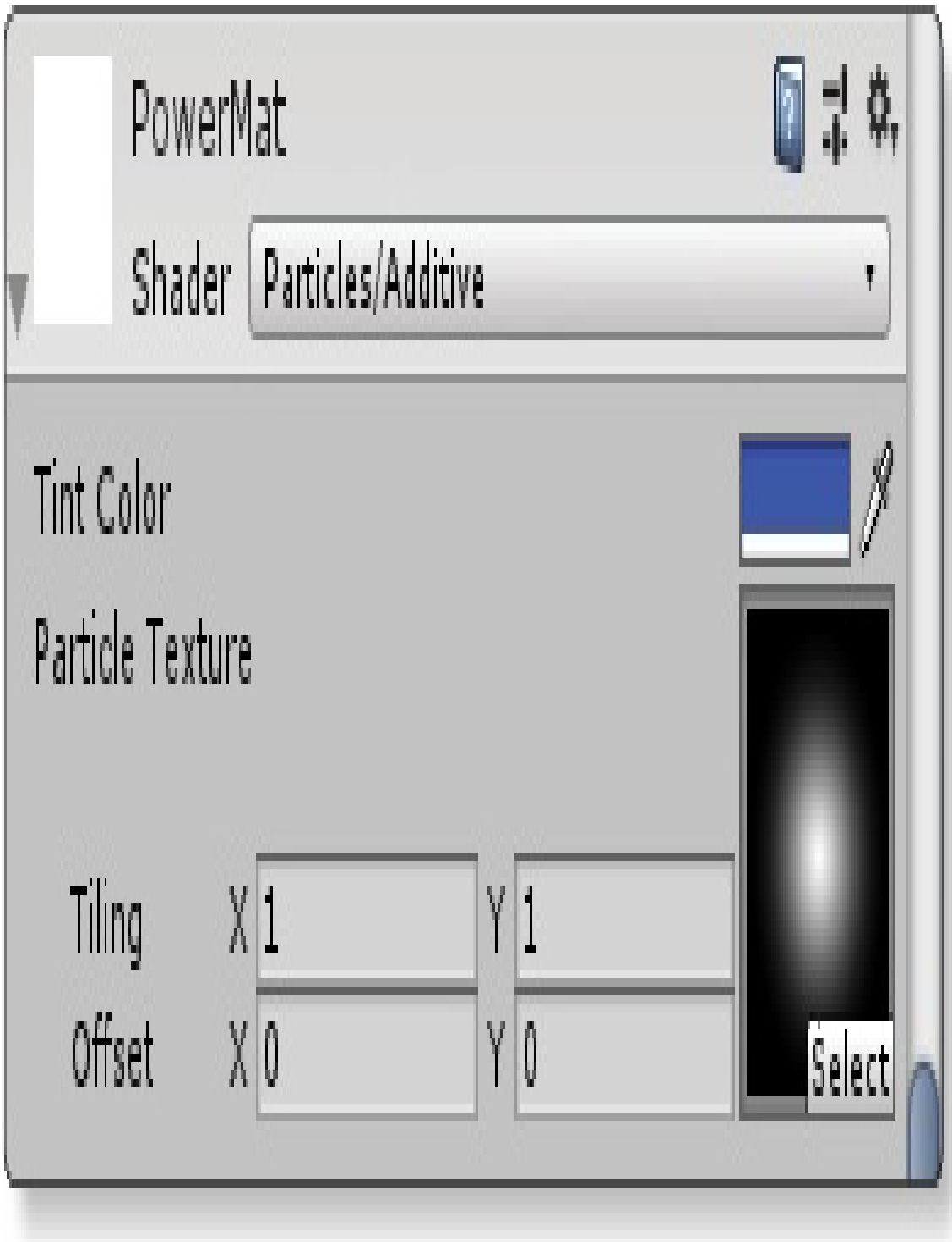

Renderer
Render Mode
Billboard
Normal Direction
1
Material
PowerMat
Trail Material
None (Material)
Sort Mode
None
Sorting Fudge
0

Asignación de material

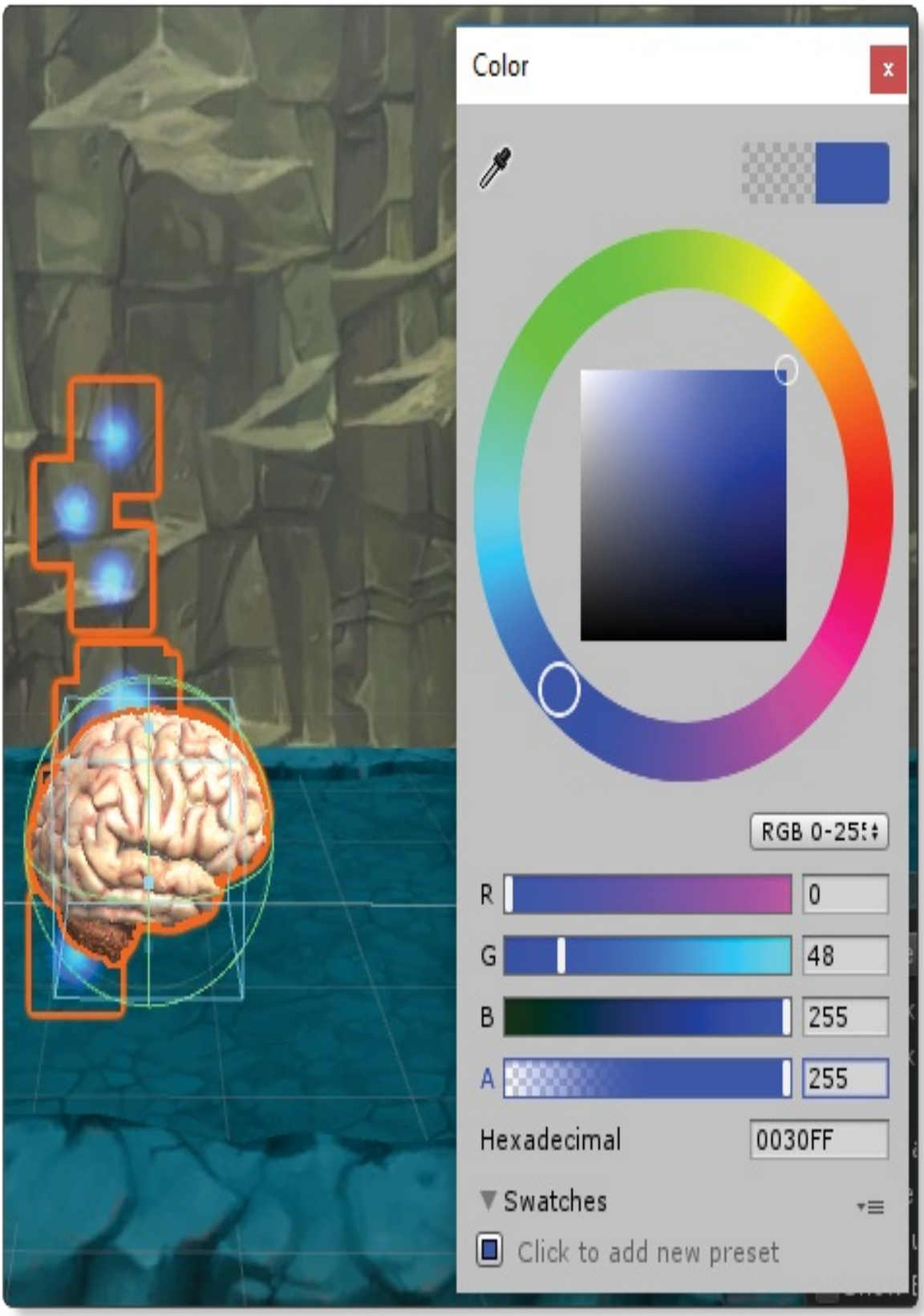


Previsualización de material PowerMat

Expanda las opciones del material y asigne los valores que se muestran para cambiar el color de las partículas.


PowerMat
Shader
Particles/Additive
Tint Color
Particle Texture
Tiling
X 1
Y 1
Offset
X 0
Y 0
Select

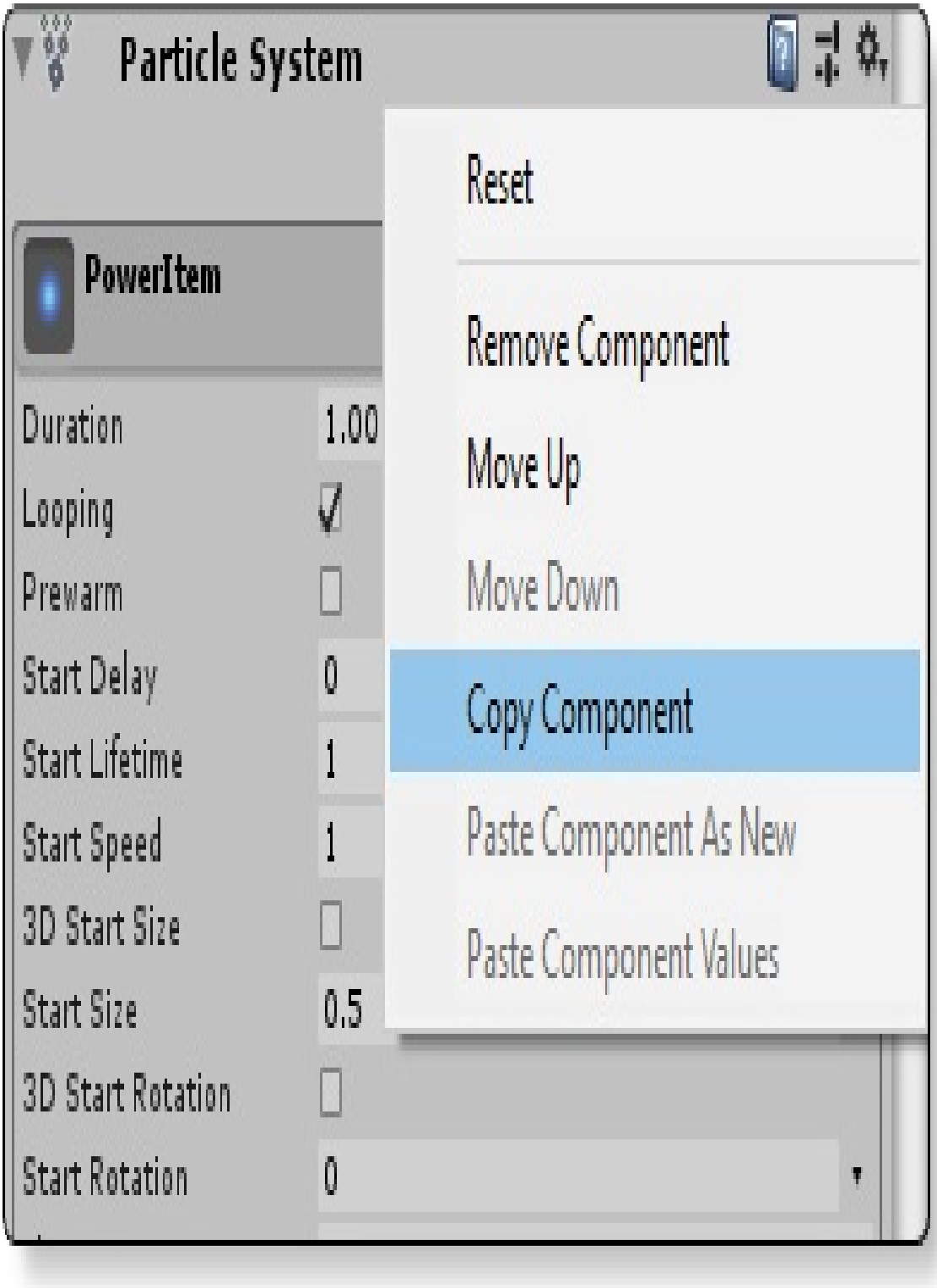
Valores del Shader de PowerMat


Color
x
RGB 0-25
R
0
G
48
B
255
A
255
Hexadecimal
0030FF
Swatches
Click to add new preset

Cambio de color a las partículas

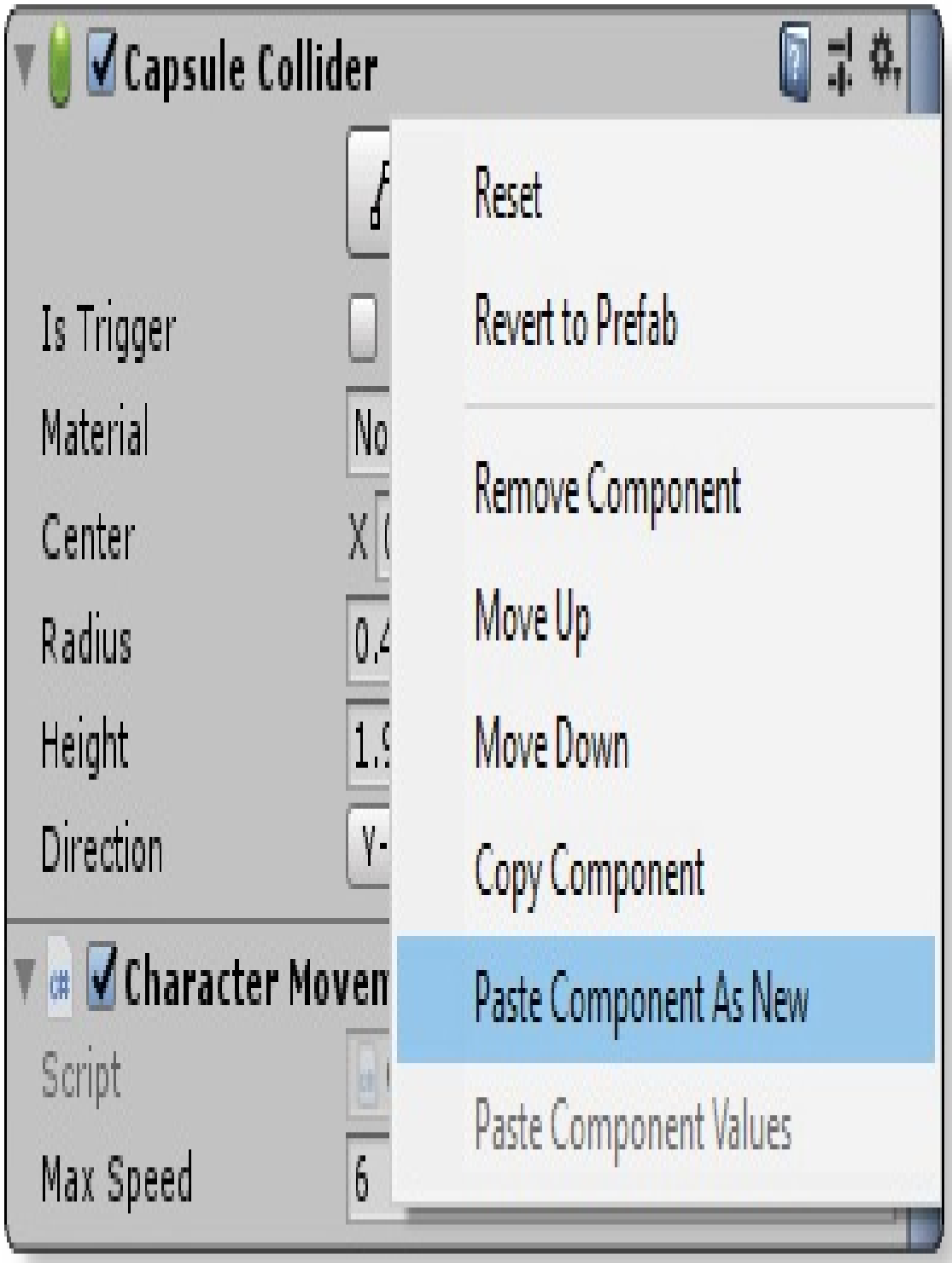
A continuación, queremos aplicar el mismo efecto de partículas a nuestro “Player”.

Copie el sistema de partículas de “PowerItem”.


Particle System
PowerItem
Duration
1.00
Looping
Prewarm
Start Delay
0
Start Lifetime
1
Start Speed
1
3D Start Size
Start Size
0.5
3D Start Rotation
Start Rotation
0
Reset
Remove Component
Move Up
Move Down
Copy Component
Paste Component As New
Paste Component Values

Copiar componente

Seleccione “Player” y pegue el componente copiado.


Capsule Collider
Is Trigger
Material
Center
Radius
Height
Direction
Character Moven
Script
Max Speed
6
Reset
Revert to Prefab
Remove Component
Move Up
Move Down
Copy Component
Paste Component As New
Paste Component Values

Pegar componente

KnifeSpawn

Ambos objetos con sistema de partículas

A continuación, necesitamos hacer lo siguiente:

Deshabilitar el sistema de partículas del Player

Tomar el ítem cuando el Player colisione con él

Desactivar el script de “PlayerHealth” durante un lapso de tiempo para volverlo invencible

Habilitar el sistema de partículas del “Player” durante el tiempo que permanezca invencible

Vaya al script “PowerItemScript” y escriba el siguiente código

```
public class PowerItemScript : MonoBehaviour {

    private GameObject player;
    private AudioSource audio;
    private ParticleSystem particleSystem;
    private PlayerHealth playerHealth;

    // Use this for initialization
    void Start () {

        player = GameManager.instance.Player;
        playerHealth = player.GetComponent<PlayerHealth> ();
        playerHealth.enabled = true;

        particleSystem = player.GetComponent<ParticleSystem> ();
        particleSystem.enableEmission = false;

    }
```

Creación e inicialización de variables

Previsualice el juego para observar que “PowerItem” tiene activas sus partículas y “Player” no.

Previsualización de Player sin partículas activas

PROBLEMA: Cuando no se encuentra en escena el “PowerItem”, nuestro “Player” tiene activo su sistema de partículas.

SOLUCIÓN: Debemos de desactivar las partículas del “Player” al inicializar su script de “PlayerHealth”.

Vaya al script de “PlayerHealth”, haga e inicialice la variable que se muestra.

```
public AudioClip pickItem;

private ParticleSystem particleSystem;

public int CurrentHealth {
    get { return currentHealth; }
    set {
        if (value < 0)
            currentHealth = 0;
        else
            currentHealth = value;
    }
}

// Use this for initialization
void Start () {

    anim = GetComponent<Animator> ();
    currentHealth = startingHealth;
    characterMovement = GetComponent<CharacterMovement> ();
    audio = GetComponent <AudioSource> ();
    particleSystem = GetComponent<ParticleSystem>();
    particleSystem.enableEmission = false;

}
```

Solución al error

Regrese al script y escriba la siguiente función.

```
    // Update is called once per frame
    void Update () {

    }


    void OnTriggerEnter (Collider other){
        if (other.gameObject == player){
            print ("Pick PowerItem");
            particleSystem.enableEmission = true;
            playerHealth.enabled = false;
            Destroy (gameObject);
        }
    }
}
```

Función OnTriggerEnter

Previsualice el juego para comprobar que al colisionar con el “Player”:

Se destruye el Item

El personaje adquiere partículas

Player con partículas activas

Se deshabilita el script de “PlayerHealth” del “Player” haciéndolo invencible.

Player Health (Script)
Script
PlayerHealt
Starting Health
100

Deshabilitación de PlayerHealth

A continuación, debemos de dar un tiempo específico para la invencibilidad.

Regrese al script de “PowerItemScript”, haga e inicialice las variables que se indican.

```
private GameObject player;
private AudioSource audio;
private ParticleSystem particleSystem;
private PlayerHealth playerHealth;

private MeshRenderer meshRenderer;
private ParticleSystem brainParticles;

// Use this for initialization
void Start () {

    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealth> ();
    playerHealth.enabled = true;

    particleSystem = player.GetComponent<ParticleSystem> ();
    particleSystem.enableEmission = false;

    meshRenderer = GetComponentInChildren <MeshRenderer> ();
    brainParticles = GetComponent<ParticleSystem> ();

}
```

Creación e inicialización de variables

Dado que el componente “MeshRenderer” se encuentra en el modelo hijo de “PowerItem”, por ello lo mandamos llamar con “GetComponentInChildren”.

Haga el siguiente “IEnumerator” para brindar 10 segundos de invencibilidad y después de dicho tiempo, rehabilitar el componente “PlayerHealth” a nuestro “Player”.

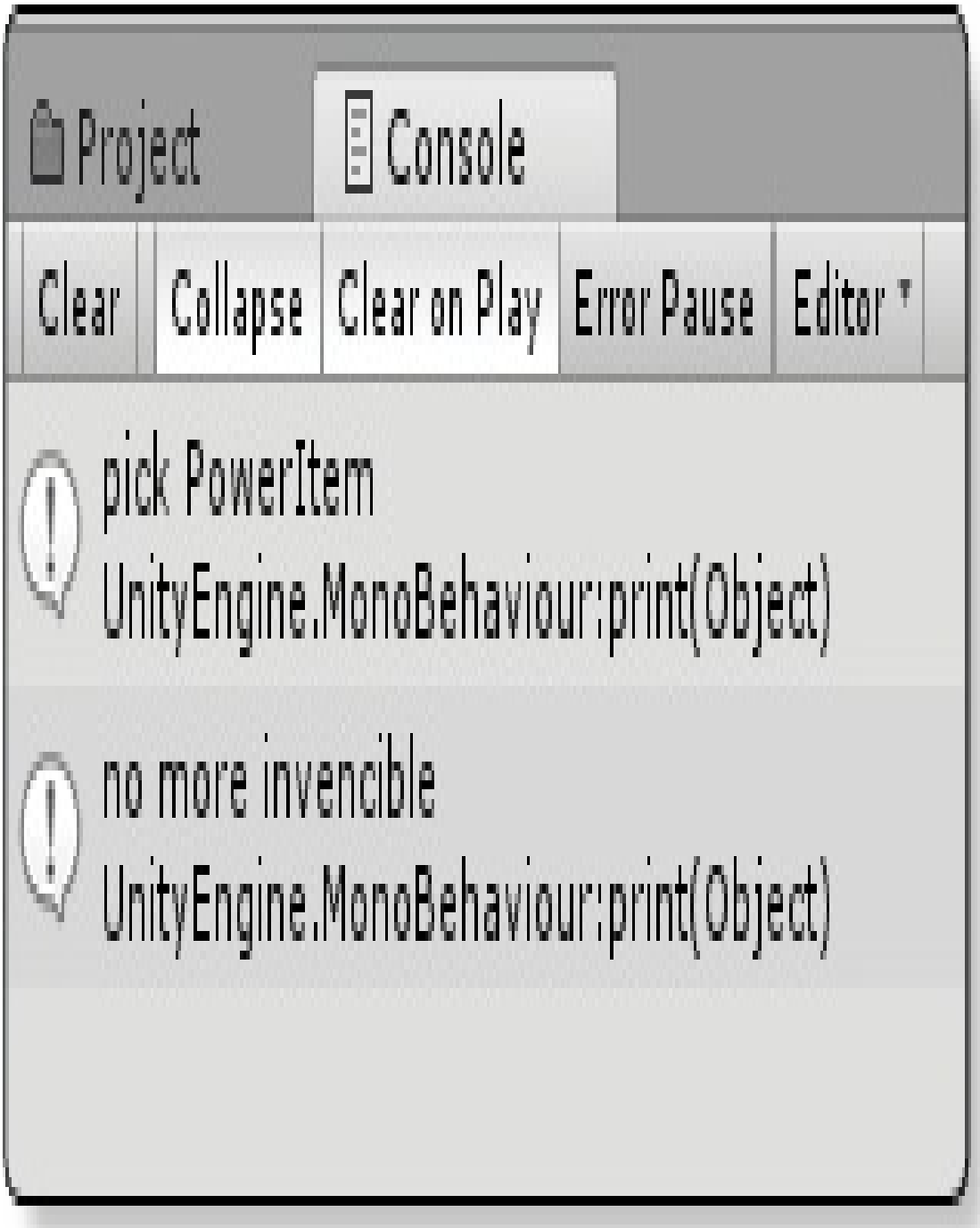
NOTA:

Deberá reestructurar la función “OnTriggerEnter” como se muestra para mandar llamar el “Coroutine”.

```
    void OnTriggerEnter (Collider other){
        if (other.gameObject == player){

            StartCoroutine (InvincibleRoutine ());
            meshRenderer.enabled = false;

        }
    }

    public IEnumerator InvincibleRoutine () {

        print ("pick PowerItem");
        particleSystem.enableEmission = true;
        playerHealth.enabled = false;
        brainParticles.enableEmission = false;

        yield return new WaitForSeconds (10f);
        print ("no more invencible");
        particleSystem.enableEmission = false;
        playerHealth.enabled = true;
        Destroy (gameObject);
    }
}
```

Creación de IEnumerator

Previsualice el juego para comprobar la invencibilidad.


Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
pick PowerItem
UnityEngine.MonoBehaviour:print(Object)
no more invencible
UnityEngine.MonoBehaviour:print(Object)

Comprobación de funcionalidad

APLICACIÓN DE ANIMACIONES PARA LA TOMA DE ITEMS

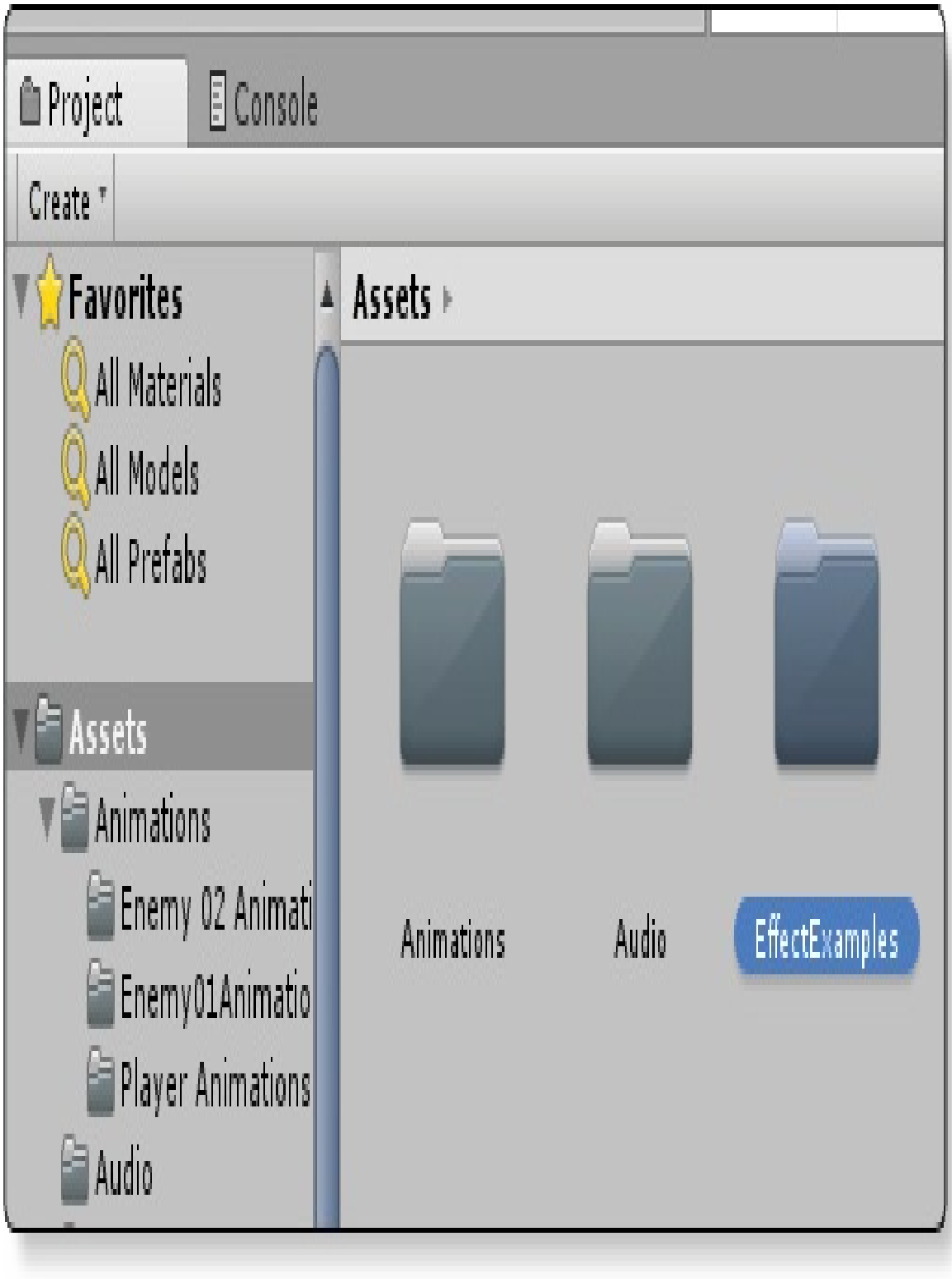
En este apartado utilizaremos una librería externa para configurar un efecto de explosión cuando nuestro “Player” recoja el Item de invencibilidad y nos ocuparemos de algunos “bugs” que puedan existir al momento de la configuración.

Vaya a “AssetSotre” de Unity y descargue el siguiente paquete de efectos:

Unity Particle Pack. https://goo.gl/NQJy4C

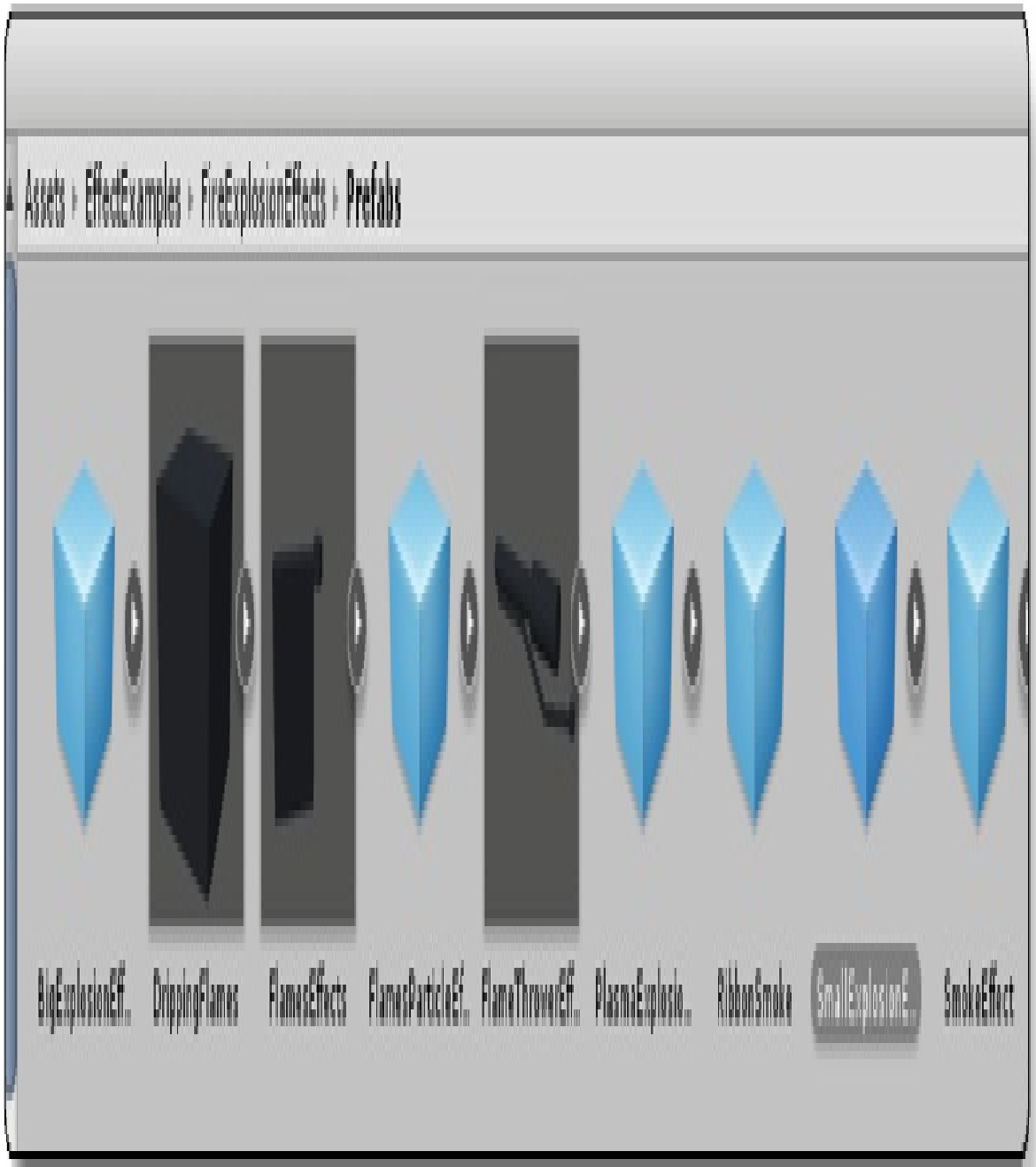
Scene
Game
Asset Store
Animator
Language:English
Matu007
Filters
Unity Particle Pack
Unity Essentials/Asset Packs
Unity Technologies
(511)
FREE
Update
A set of sample particle assets to use in your games, and to help you understand how to achieve certain effects using the Unity Particle System component and modules. You'll find Fire, Explosions, Rain and other cool effects here all for free! Note: this pack is new for Unity 5.5 and as such is only for use in version 5.5 and above.
Home
3D Models
Animation
Audio
Complete Projects
Editor Extensions
Particle Systems
Scripting
Services
Shaders
Textures & Materials
Unity Essentials
Asset Packs
Beta Content
Certification
Sample Projects
Other
NEW
Top Paid

La carpeta de los efectos aparecerá en raíz de “Assets”.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
Enemy 02 Animati
Enemy01Animatio
Player Animations
Audio
Assets
Animations
Audio
EffectExamples

Folder de Efectos Importados

En la carpeta de “EffectExamples” localice el prefab “SmallExplolsion” y en “Inspector” desactive su loop.


Assets ▸ EffectExamples ▸ FireExplosionEffects ▸ Prefabs
BigExplosionEf...
DrippingFlames
FlameEffects
FlameParticleEf...
FlameThrowerEf...
PlasmaExplosio...
RibbonSmoke
SmallExplosionEf...
SmokeEffect

Prefab SmallExplosion

SmallExplosionEffect
Duration
4.00
Looping
Prewarm

Desactivación de Loop

Vaya al script “PowerItemScript”, haga la siguiente variable y la función que se muestra.

private MeshRenderer meshRenderer;
private ParticleSystem brainParticles;
public GameObject pickupEffect;

Variable para el efecto

```
        Destroy (gameObject);
    }


    void Pickup () {
        Instantiate (pickupEffect, transform.position, transform.rotation);
    }
}
```

Función PickUp

En Unity asigne el efecto “SmallExplosion” en el script.

Freeze Rotation
X
Y
Z
Power Item Script (Script)
Script
PowerItemScript
Pickup Effect
SmallExplosionEffect

Asignación de efecto en script

Haga un nuevo script, nómbrelo “ItemExplode”, asígnelo al objeto “PowerItem” y escriba el siguiente código para instanciar el efecto y en seguida destruirlo después de un segundo.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ItemExplode : MonoBehaviour {

    public GameObject pickupEffect;

    public void PickUp (){

        GameObject newPickupEffect =
        (GameObject)Instantiate (pickupEffect, transform.position, transform.rotation);
        Destroy (newPickupEffect, 1);
    }
}
```

Script ItemExplode

A continuación, debemos de mandar llamar la función creada desde el script de “PowerItemScript”

Vaya al script “PowerItemScript”, haga las variables que se muestran, inicialícelas y modifique la función “IEnumerator” como se indica.

```
public GameObject pickupEffect;

private ItemExplode itemExplode;
private SphereCollider sphereCollider;


// Use this for initialization
void Start () {

    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealth> ();
    playerHealth.enabled = true;

    particleSystem = player.GetComponent<ParticleSystem> ();
    particleSystem.enableEmission = false;

    meshRenderer = GetComponentInChildren <MeshRenderer> ();
    brainParticles = GetComponent<ParticleSystem> ();

    itemExplode = GetComponent<ItemExplode> ();
    sphereCollider = GetComponent<SphereCollider> ();
```

Variables para efecto

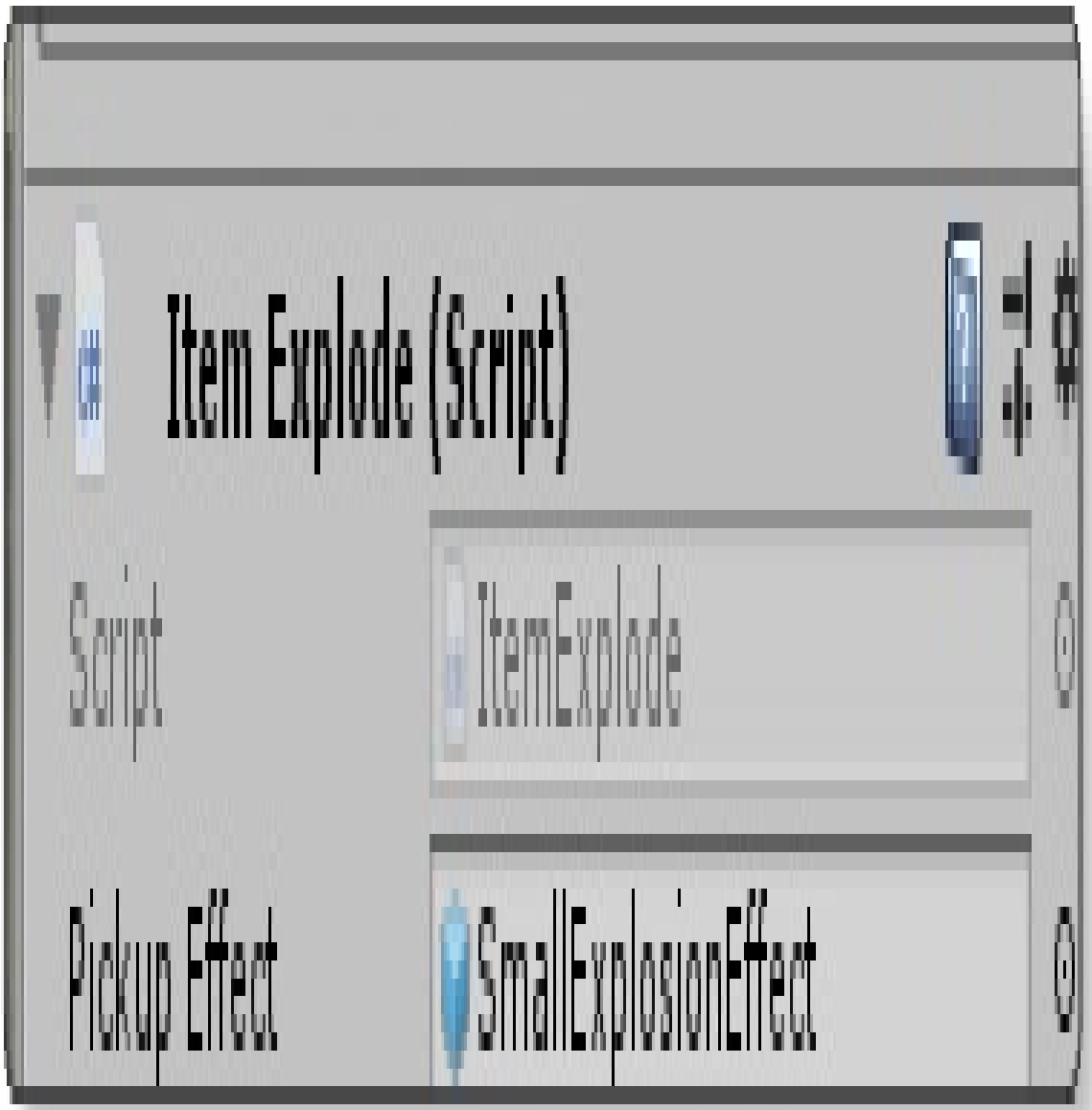
```
public IEnumerator InvincibleRoutine () {


    print ("pick PowerItem");
    itemExplode.Pickup ();
    particleSystem.enableEmission = true;
    playerHealth.enabled = false;
    brainParticles.enableEmission = false;
    sphereCollider.enabled= false;
```

Modificación de código

NOTA:

Deshabilitamos su “SphereCollider” para que no vuelva a tomar el item.

En Unity asigne el prefab de la explosión en el script “ItemExplode”.


Item Explode (Script)
Script
ItemExplode
Pickup Effect
SmallExplosionEffect

Asignación de efecto en el script

Previsualice el juego para observar el efecto de explosión cuando el “Player” recoge el “ítem”.

Previsualización de efecto

Haga un Prefab de “PowerItem” y elimínelo de la escena.

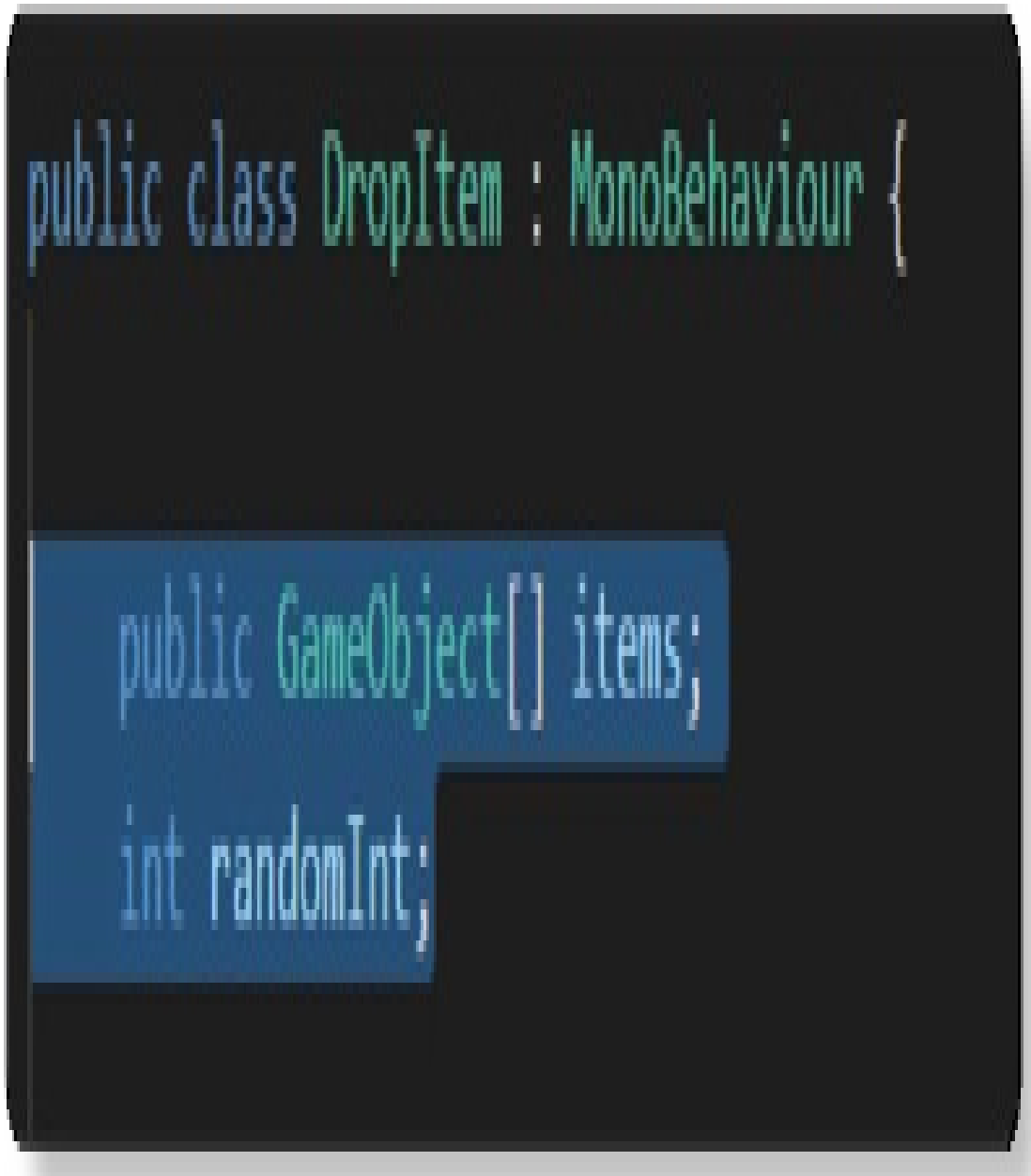
PROGRAMACIÓN DE ENEMIGOS QUE ARROJAN ITEMS

En este apartado programaremos la funcionalidad para que un enemigo arroje un “ítem” aleatorio al ser destruido.

Se verá un concepto nuevo “Arrays” o matrices para poder regresar un valor aleatorio de una lista que en este caso contendrá los items.

Haga un nuevo script, nómbrelo “DropItems” y asígnelo a “Enemy02.

Haga las variables que se muestran para crear un “Array” y una variable tipo “Int”.

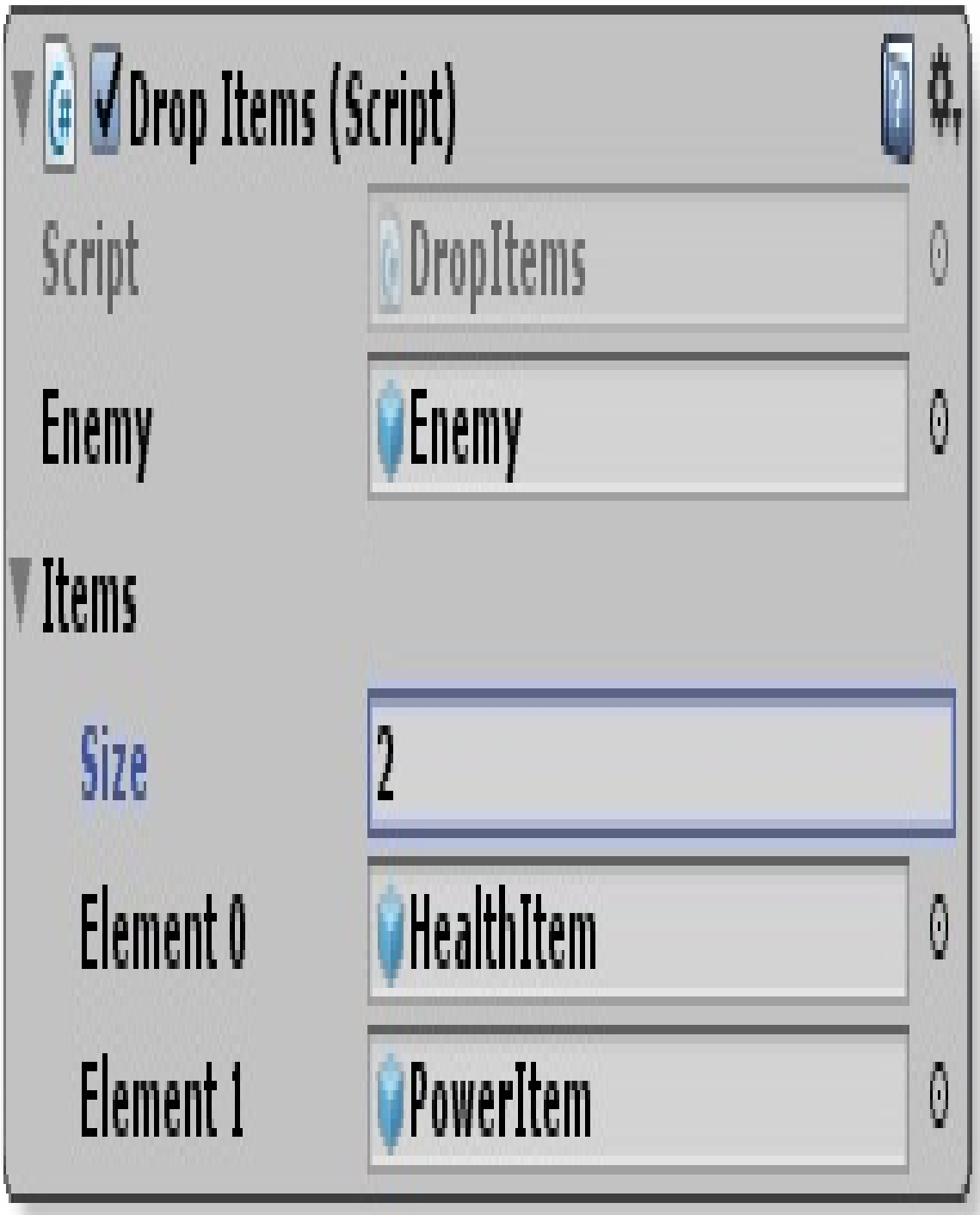

public class DropItem : MonoBehaviour {
public GameObject[] items;
int randomInt;

Variable tipo Array

Cuando necesitamos que una variable almacene una serie de objetos, creamos una matriz o como se le llama en inglés un “Array”.

Este “Array” lo declaramos con los paréntesis cuadrados [ ]

En Unity asigne el tamaño de la matriz y asigne los “ítems” como objetos

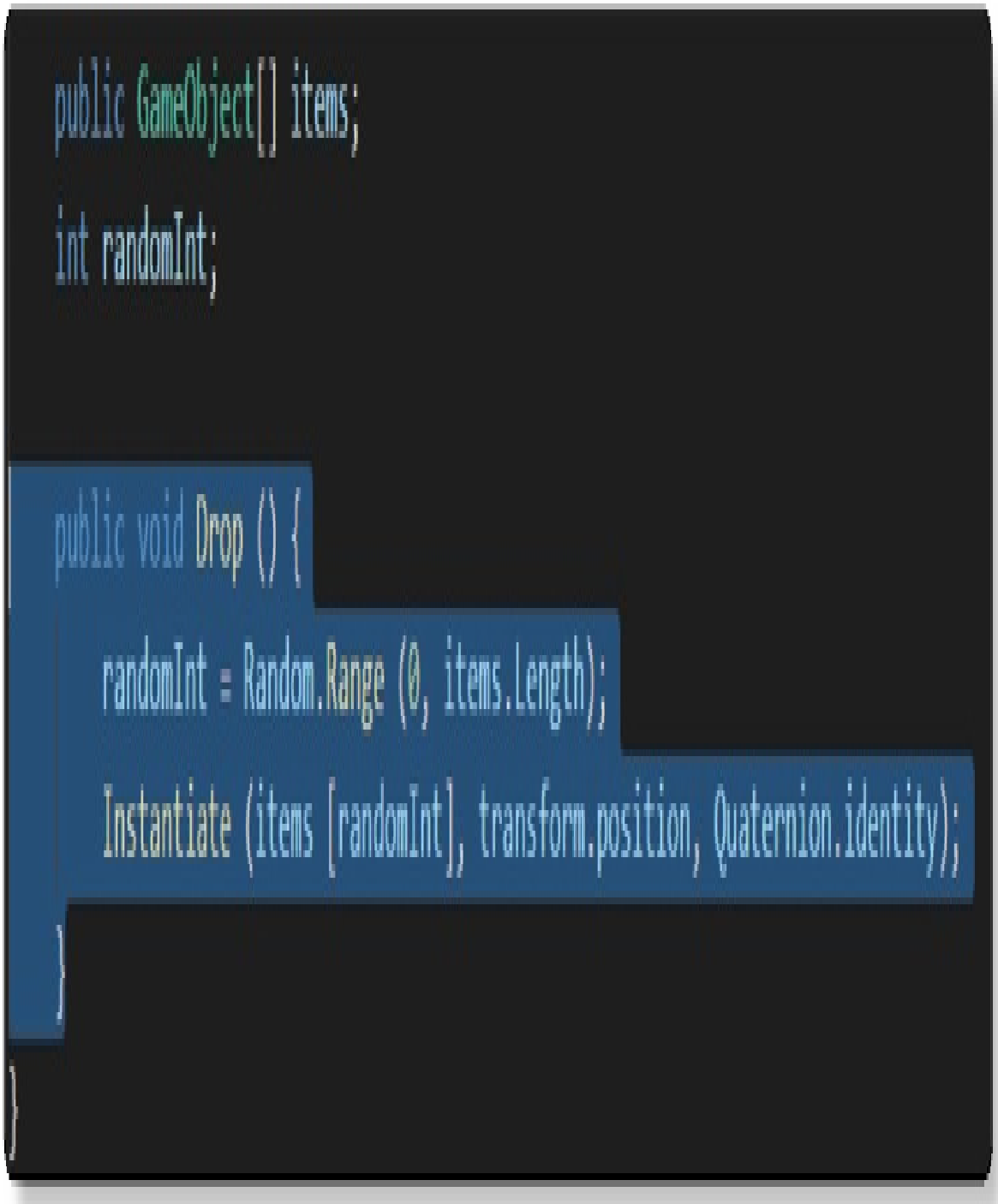

Drop Items (Script)
Script
DropItems
Enemy
Enemy
Items
Size
2
Element 0
HealthItem
Element 1
PowerItem

Asignación de elementos para Array

NOTA:

El número cero cuenta como valor y los “ítems” que añadimos son los Prefabs creados previamente.

Regrese al script “DropItems” y escriba la siguiente función para arrojar un número aleatorio entre el tamaño del “Array”.


public GameObject[] items;
int randomInt;
public void Drop () {
randomInt = Random.Range (0, items.Length);
Instantiate (items [randomInt], transform.position, Quaternion.identity);
}
}

Función Drop

El ítem se instanciará conforme a la posición y la rotación (Quaternion) del objeto original.

Vaya al script “Enemy02Health” y haga la siguiente variable e inicialícela para mandar llamar la función creada en el script “DropItems”.

```
    private DropItem dropItem;

    // Use this for initialization
    void Start () {

        rigidbody = GetComponent <Rigidbody> ();
        capsuleCollider = GetComponent <CapsuleCo
        anim = GetComponent <Animator> ();
        isAlive = true;
        currentHealth = startingHealth;
        audio = GetComponent<AudioSource> ();
        dropItem = GetComponent<DropItem> ();
    }
```

Variable para mandar llamar elementos de DropItem

Mande llamar la función “Drop” en la función “KillEnemy”.

```
void KillEnemy () {

    capsuleCollider.enabled = false;
    anim.SetTrigger ("EnemyDie");
    rigidbody.isKinematic = true;
    audio.PlayOneShot (killAudio);

    StartCoroutine (removeEnemy ());
    print ("drop random Item");
    dropItem.Drop ();
}
```

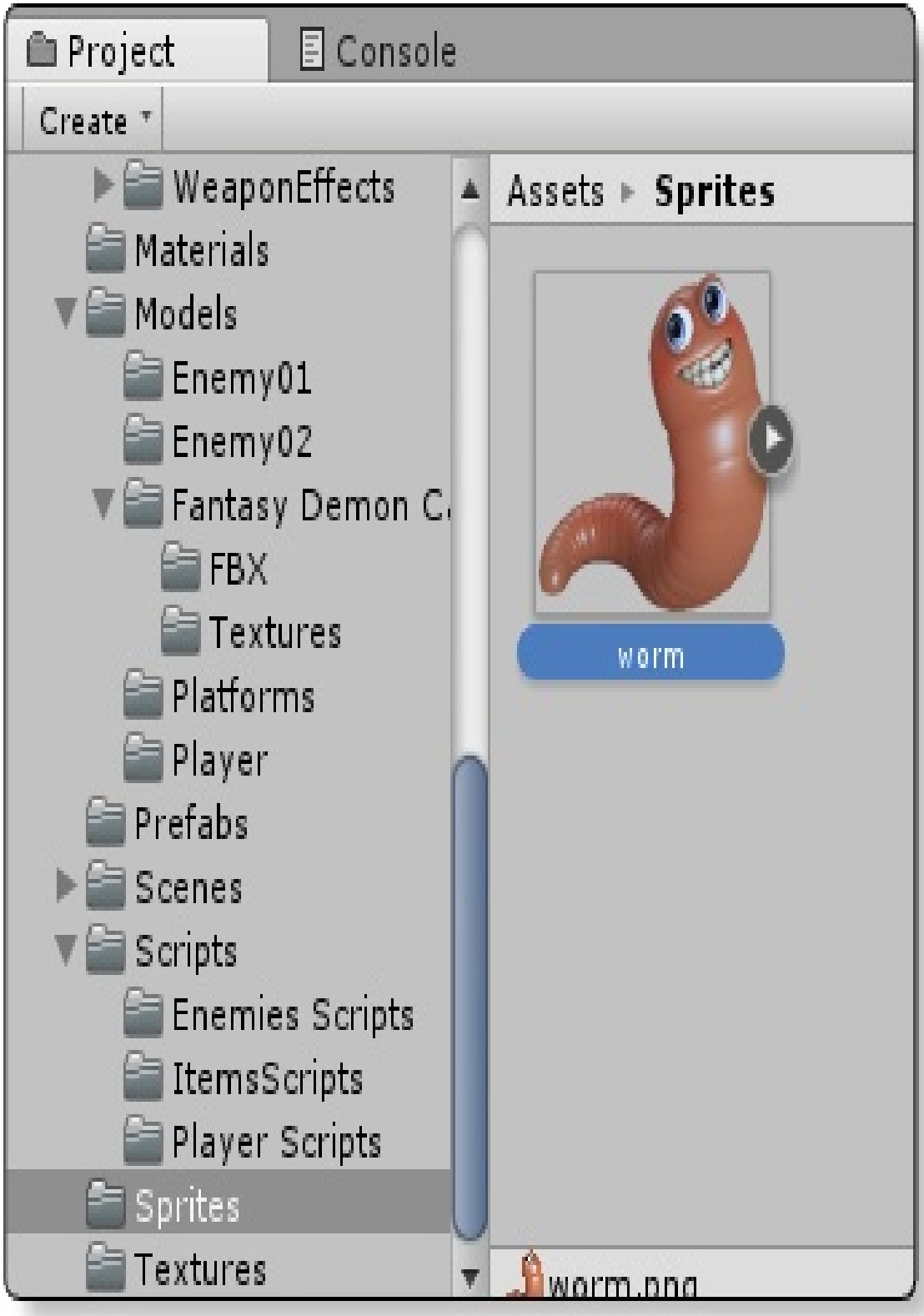
Mandar llamar la función Drop

Previsualice el juego para observar que cuando se destruye el enemigo, éste arroja un Item.

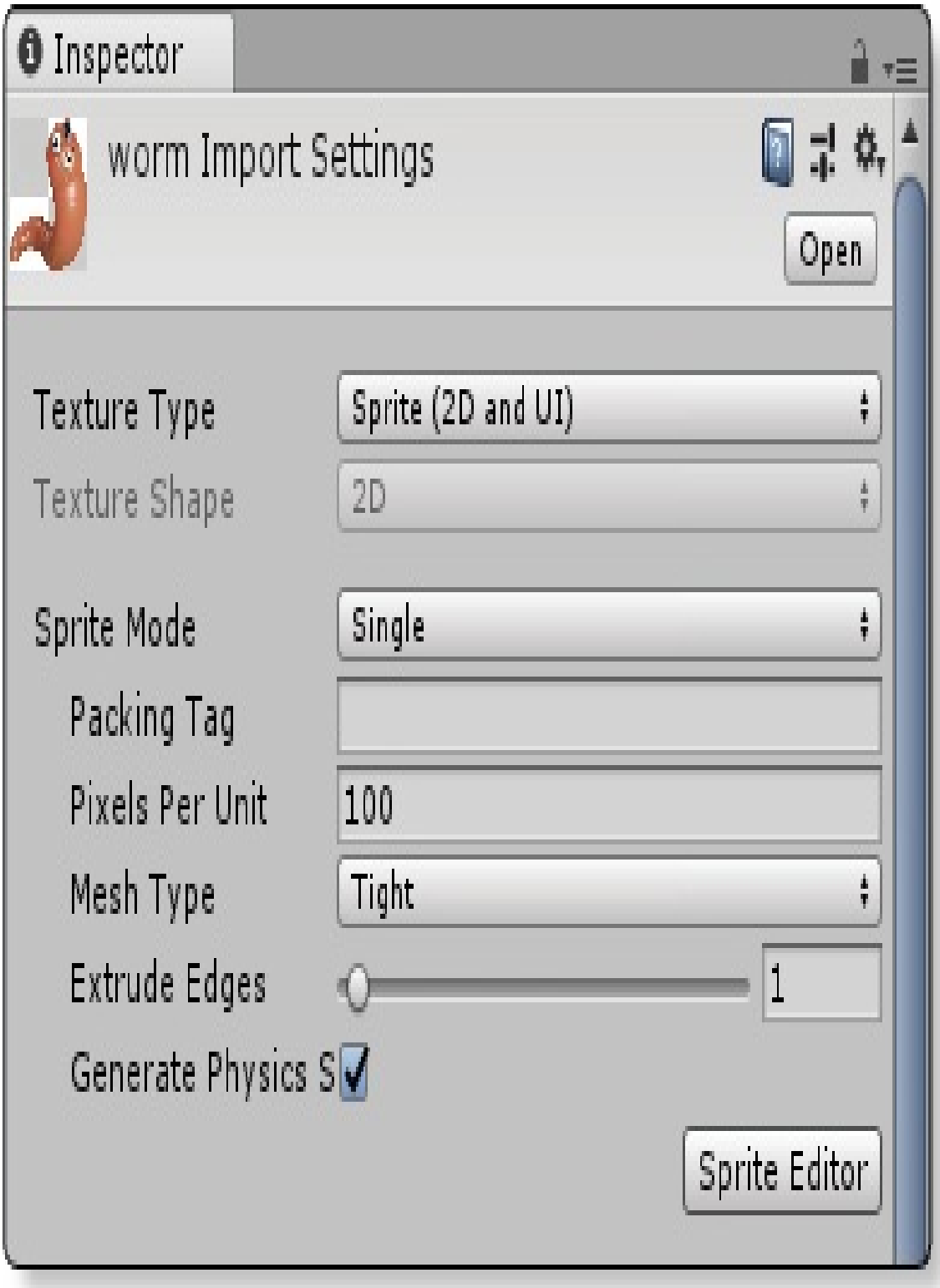
PROBLEMA: El ítem de salud aparece debajo de la plataforma.

Error de pivote

SOLUCIÓN: Localice la “Sprite” original y en “Inspecto” clic en “Sprite Editor”.

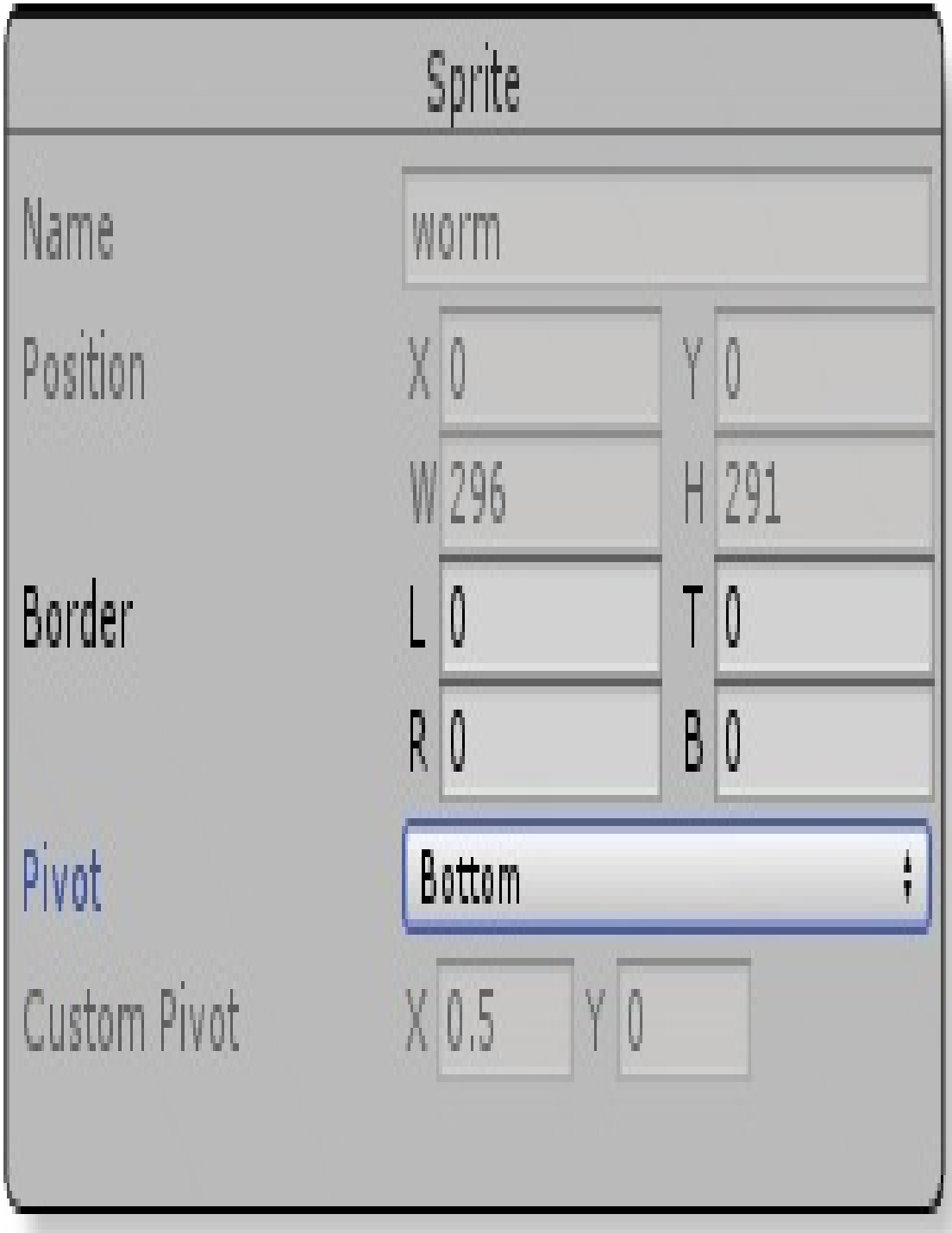

Project
Console
Create
WeaponEffects
Materials
Models
Enemy01
Enemy02
Fantasy Demon C
FBX
Textures
Platforms
Player
Prefabs
Scenes
Scripts
Enemies Scripts
ItemsScripts
Player Scripts
Sprites
Textures
Assets ▸ Sprites
worm
worm.png

Sprite original

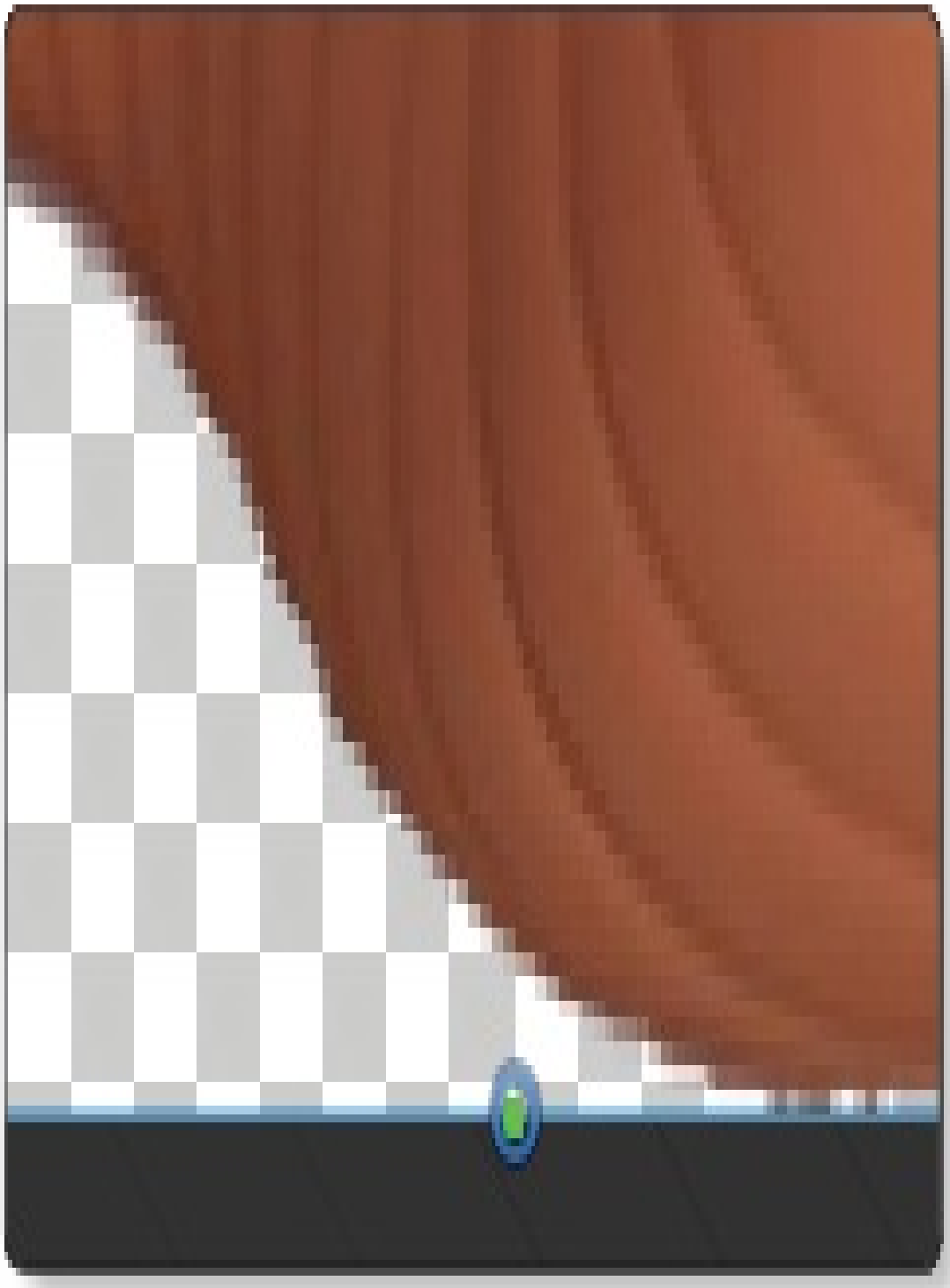

Inspector
worm Import Settings
Open
Texture Type
Sprite (2D and UI)
Texture Shape
2D
Sprite Mode
Single
Packing Tag
Pixels Per Unit
100
Mesh Type
Tight
Extrude Edges
1
Generate Physics S
Sprite Editor

Botón Sprite Editor

En el cuadro de valores que aparece en la parte inferior derecha de “SpriteEditor” asigne su pivote abajo (“Bottom”) como se muestra, clic en Apply y cierre el editor.


Sprite
Name
worm
Position
X 0
Y 0
W 296
H 291
Border
L 0
T 0
R 0
B 0
Pivot
Bottom
Custom Pivot
X 0.5
Y 0

Pivote Bottom



Correcta posición del pivote

Previsualice el juego para observar que el ítem de salud aparece correctamente.

Item apareciendo correctamente

PROBLEMA: El “PowerItem” aparece por debajo de la plataforma.

SOLUCIÓN:

Coloque el “PowerItem” en la escena

Desempareje el objeto “goodBrain2” del objeto “PowerItem”

Coloque el objeto “PowerItem” en la base de “goodBrain2”

Empareje nuevamente

Conviértalo en Prefab nuevamente

Asígnelo nuevamente en el “Array”

Item apareciendo correctamente

Haga más elementos en el “Array” para que en algunas ocasiones no arrojen “ítems”.

3D Sound Settings
Drop Items (Script)
Script
DropItems
Items
Size
4
Element 0
HealthItem
Element 1
PowerItem
Element 2
None (Game Object)
Element 3
None (Game Object)

Asignación de elementos nulos en Array

RETO AL ESTUDIANTE

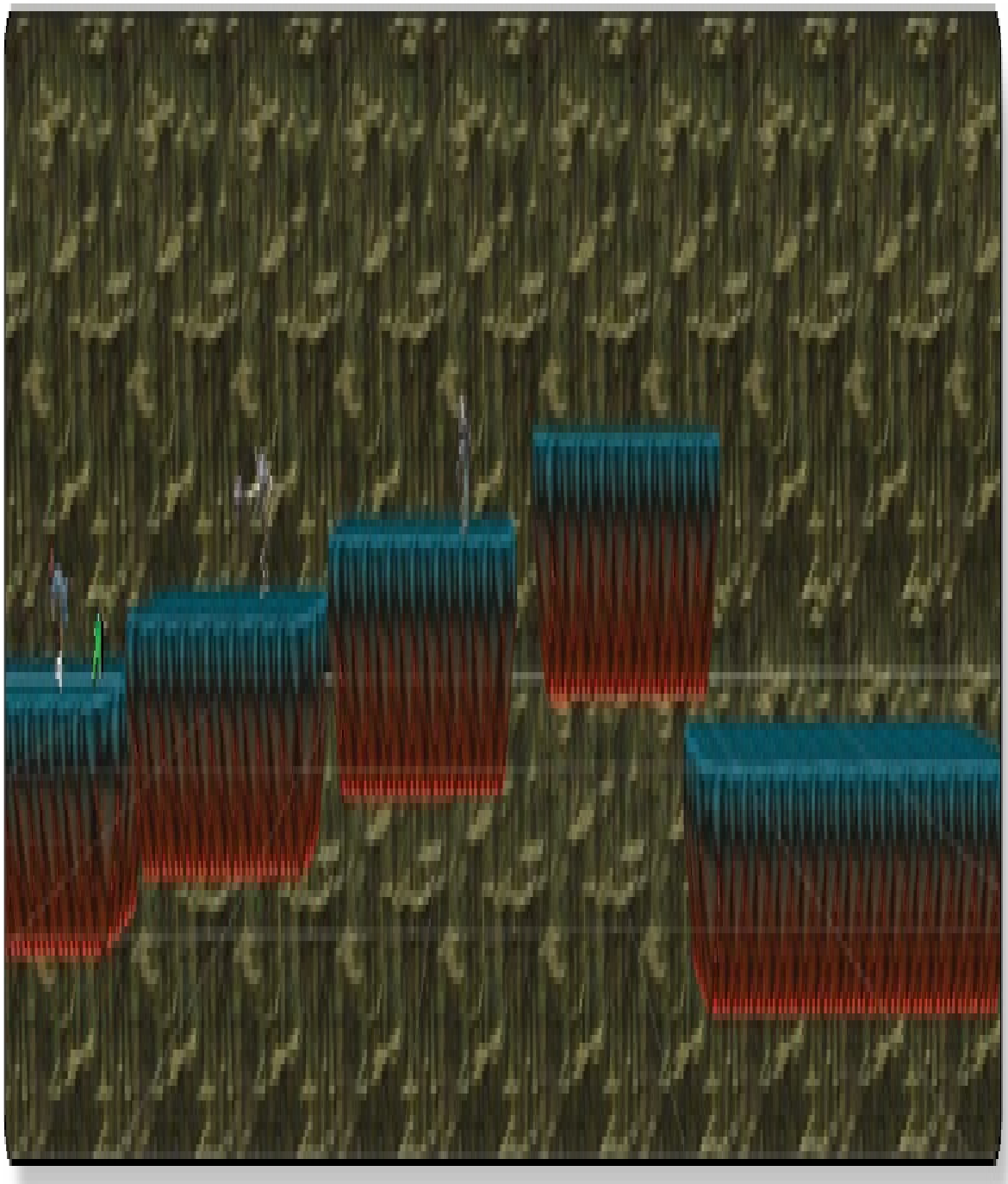
Hacer un ítem para que el personaje salte más alto y que dure 10 segundos

Añadir el nuevo Item al Array

Añadir animación al ítem de Health para cuando sea recogido por el player

Añadir clip de audio para cuando el ítem de power sea recogido

Expandir el nivel



Expansión del nivel

6

DETALLES AVANZADOS

INTRODUCCIÓN

En este capítulo trabajaremos en las siguientes funcionalidades para añadir elementos avanzados profesionales para nuestro videojuego.

Creación y programación de plataformas con movimientos verticales

Crearemos un método de “KillBox” para que cuando nuestro Player caiga a través de él en el vació, su salud sea cero.

Programaremos la funcionalidad de reaparición del Player una vez que su salud haya llegado a cero y trabajaremos en la solución de “bugs” conforme vayamos avanzando.

Programaremos más “Checkpoints” para que se active su funcionalidad una vez que nuestro “Player” pase a través de ellos.

Crearemos un sistema de vidas de nuestro Player apoyados del sistema GUI para poderlas visualizar en el juego.

Crearemos una pantalla de inicio, sus botones, sus funcionalidades y conoceremos como cargar escenas a partir de eventos.

Generaremos una pantalla y un estado de “GameOver”.

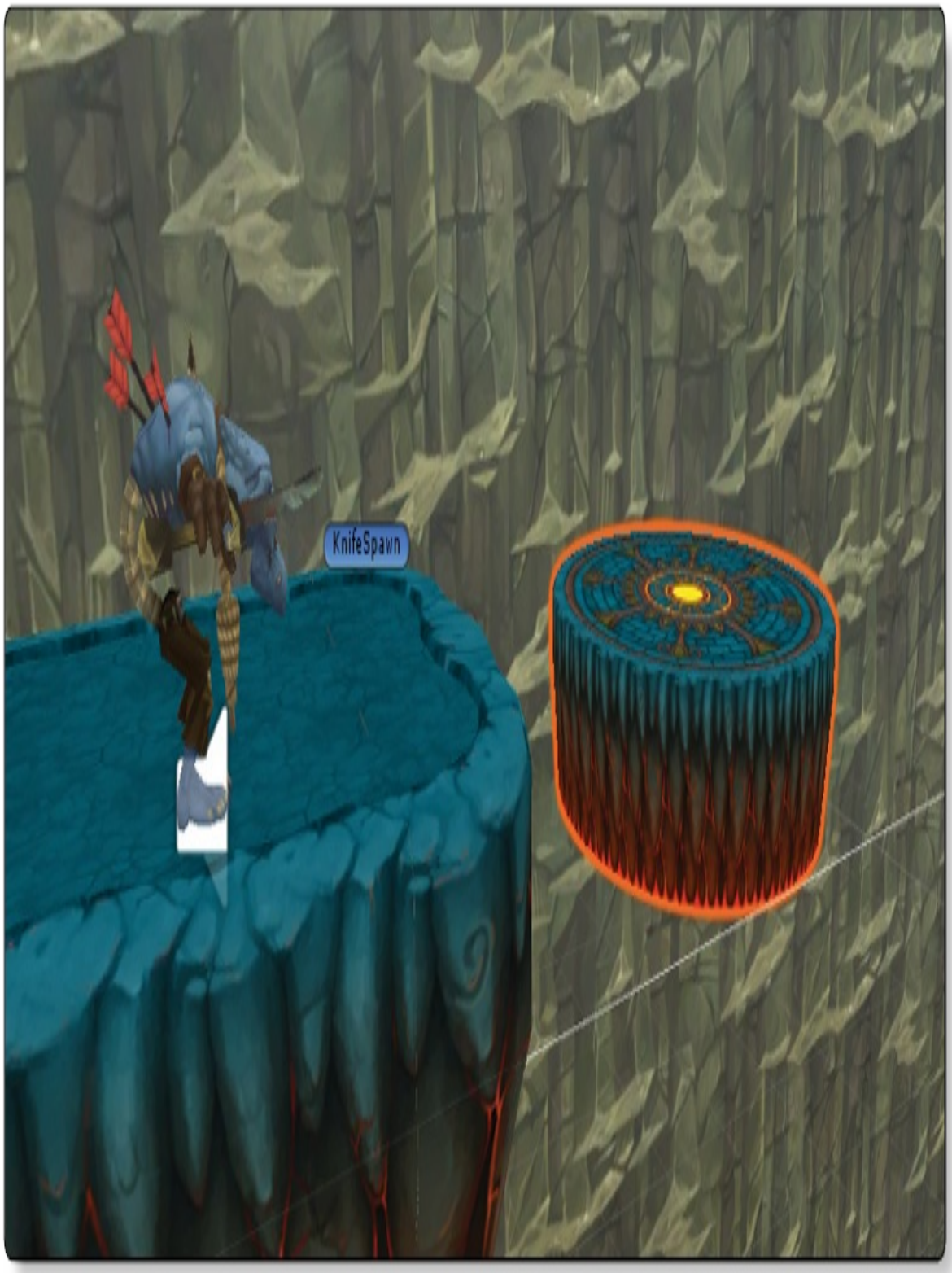
Finalizaremos con la creación de un Item que añada vidas a nuestro Player.

Creación y preparación de la plataforma móvil

En este apartado vamos a crear una plataforma con los componentes necesarios para que nuestro personaje pueda interactuar con ella.

Seguido de convertirla en Prefab y hacer su duplicado para pasar a su programación de movimiento.

En la carpeta de “resources” del capítulo, localice el modelo center_01_floor.fbx, con los conocimientos previamente adquiridos importe el modelo, texturícelo y colóquelo en la escena (Al borde de la última plataforma).


KnifeSpawn

Posicionamiento de la plataforma

NOTA:

Recuerde eliminar el componente “Animator” del objeto.

Haga un nuevo “EmptyObject”, renómbrelo “MobilePlatform” y hágalo padre del objeto “center_01_floor.fbx”

Correcto posicionamiento del nuevo objeto

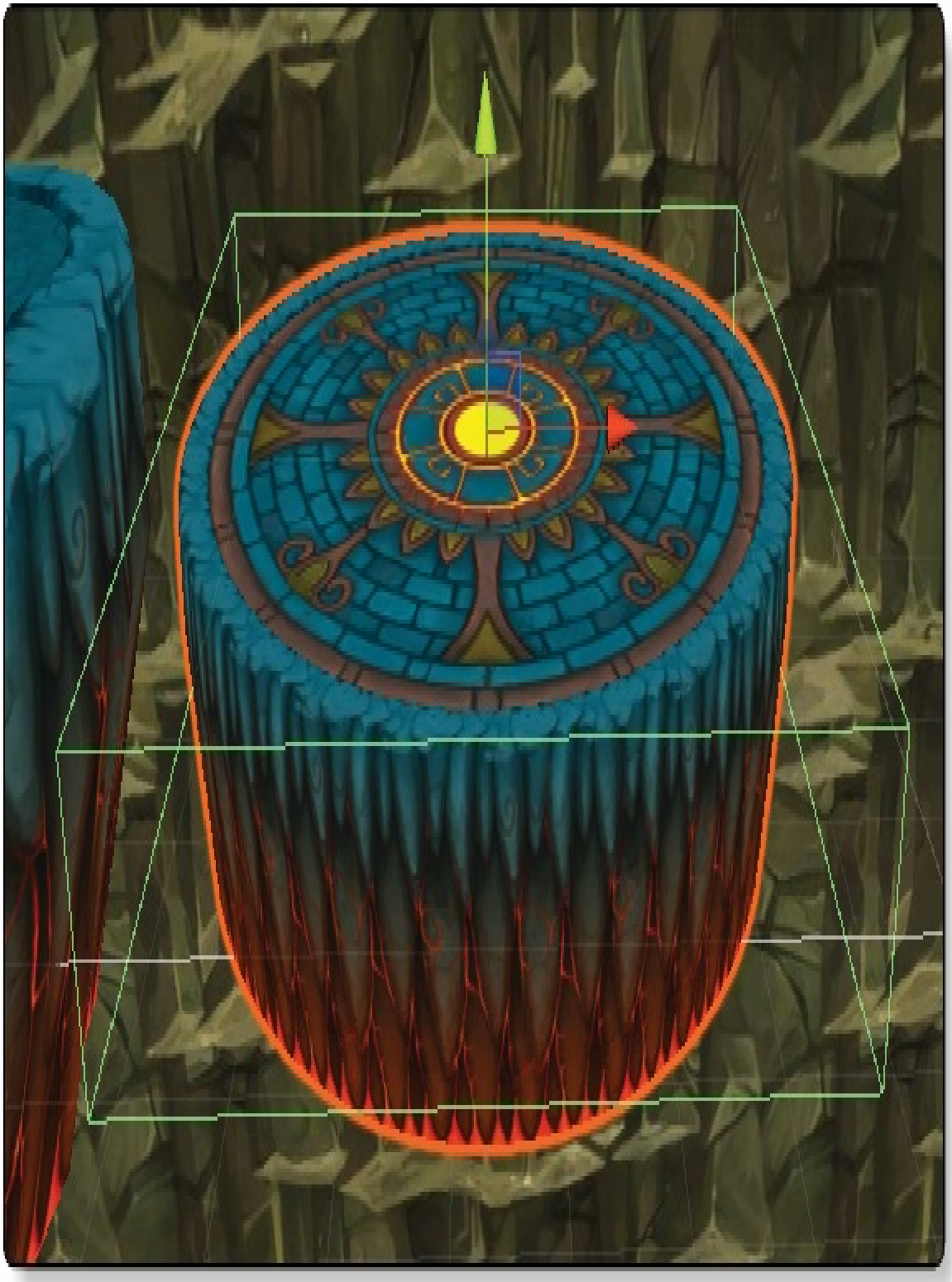
NOTA:

Recuerde que la ubicación del nuevo elemento debe de corresponder a la ubicación del objeto original

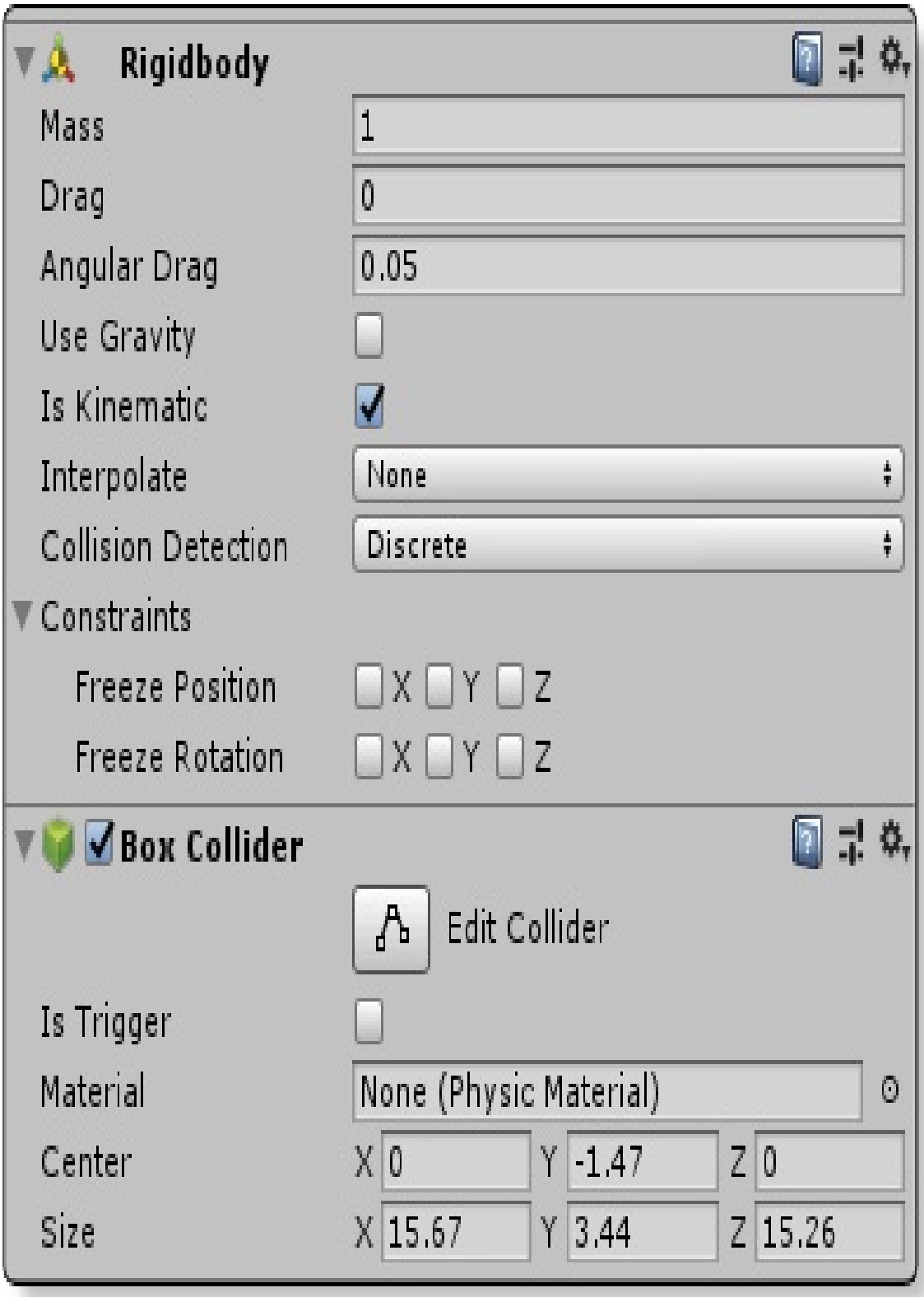
Seleccione MobilePlatform” y asigne los siguientes componentes.

RidigBody: Is Kinematic

Box Collider: Con las dimensiones que se muestran



Dimensiones del BoxCollider


Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Z
Freeze Rotation
X
Y
Z
Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y -1.47
Z 0
Size
X 15.67
Y 3.44
Z 15.26

Componentes

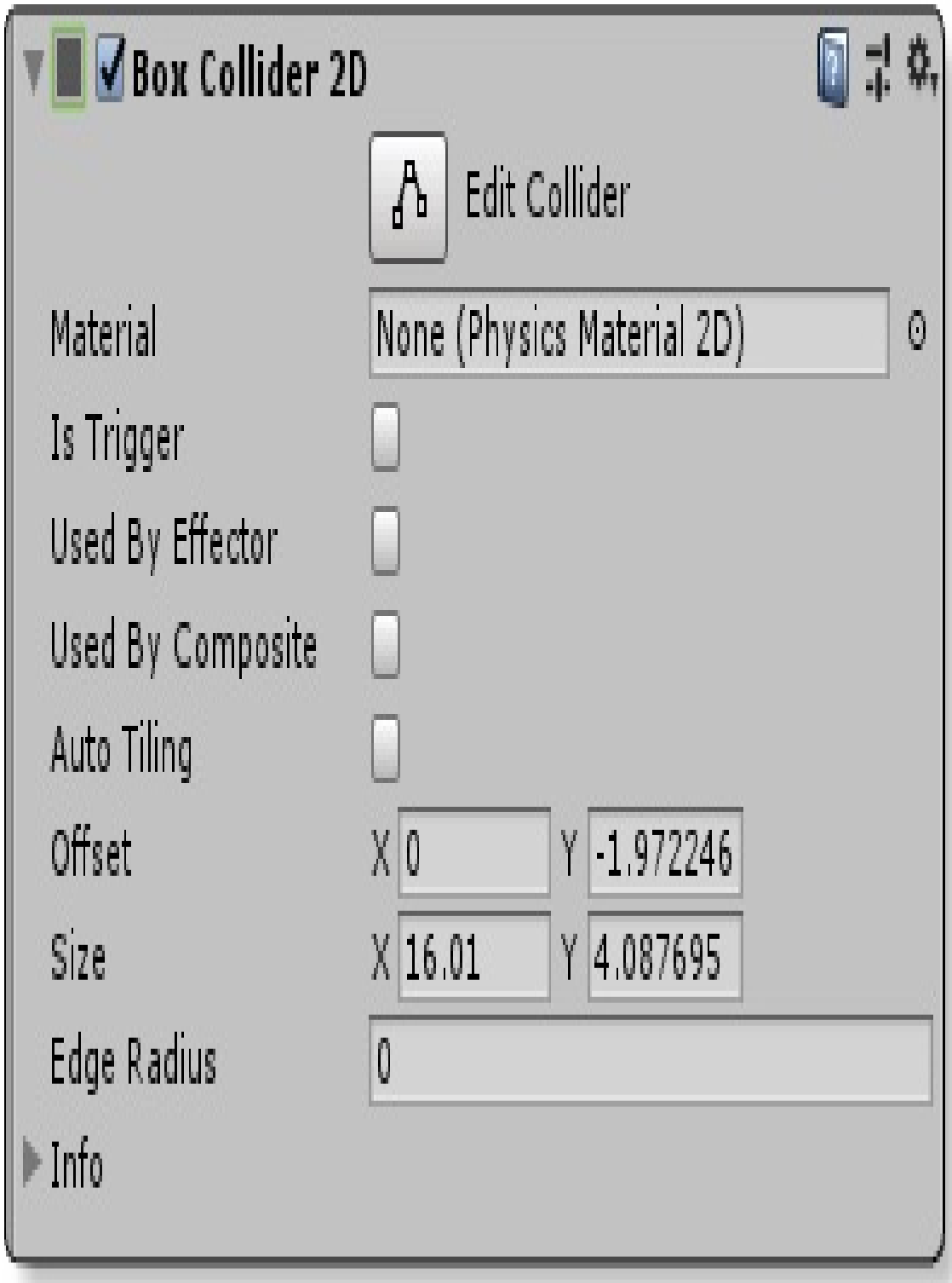
Seleccione el objeto “center_01_floor” y asigne los siguientes componentes.

2D Collider: Layer “midGround”.

Dimensiones de 2DBoxCollider

Inspector
center_01_floor
Static
Tag
Untagged
Layer
midGround
Prefab
Select
Revert
Apply

Asignación de Layer


Box Collider 2D
Edit Collider
Material
None (Physics Material 2D)
Is Trigger
Used By Effector
Used By Composite
Auto Tiling
Offset
X 0
Y -1.972246
Size
X 16.01
Y 4.087695
Edge Radius
0
Info

Componente 2DBoxCollider

Previsualice el juego para observar que el “Player” puede sostenerse en la nueva plataforma y brincar en ella.

Interacción de Player con la plataforma

NOTA:

Si el “Player” no aparece con status “Grounded” en algún punto de la plataforma, basta con subir su atributo de “GroundRadius”.

Convierta en Prefab el objeto “MobilePlatform”.

En “Hierarchy” renómbrelo como “VMobilePlatform” y conviértalo en “Prefab”.

Project
Console
Create
Assets
Prefabs
Ground Prefabs
Materials
Models
Prefabs
ForeGround
goodPlatfo...
MobilePlatf...
Platform 1
SmallPlatfo...
VMobilePlat...
Items

Prefabs nuevos

De esta manera tendremos el prefab únicamente de la plataforma y aparte la plataforma que va a tener el movimiento vertical.

Programación de la plataforma móvil vertical

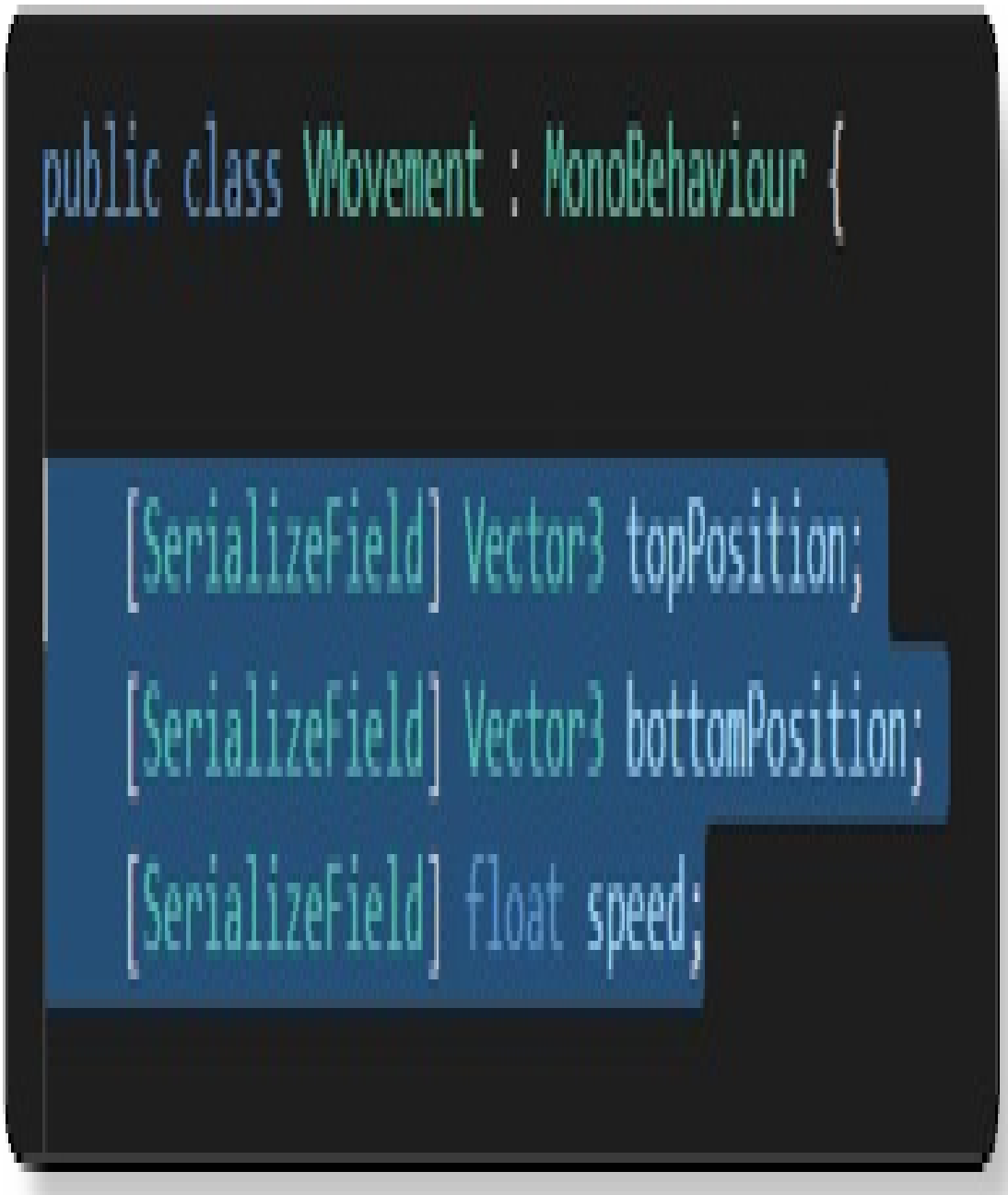
En este apartado nos ocuparemos de la programación del movimiento vertical de la plataforma con un intervalo de tiempo.

Finalizaremos aplicando la programación en el prefab para hacer duplicados y colocar las plataformas móviles en cualquier parte de nuestro juego.

En la ventana de “Project” haga un nuevo folder dentro del folder “Scripts” y nómbrelo “PlatformScripts”.

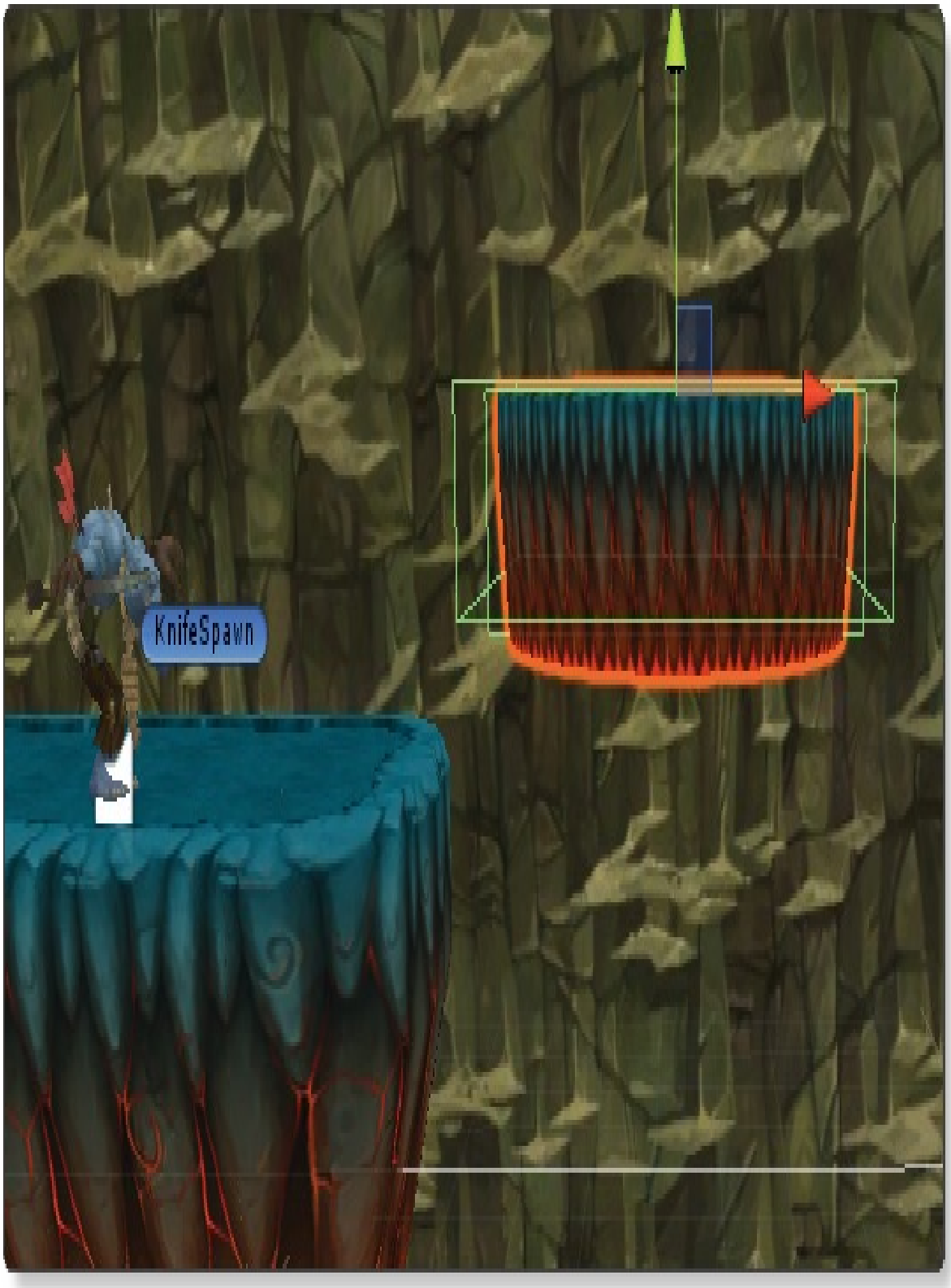
Dentro de PlatformScripts haga un nuevo C#Script y nómbrelo “VMovement” y asígnelo a la plataforma creada en la escena.

Abra el script y escriba las siguientes variables.

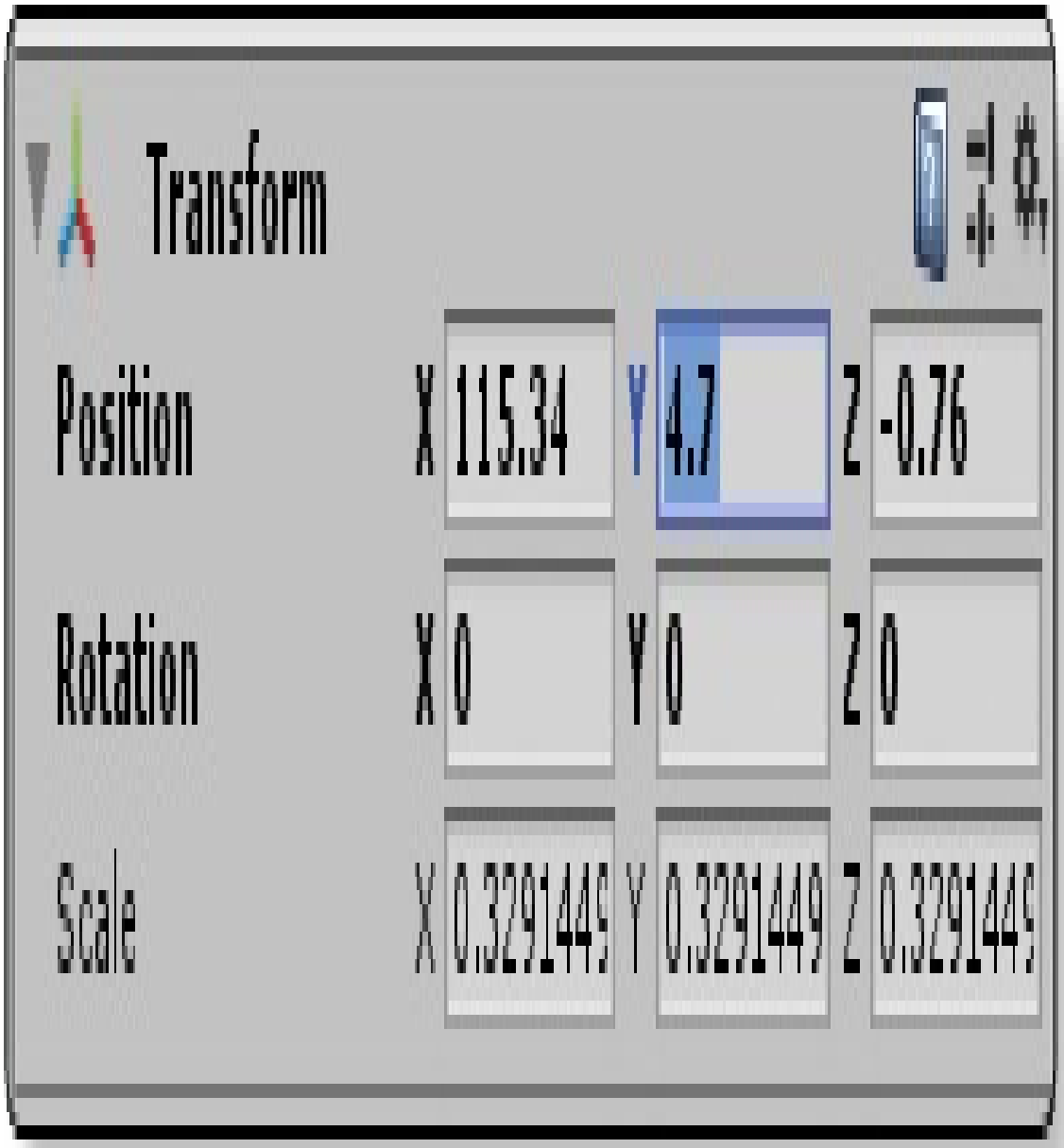

public class VMovement : MonoBehaviour {
[SerializeField] Vector3 topPosition;
[SerializeField] Vector3 bottomPosition;
[SerializeField] float speed;

Variables para el movimiento

Vaya a Unity y mueva la plataforma a la máxima posición que vaya a alcanzar su posición en el eje Y y copie sus datos de Y como se muestra.


KnifeSpawn

Máxima posición en Y de la plataforma


Transform
Position
X 115.34
Y 4.7
Z -0.76
Rotation
X 0
Y 0
Z 0
Scale
X 0.3291449
Y 0.3291449
Z 0.3291449

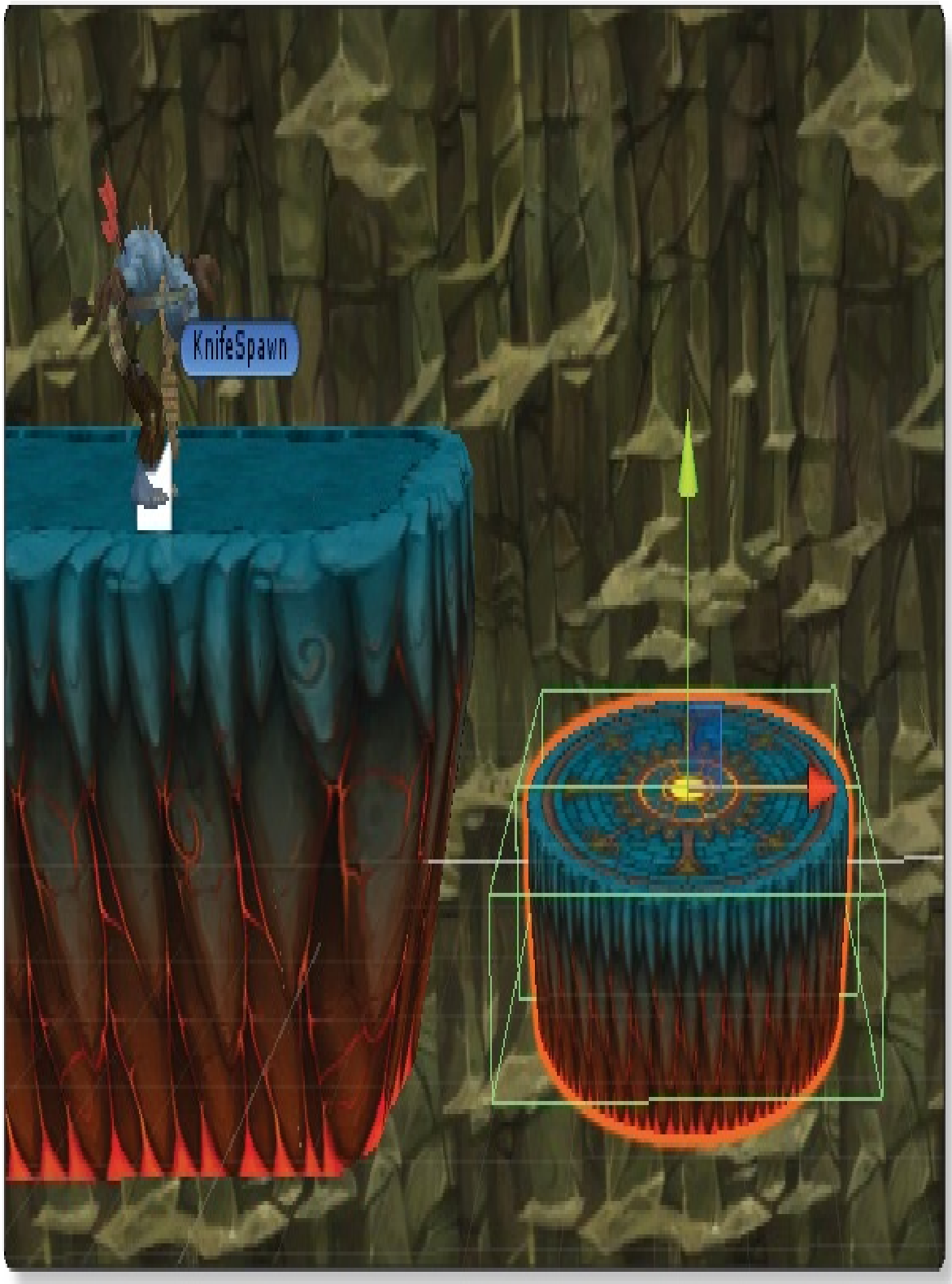
Máxima posición en Y

Asigne la posición tomada en el script en el eje Y de la variable TopPosition.

V Movement (Script)
Script
VMovement
Top Position
X 0
Y 4.7
Z 0
Bottom Position
X 0
Y 0
Z 0
Speed
3

Asignación de máximo valor en Y

Repita el procedimiento para asignar el valor de “BottomPosition” colocando la plataforma en la parte más baja que vaya a alcanzar.


KnifeSpawn

Mínima posición en Y de la plataforma

Transform
Position
X 115.34
Y 0.67
Z -0.76
Rotation
X 0
Y 0
Z 0
Scale
X 0.3291449
Y 0.3291449
Z 0.3291449

Mínima posición en Y

V Movement (Script)
Script
VMovement
Top Position
X 0
Y 4.7
Z 0
Bottom Position
X 0
Y 0.67
Z 0
Speed
3

Asignación de mínimo valor

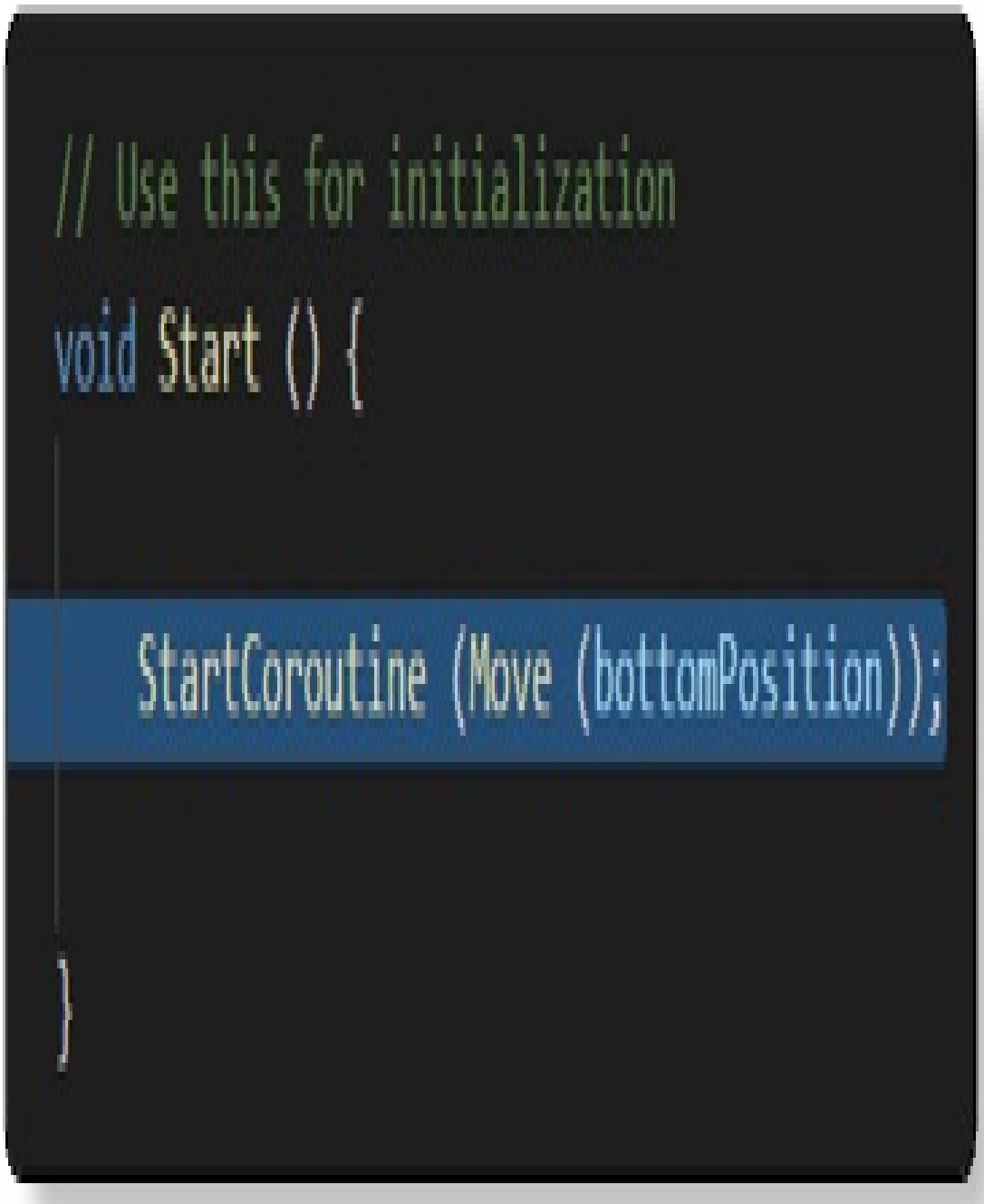
Coloque la plataforma conforme a su “TopPosition”

Vaya al script “VMovement”, elimine la función “Update” y escriba la siguiente “Coroutine” para hacer que la plataforma se mueva de arriba hacia abajo con un intervalo de tiempo de medio segundo.

```
    void Start () {

    }

    IEnumerator Move (Vector3 target) {

        while (Mathf.Abs ((target - transform.localPosition).y) > 0.20f) {

            Vector3 direction = target.y == topPosition.y ? Vector3.up : Vector3.down;
            transform.localPosition += direction * Time.deltaTime * speed;

            yield return null;
        }

        yield return new WaitForSeconds (0.5f);

        Vector3 newTarget = target.y == topPosition.y ? bottomPosition : topPosition;

        StartCoroutine (Move (newTarget));

    }
}
```

Función Move

Mande llamar la función creada en la función “Start”.

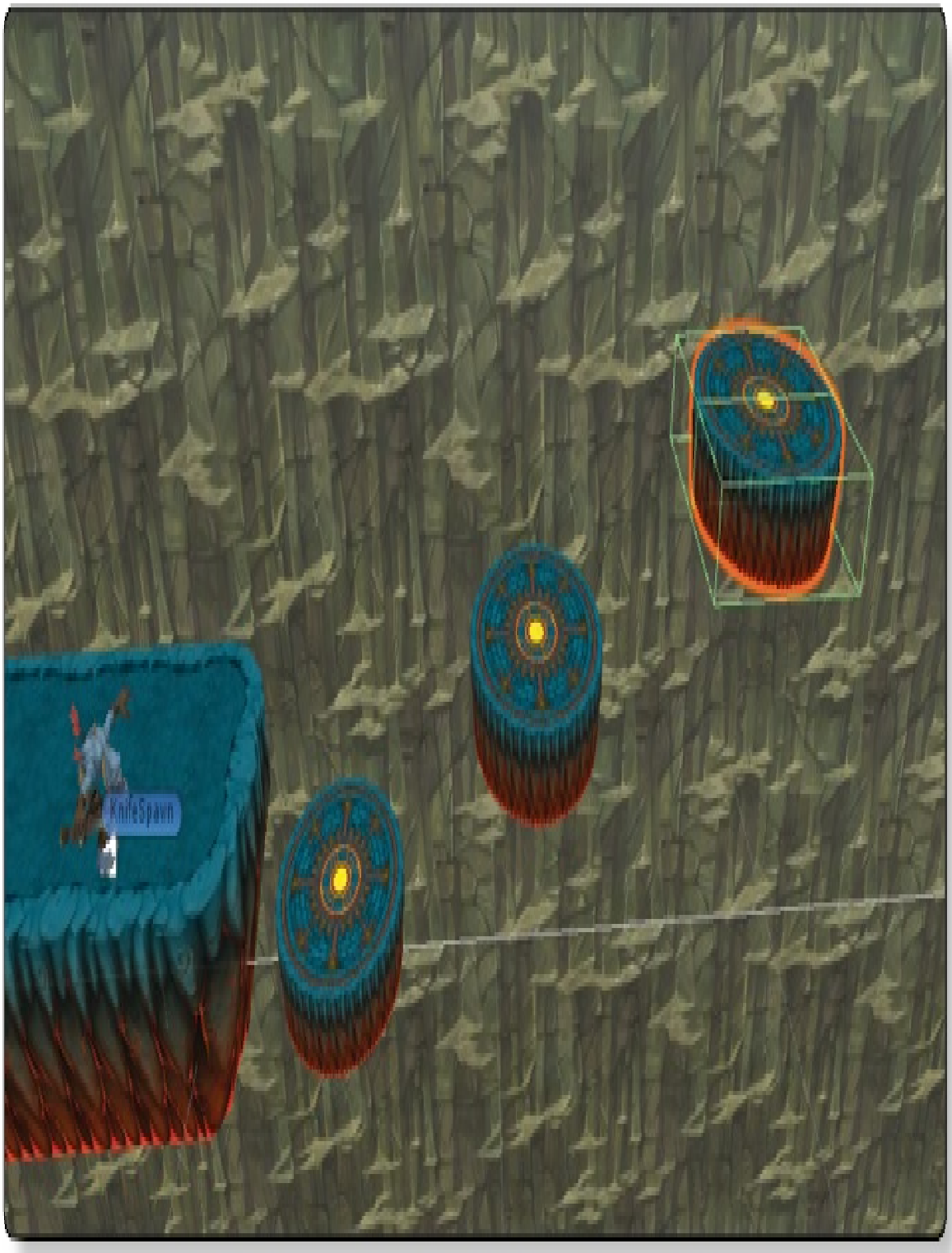

// Use this for initialization
void Start () {
StartCoroutine (Move (bottomPosition));
}

Mandar llamar la función con “bottomPosition” como parámetro

Vaya a Unity, en el script asigne una velocidad de 3 y previsualice el juego para observar el movimiento de la plataforma.

Seleccione la plataforma y en su componente de transformación haga clic en “Apply” para que los cambios se realicen también en el prefab.

Haga 3 duplicados de la plataforma y colóquelos como se muestra.


KnifeSpawn

Duplicado y colocación de plataformas móviles

Previsualice el juego para observar los movimientos de las 3 plataformas.

Previsualización del juego

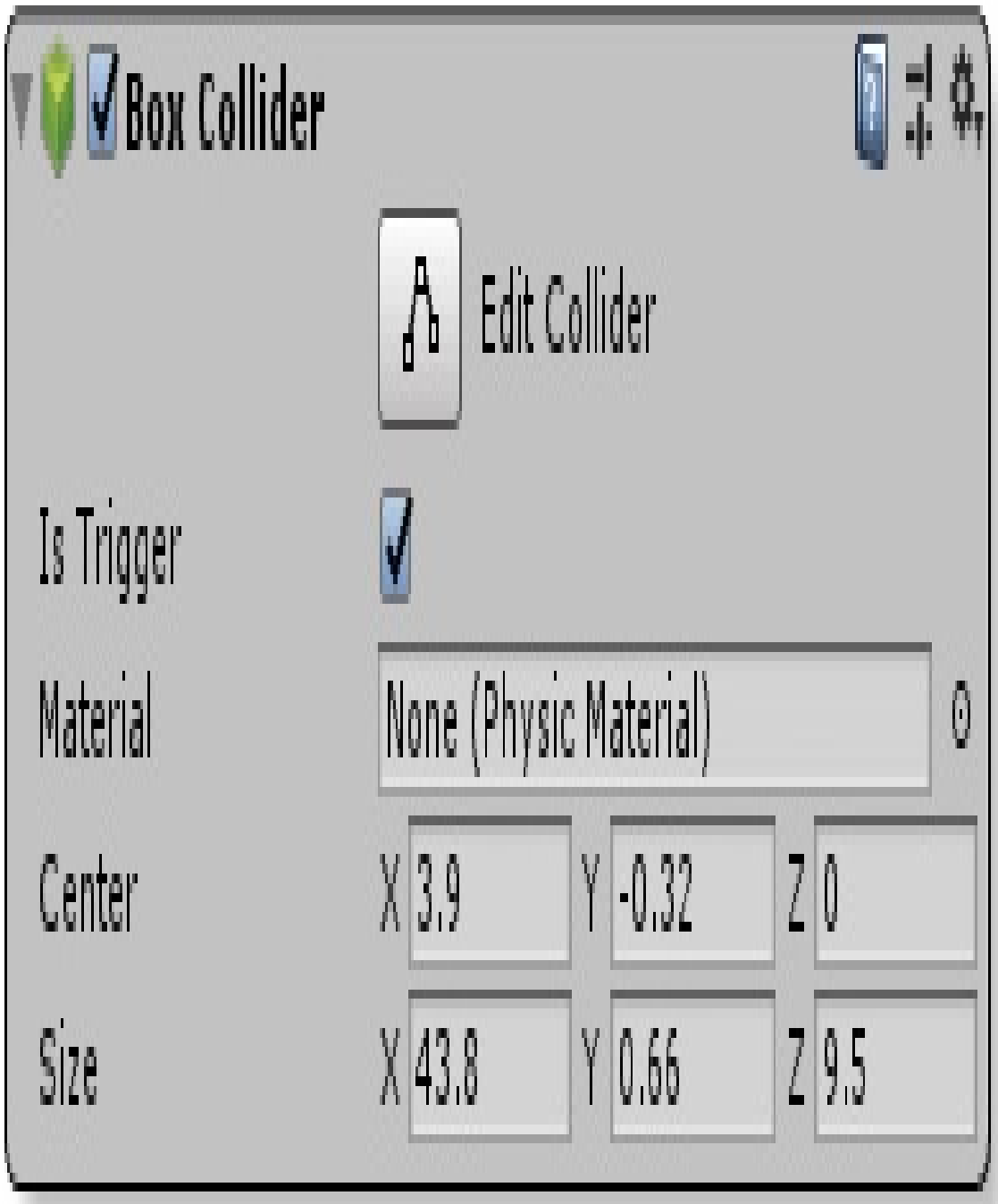
Creación de un “KillBox”

En este apartado crearemos un “KillBox” para que cuando nuestro personaje caiga al vacío pase por dicho “KillBox” y programaremos y mandaremos llamar las funciones correspondientes para asignar su salud a cero cuando pase por él.

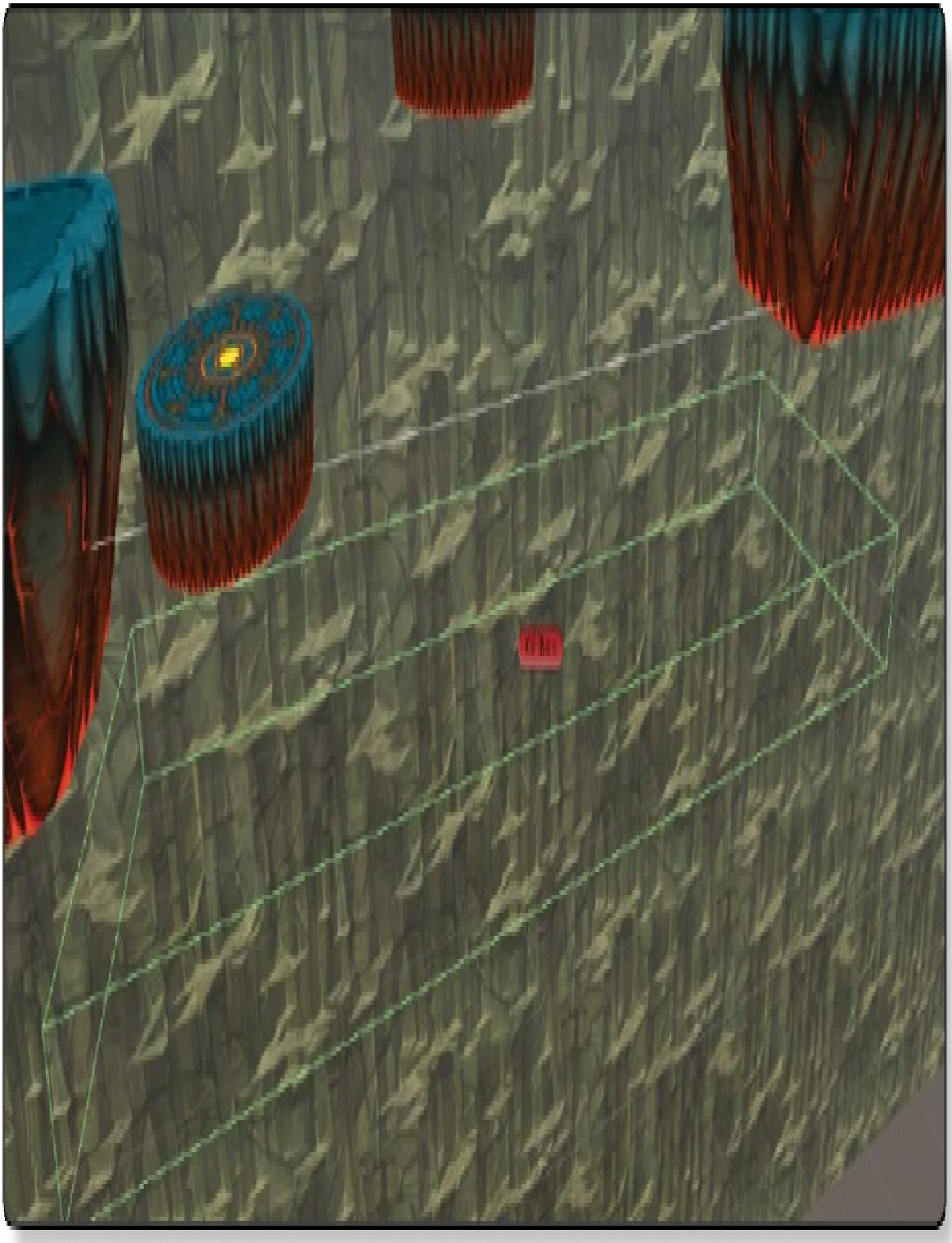
Finalizaremos creando duplicados del prefab del “KillBox” en los lugares donde nuestro “Player” pueda caer al vacío.

Haga un nuevo “Empty GameObject”, renómbrelo “KillBox”, asigne un icono color rojo.

Con el objeto seleccionado asigne un componente “Physics/ Box Collider”, asigne los siguientes valores, asegúrese que sea “Trigger” y colóquelo como se muestra.


Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 3.9
Y -0.32
Z 0
Size
X 43.8
Y 0.66
Z 9.5

Valores del BoxCollider de KillBox



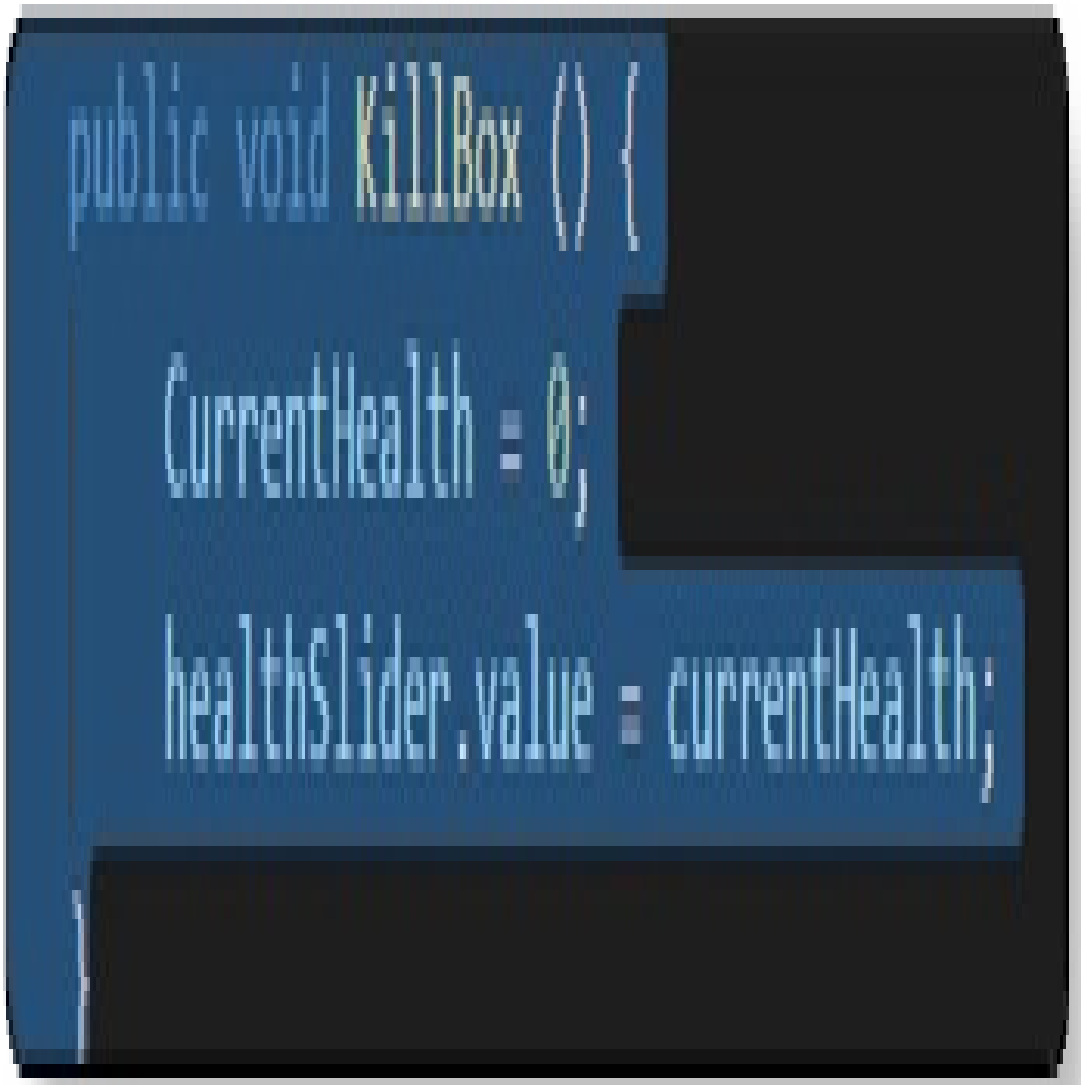
Colocación del KillBox

Haga un nuevo script, nómbrelo “KillBoxScript”, asígnelo al objeto KillBox y escriba el siguiente código:

```
public class KillBoxScript : MonoBehaviour {

    private GameObject player;
    private PlayerHealth playerHealth;

    // Use this for initialization
    void Start () {
        player = GameManager.instance.Player;
        playerHealth = player.GetComponent<PlayerHealth> ();
    }

    void OnTriggerEnter (Collider other) {
        if (other.gameObject == player) {
            playerHealth.KillBox ();
        }
    }
}
```

Script del KillBox

Vaya al script “PlayerHealth” y escriba la siguiente función

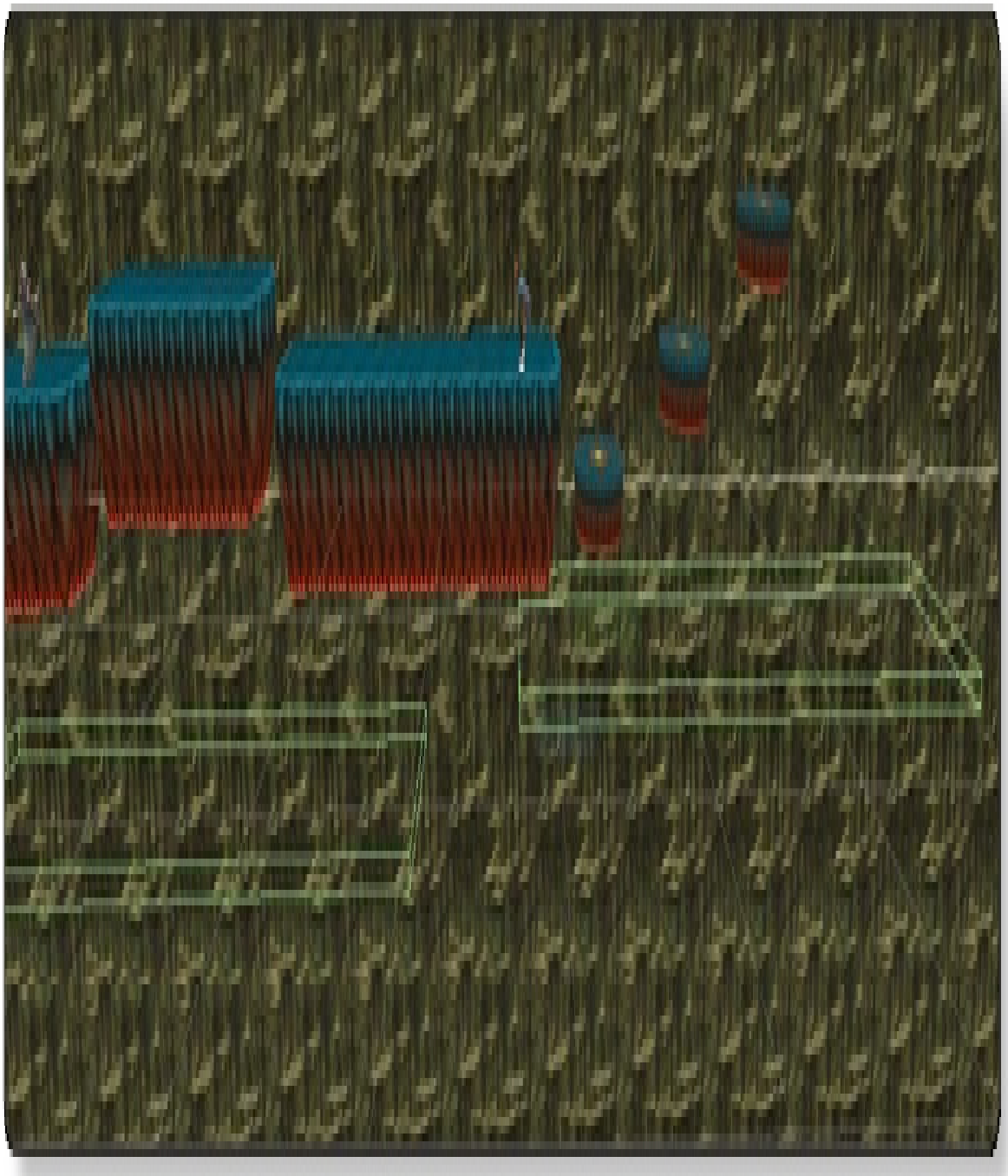

public void KillBox () {
currentHealth = 0;
healthSlider.value = currentHealth;
}

Función KillBox

Previsualice el juego para observar que la salud del Player se convierte en cero cuando pasa por el KillBox.

Convierta el KillBox en Prefab.

Coloque KillBox en los lugares donde el personaje puede caer.



Colocación de duplicados de KillBox

Reaparición del “Player” I

En este apartado crearemos un “Checkpoint” donde va a reaparecer nuestro “Player” cuando muera.

De igual manera programaremos las funciones necesarias para poder realizar la comprobación de que se encuentran funcionando al momento en que la salud del Player llega a cero.

Haga un nuevo “Empty Object”, renómbrelo “Checkpoint”, asigne un label y colóquelo al inicio del nivel.

KnifeSpawn
Checkpoint

Creación de un Checkpoint

Haga un nuevo “Empty Object” y renómbrelo como “Level Manager”.

NOTA:

No importa en donde lo coloque.

Haga un nuevo script y nómbrelo “LevelManager” asígnelo al objeto “LevelManager”.

Haga un nuevo script y nómbrelo “Checkpoints” y asígnelo al objeto “Checkpoint”.

Vaya al script de LevelManager y escriba la siguiente función.

public void RespawnPlayer (){
print ("Player Respawn");
}

Función “RespawnPlayer”

Vaya al script “PlayerHealth”, haga la siguiente variable e inicialícela como se muestra.

private ParticleSystem particleSystem;
public LevelManager levelManager;

Creación de variable

audio = GetComponent<AudioSource>();
levelManager = FindObjectOfType<LevelManager>();

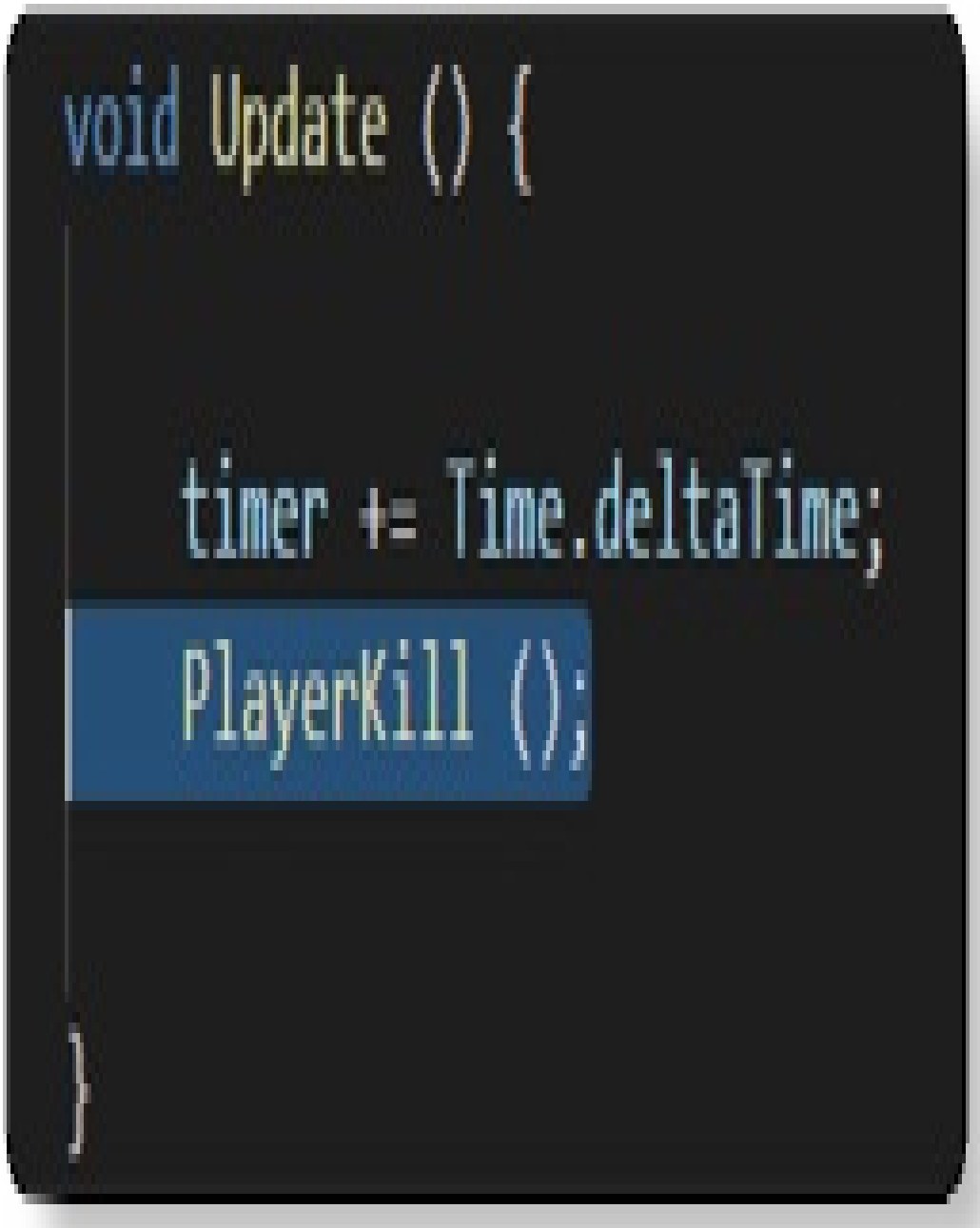
Inicialización de la variable creada

En el mismo script haga la función que se muestra.

public void PlayerKill () {
if (currentHealth <= 0) {
levelManager.RespawnPlayer ();
}
}

Función “Playerilla”

Mande llamar la función creada en “Update”.


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

Llamar función Playerilla

Previsualice el juego para observar que cuando el Player cae al vacío, se imprime en la consola “Player Respawn”.

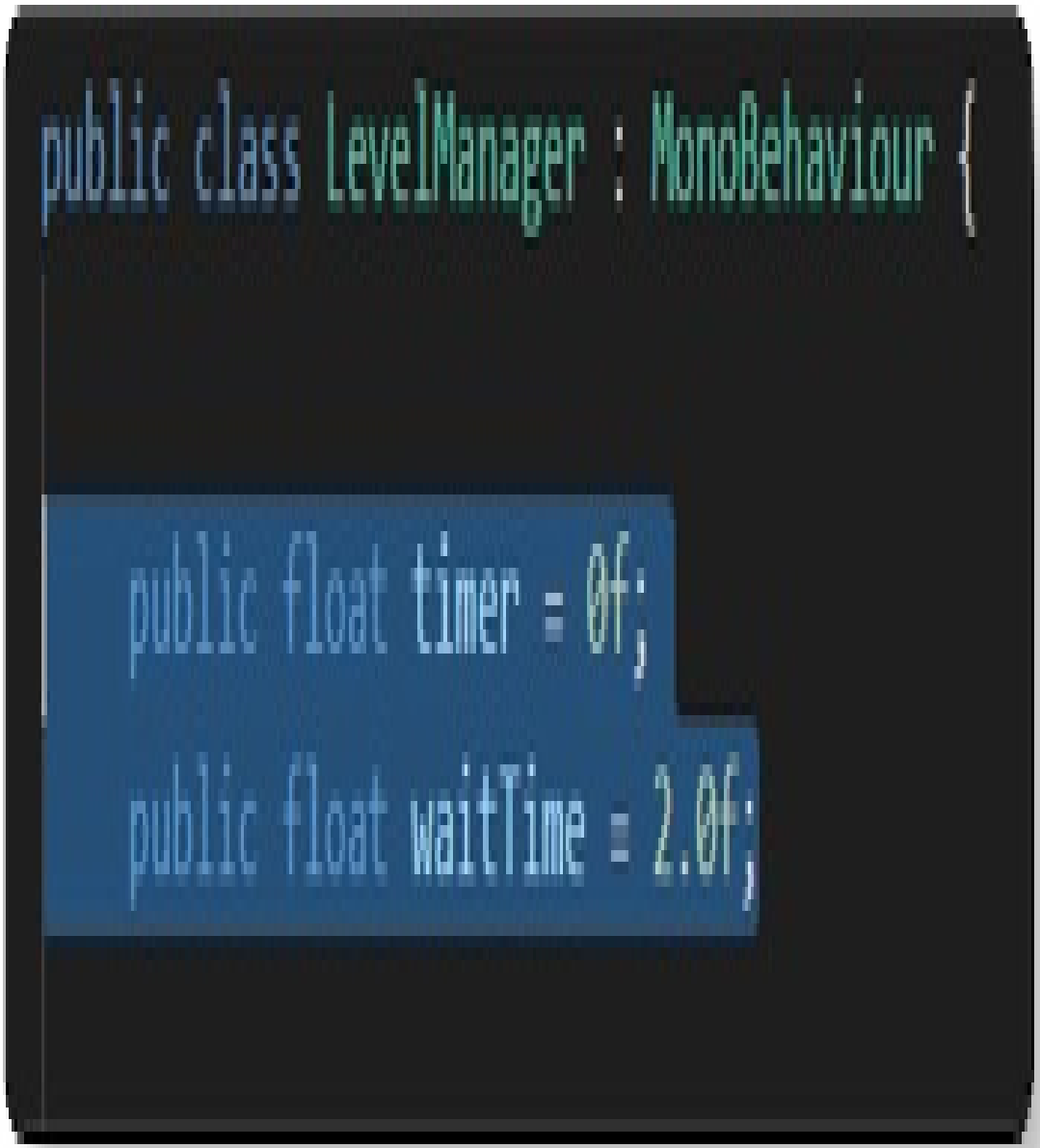
Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
Player in range: False
UnityEngine.MonoBehaviour:print(Object)
Player Respawn
UnityEngine.MonoBehaviour:print(Object)

Comprobación en Consola

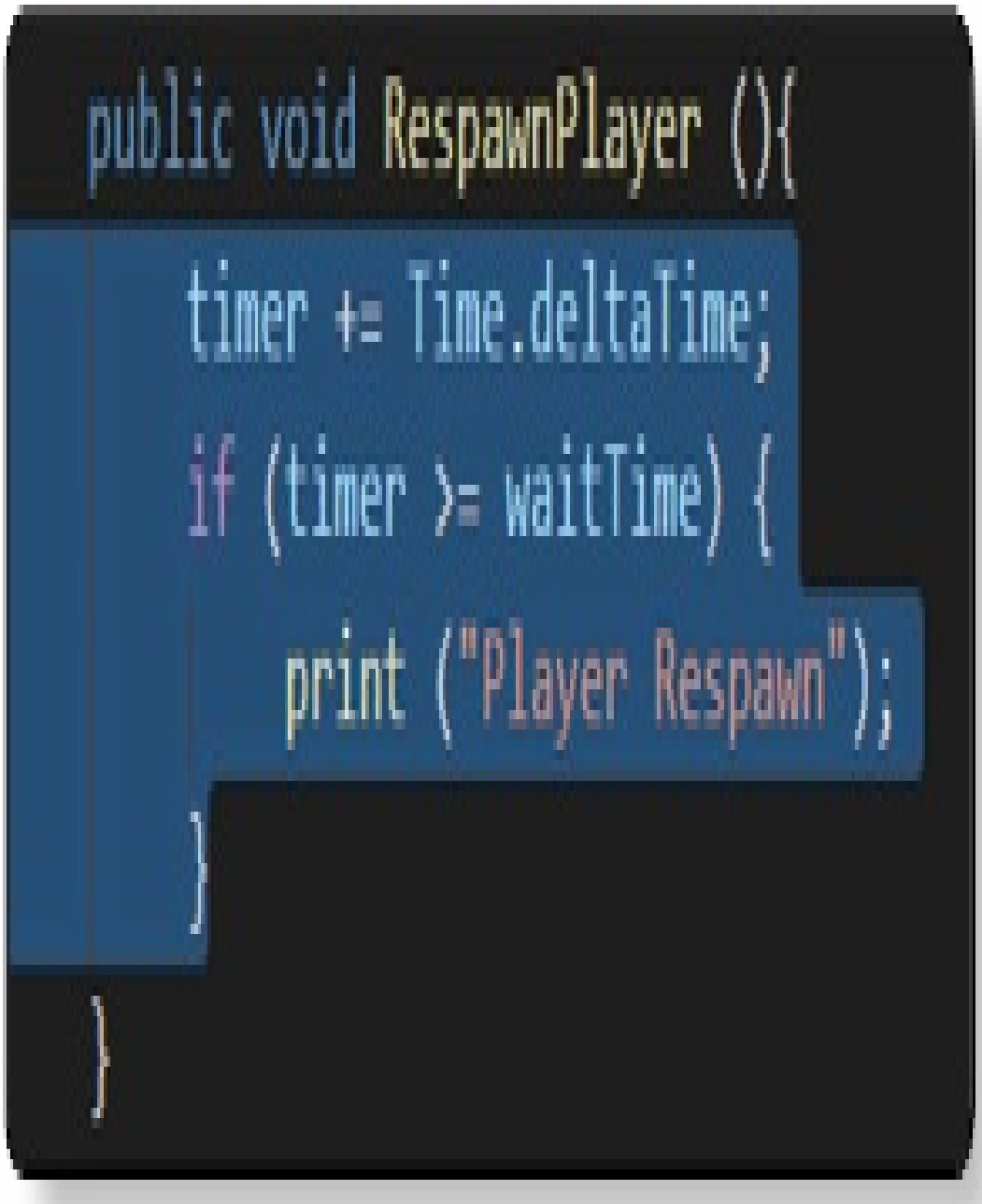
Reaparición del “Player” II

En este apartado programaremos la funcionalidad de reaparición del “Player” en la ubicación de nuestro “CheckPoint”.

Para colocar un intervalo de tiempo para que corra la función “RespawnPlayer”, vaya al script “LevelManager” y escriba las variables que se muestran y el código que se indica en la función “RespawnPlayer”.


public class LevelManager : MonoBehaviour {
public float timer = 0f;
public float waitTime = 2.0f;

Creación de variables


public void RespawnPlayer (){
timer += Time.deltaTime;
if (timer >= waitTime) {
print ("Player Respawn");
}
}

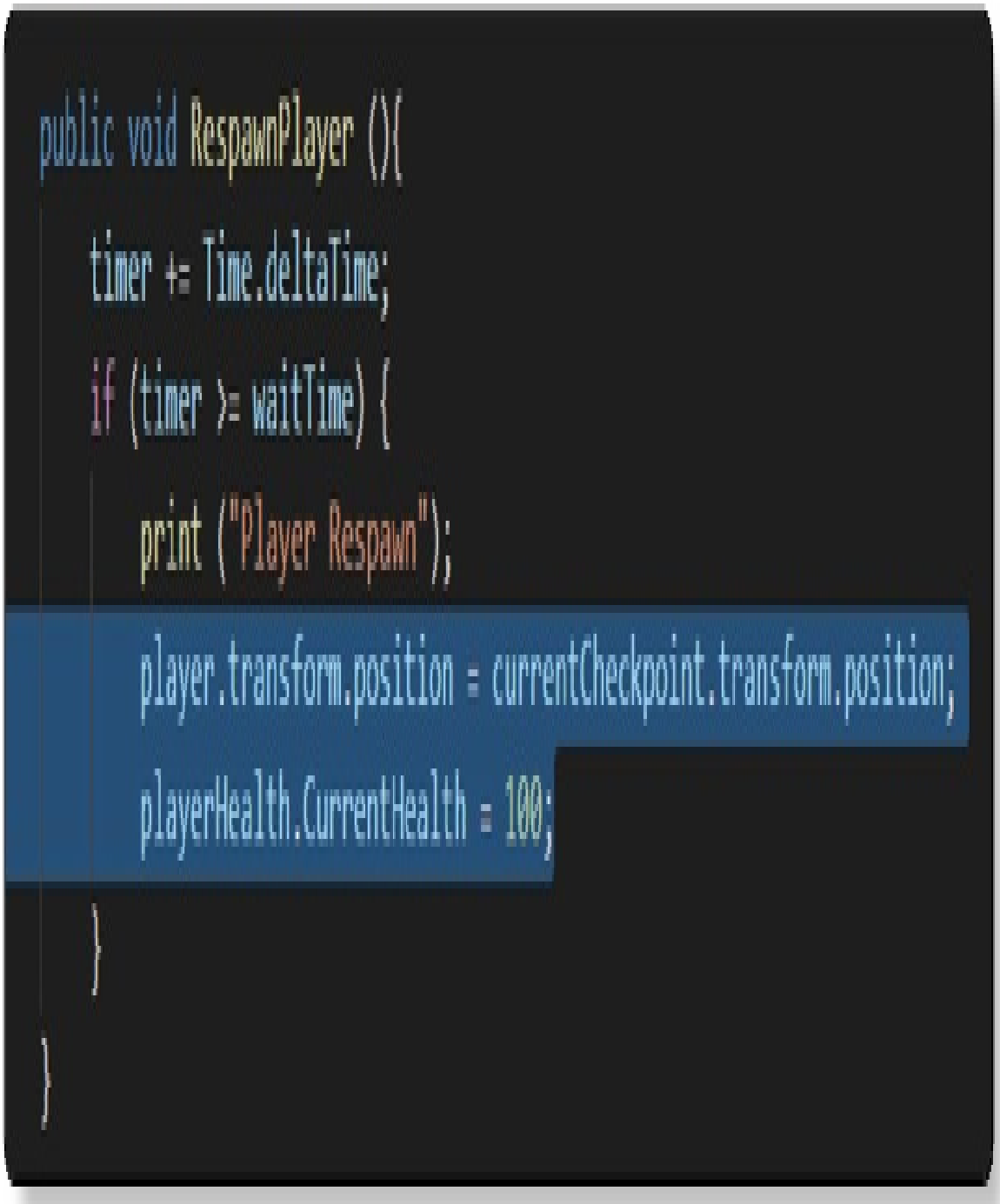
Modificación de función

Para hacer que nuestro “Player” reaparezca en el “Checkpoint”, vaya al script “LevelManager”, escriba e inicialice las variables que se muestran.

```
public float timer = 0f;
public float waitTime = 2.0f;

public GameObject currentCheckpoint;
private GameObject player;
private PlayerHealth playerHealth;

// Use this for initialization
void Start () {

    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealth> ();
```

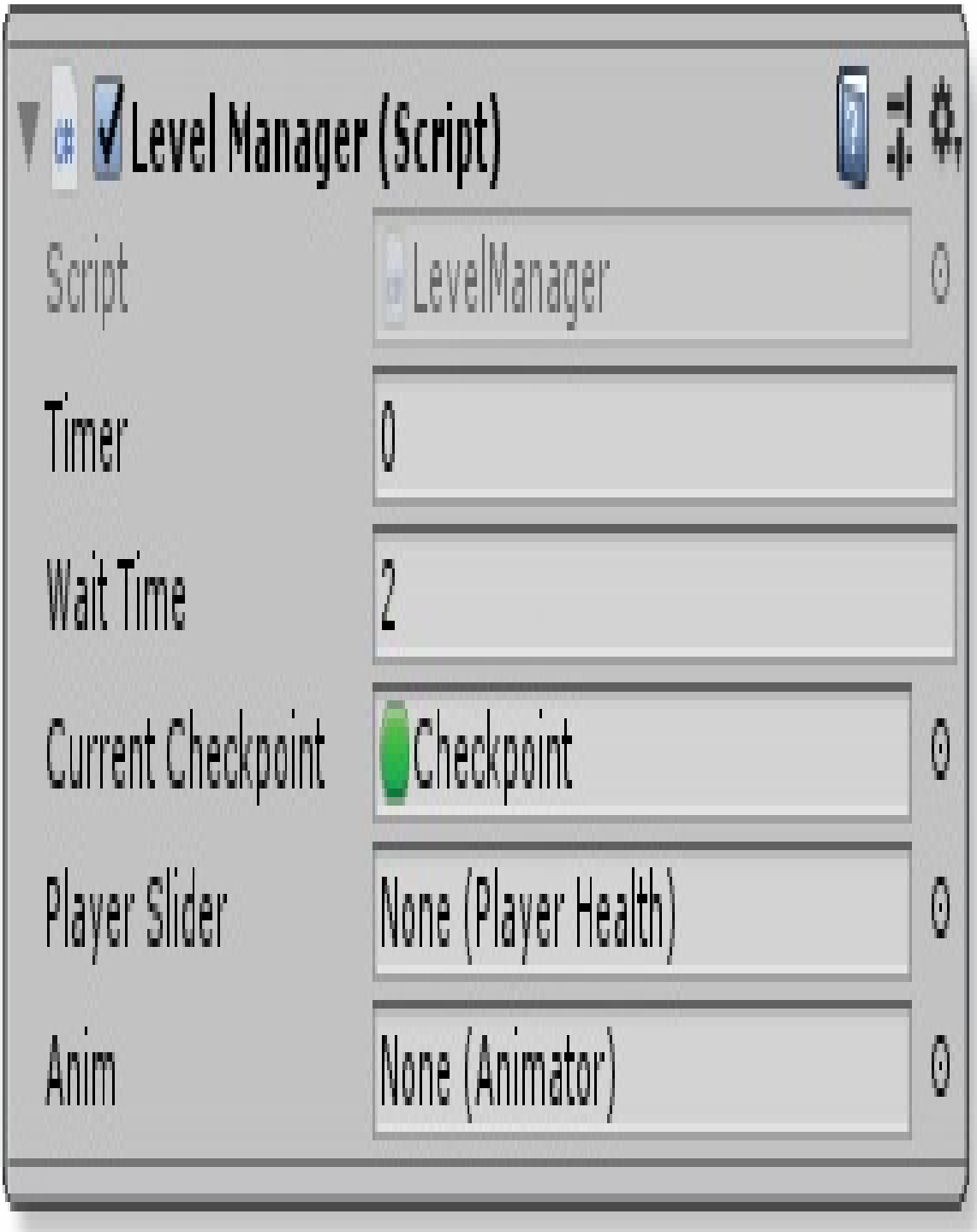
Creación e inicialización de variables

En el mismo script, añada las líneas de código que se muestran en la función “RespawnPlayer”.


public void RespawnPlayer (){
timer += Time.deltaTime;
if (timer >= waitTime) {
print ("Player Respawn");
player.transform.position = currentCheckpoint.transform.position;
playerHealth.CurrentHealth = 100;
}
}

Inserción de funcionalidad en la función

Vaya a Unity, seleccione el objeto “LevelManager” y asigne el “Checkpoint” en el script como se muestra


Level Manager (Script)
Script
LevelManager
Timer
0
Wait Time
2
Current Checkpoint
Checkpoint
Player Slider
None (Player Health)
Anim
None (Animator)

Previsualice el juego para observar que cuando el “Player” cae por el vacío reaparece en el Checkpoint.

Previsualización de reaparición

NOTA:

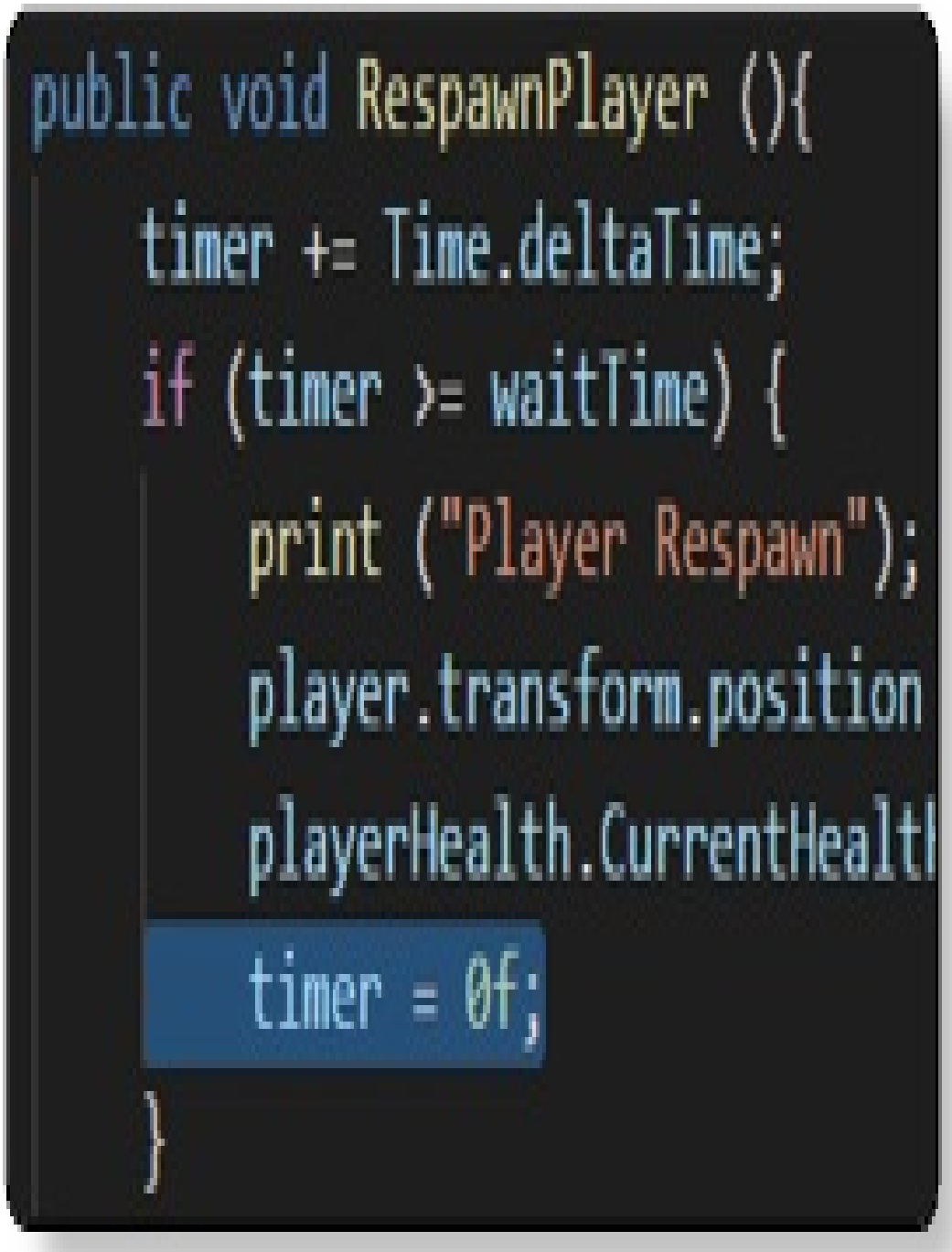
Nuestro “Player” aparece en el “Checkpoint” pero todavía tenemos algunos “bugs” que reparar, lo cual lo haremos en el siguiente apartado.

Reaparición del “Player” III

En este apartado nos ocuparemos de los “bugs” o problemáticas que surgen al reaparecer nuestro “Player”.

PROBLEMA 1: Al parecer todo funciona bien después de su primera muerte en el vacío, pero al morir de la misma forma la segunda vez, el Player reaparece en el “Checkpoin” sin esperar el intervalo de tiempo de la variable “waitTime”.

SOLUCIÓN: Vaya al script “LevelManager” y añada la siguiente línea de código en la función “RespawnPlayer”.


public void RespawnPlayer (){
timer += Time.deltaTime;
if (timer >= waitTime) {
print ("Player Respawn");
player.transform.position
playerHealth.CurrentHealt
timer = 0f;
}

Reset del tiempo

Al hacer esto estamos reiniciando nuestra variable “timer” a cero cuando corra la función

Previsualice el juego para comprobar la solución del problema.

PROBLEMA 2: La barra de “healthSlider” del Player no se actualiza y al momento de la reaparición sigue en blanco.

SOLUCIÓN: Vaya al script de “PlayerHealth” y escriba el siguiente “Getter”.

public Slider HealthSlider {
get {return healthSlider; }
}

Creación de Getter

Vaya al script de “LevelManager”, escriba e inicialice la variable que se muestra.

```
private PlayerHealth playerHealth;

public PlayerHealth playerSlider;

// Use this for initialization
void Start () {

    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealth> ();
    playerSlider = player.GetComponent<PlayerHealth> ();
```

Creación e inicialización de variable

Vaya a la función “RespawnPlayer” y añada el siguiente código para que se actualice la barra de salud de Player.

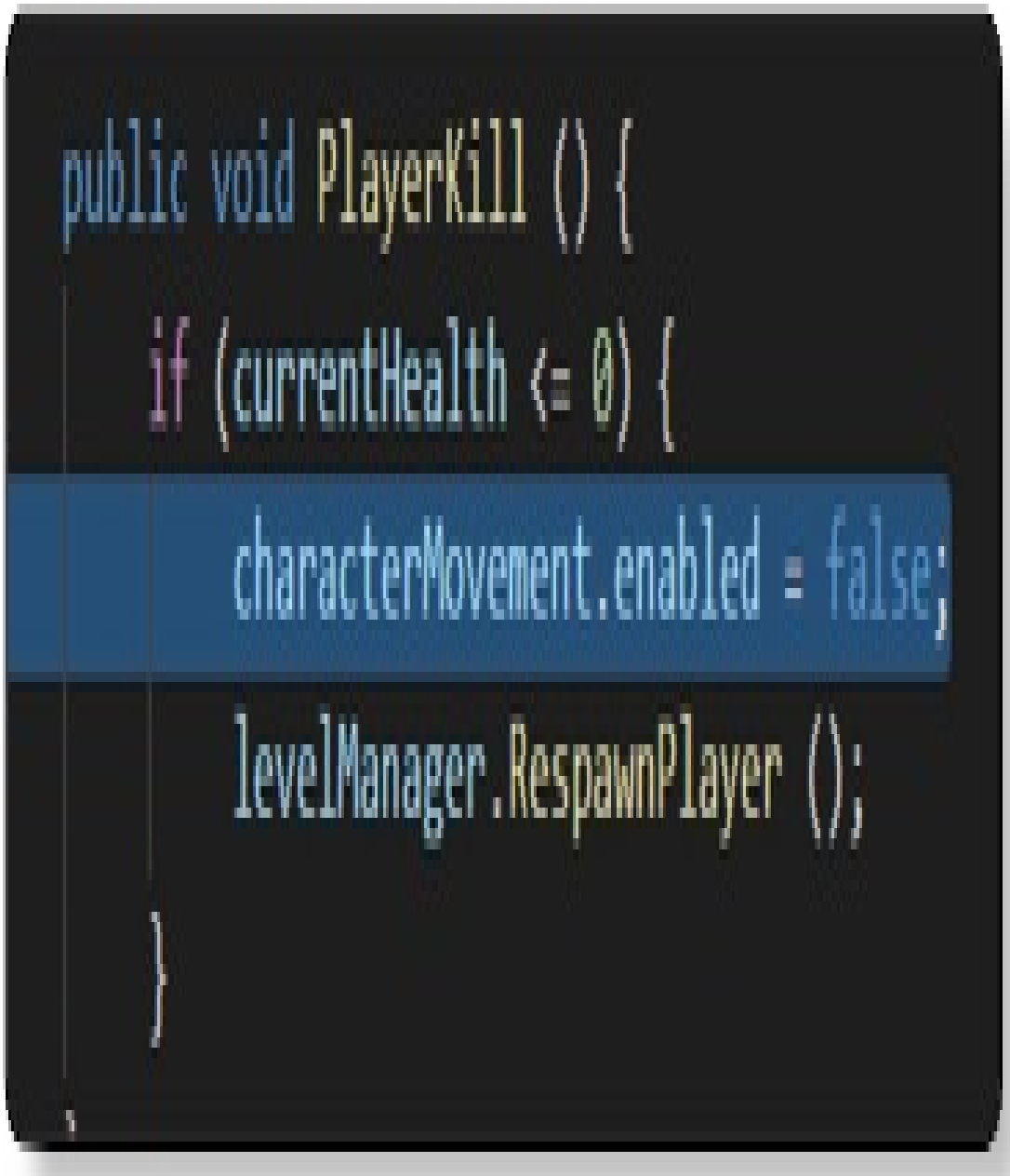
```
public void RespawnPlayer (){
    timer += Time.deltaTime;
    if (timer >= waitTime) {
        print ("Player Respawn");
        player.transform.position = currentCheckpoint.transform.position
        playerHealth.CurrentHealth = 100;
        timer = 0f;
        playerHealth.HealthSlider.value = playerHealth.CurrentHealth;
    }
}
```

Actualización de Slider

Previsualice el juego para observar la solución del problema.

PROBLEMA 3: Cuando muere nuestro Player, su movimiento sigue activo, debemos de deshabilitar su movimiento.

SOLUCIÓN: Vaya al script de PlayerHealth y escriba el código que se muestra en la función “Playerilla.


public void PlayerKill () {
if (currentHealth <= 0) {
characterMovement.enabled = false;
levelManager.RespawnPlayer ();
}
}

Deshabilitación del movimiento del Player

A continuación, debemos de reactivar el movimiento del “Player” en el momento de su reaparición.

Vaya al script “LevelManager”, haga la siguiente variable e inicialícela.

```
public PlayerHealth playerSlider;
private CharacterMovement characterMovement;

// Use this for initialization
void Start () {

    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealth> ();
    playerSlider = player.GetComponent<PlayerHealth> ();
    characterMovement = player.GetComponent<CharacterMovement> ();
```

Creación e inicialización de variable

Vaya a la función “RespawnPlayer” y escriba el siguiente código para reactivar la movilidad del Player.

```
public void RespawnPlayer (){
    timer += Time.deltaTime;
    if (timer >= waitTime) {
        print ("Player Respawn");
        player.transform.position = currentCh
        playerHealth.CurrentHealth = 100;
        timer = 0f;
        playerHealth.HealthSlider.value = pla
        characterMovement.enabled = true;
    }
```

Rehabilitación de movimiento del Player

Previsualice el juego para comprobar la solución del problema.

PROBLEMA 4: Cuando nuestro Player muere de otra forma que no sea el KillBox (por ejemplo, un proyectil), cuando reaparece aparece con su animación de muerte.

Animación de la muerte del Player al reaparecer

El “Player” se sigue moviendo, aunque aparezca con su animación de muerte.

SOLUCIÓN: Vaya al script “LevelManager”, escriba e inicialice la variable que se muestra.

```
private CharacterMovement characterMovement;

public Animator anim;

// Use this for initialization
void Start () {

    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealt
    playerSlider = player.GetComponent<PlayerHealt
    characterMovement = player.GetComponent<Charac
    anim = player.GetComponent<Animator> ();
}
```

Creación e inicialización de variable

Vaya a la función “RespawnPlayer” y escriba el siguiente código para mandar llamar el “BlendTree” de la animación del “Player” al momento de su reaparición.

```
public void RespawnPlayer (){
    timer += Time.deltaTime;
    if (timer >= waitTime) {
        print ("Player Respawn");
        player.transform.position = current
        playerHealth.CurrentHealth = 100;
        timer = 0f;
        playerHealth.HealthSlider.value = p
        characterMovement.enabled = true;
        anim.Play ("Blend Tree");
    }
}
```

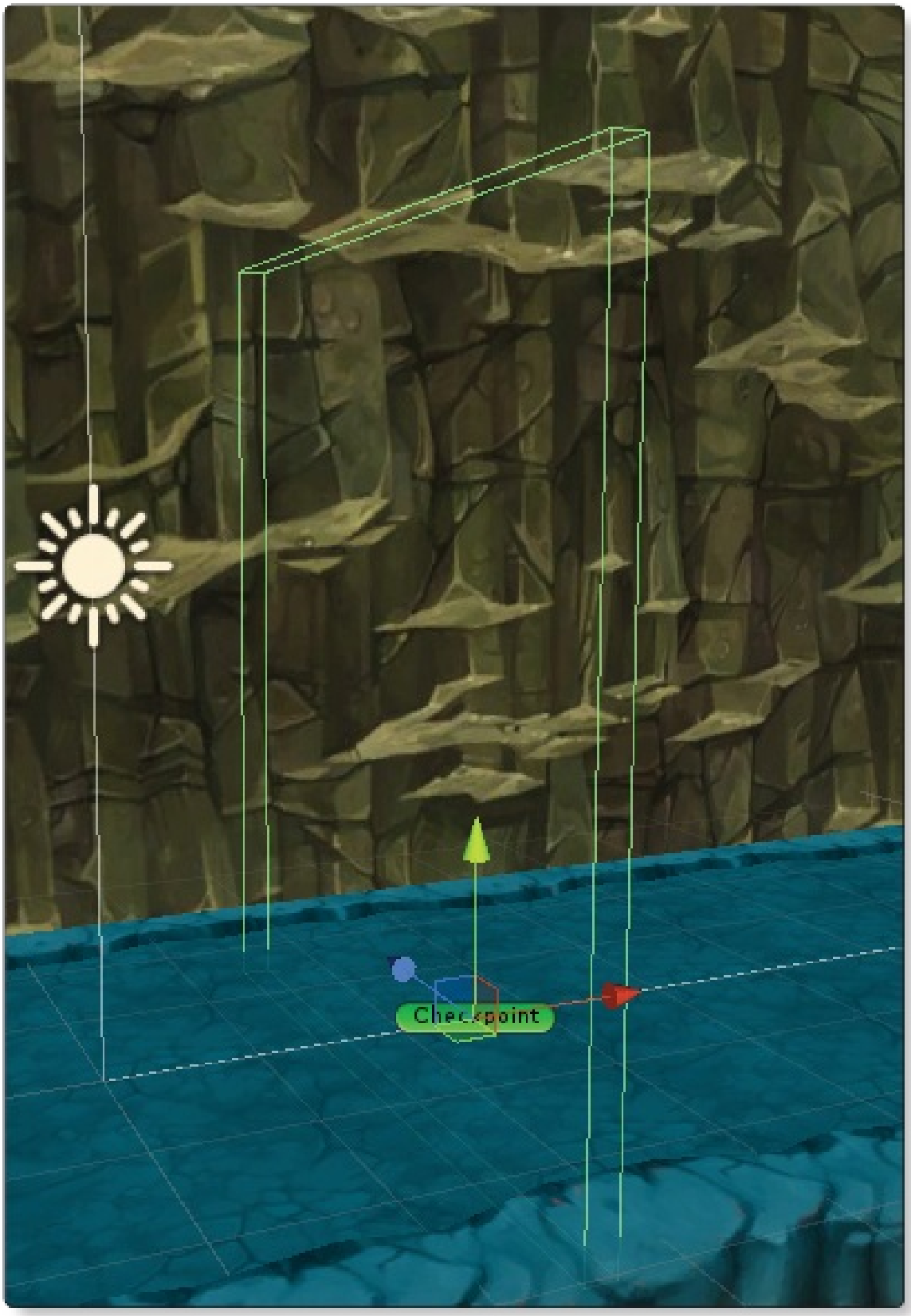
Correr la animación de BlendTree del Player

Previsualice el juego para comprobar la solución del problema.

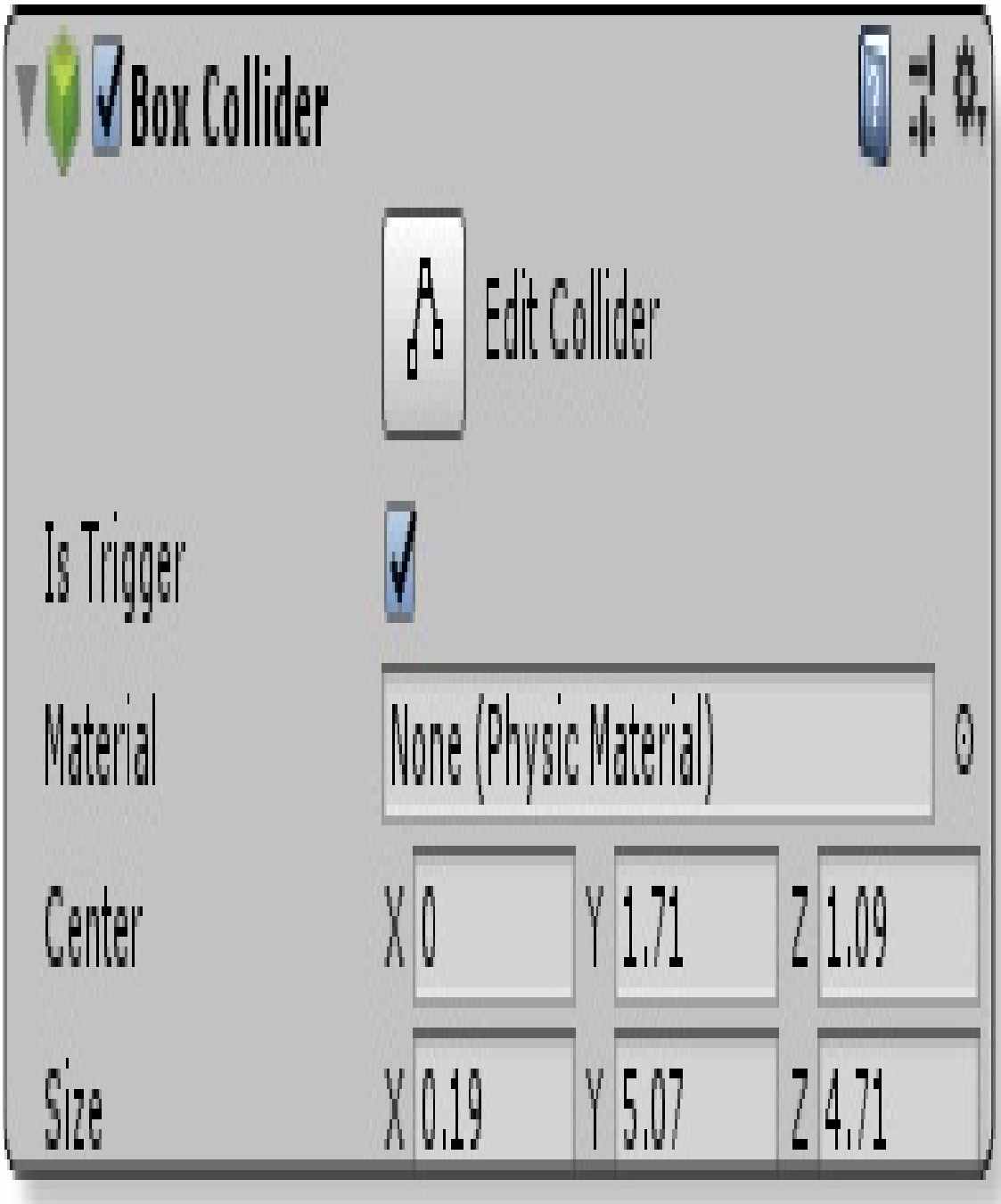
Programación de más Checkpoints

En este apartado colocaremos más Checkpoints dependiendo del lugar por donde nuestro Player vaya avanzando y los programaremos para que pueda reaparecer en ellos conforme a que haya pasado a través de los mismos.

Seleccione el objeto “Checkpoint” y asigne un componente “BoxCollider”, active su atributo “IsTrigger” y asigne las dimensiones que se muestran.


Checkpoint

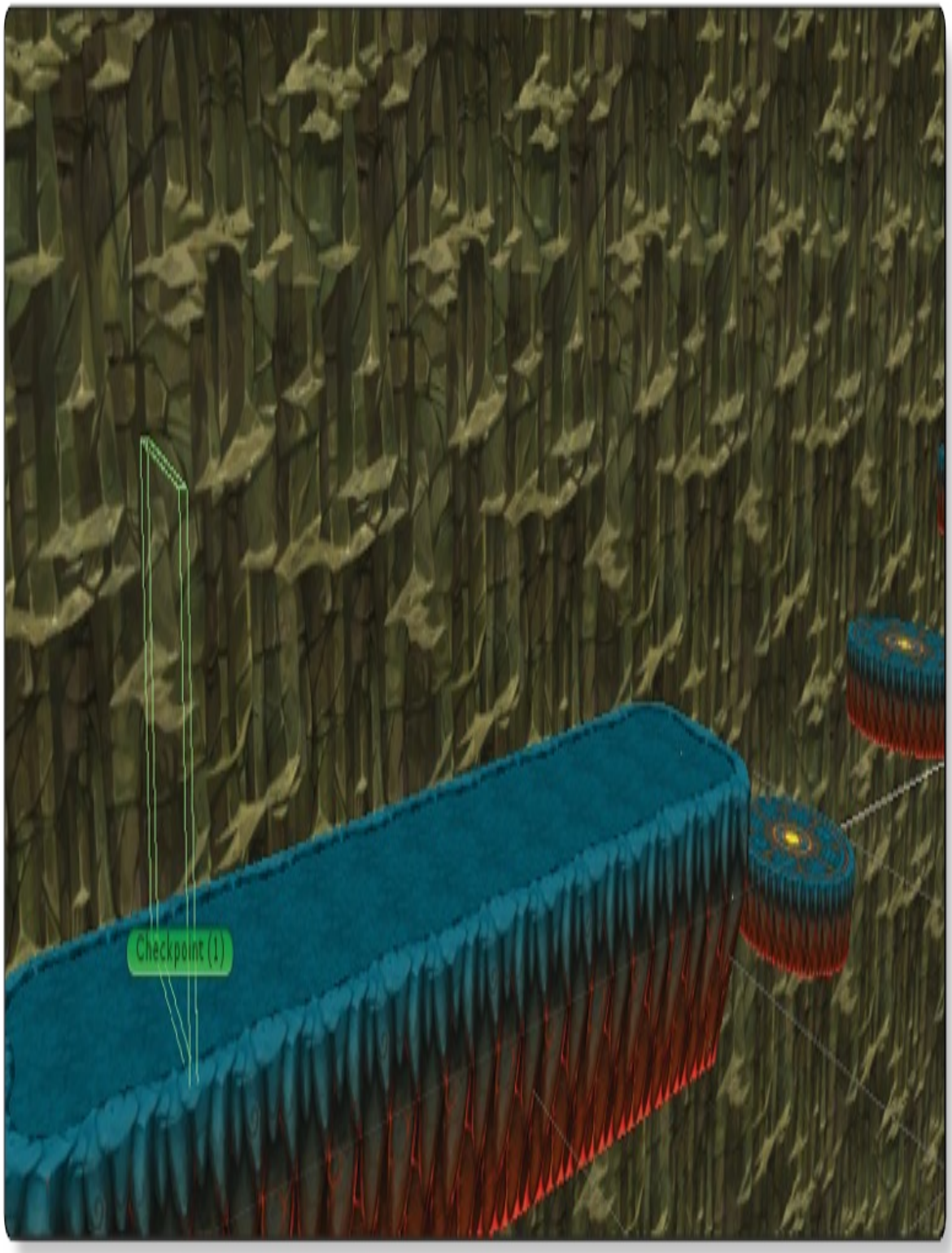
BoxCollider del Checkpoint


Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 1.71
Z 1.09
Size
X 0.19
Y 5.07
Z 4.71

Valores del BoxCollider

Convierta el objeto “Checkpoint” a Prefab.

Coloque el prefab “Checkpoint” antes de las plataformas móviles.


Checkpoint (1)

Colocación del segundo Checkpoint

Vaya al script “Checkpoints” y escriba el siguiente código

```
public class Checkpoints : MonoBehaviour {

    public LevelManager levelManager;

    // Use this for initialization
    void Start () {

        levelManager = FindObjectOfType<LevelManager> ();
    }

    void OnTriggerEnter (Collider other) {

        if (other.name == "Player") {

            levelManager.currentCheckpoint = gameObject;
        }
    }
}
```

Script Checkpoints

Previsualice el juego para comprobar que cuando nuestro “Player” muerte después de pasar por el segundo “Checkpoint”, este reaparece en el segundo “Checkpoint”.

Creación de un sistema de vidas para “Player”

En este apartado crearemos un sistema de vidas para nuestro Player, mismo que será visualizado con elementos GUI (Graphic User Interfase).

También el estudiante podrá sumergirse más a fondo en la interacción de clases para utilizar funciones de otros scripts.

Se verán los contadores para añadir o restar vidas y se creará un estado booleano para indicar si al “Player” le quedan vidas.

En “Hierarchy” asigne clic derecho sobre el objeto GUI y haga los siguientes elementos.

Nuevo EmptyObject y renómbrelo “LifeSystem”

Dentro de “LifeSystem” haga un “UI/ Image”

Dentro de “LifeSystem” haga un “UI/ Text”

GameManager
GUI
HealthBar
LifeSystem
Image
Text
EventSystem

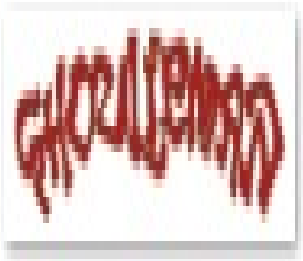
Creación de objetos GUI

Importe del folder de “resources” del capítulo la imagen “ghoulieLife” a la carpeta de “Sprites” dentro de Unity.

DO VIDEOJUEGOS 2017 › MODULO 3 (udemy) › LIBRO › CA

52085

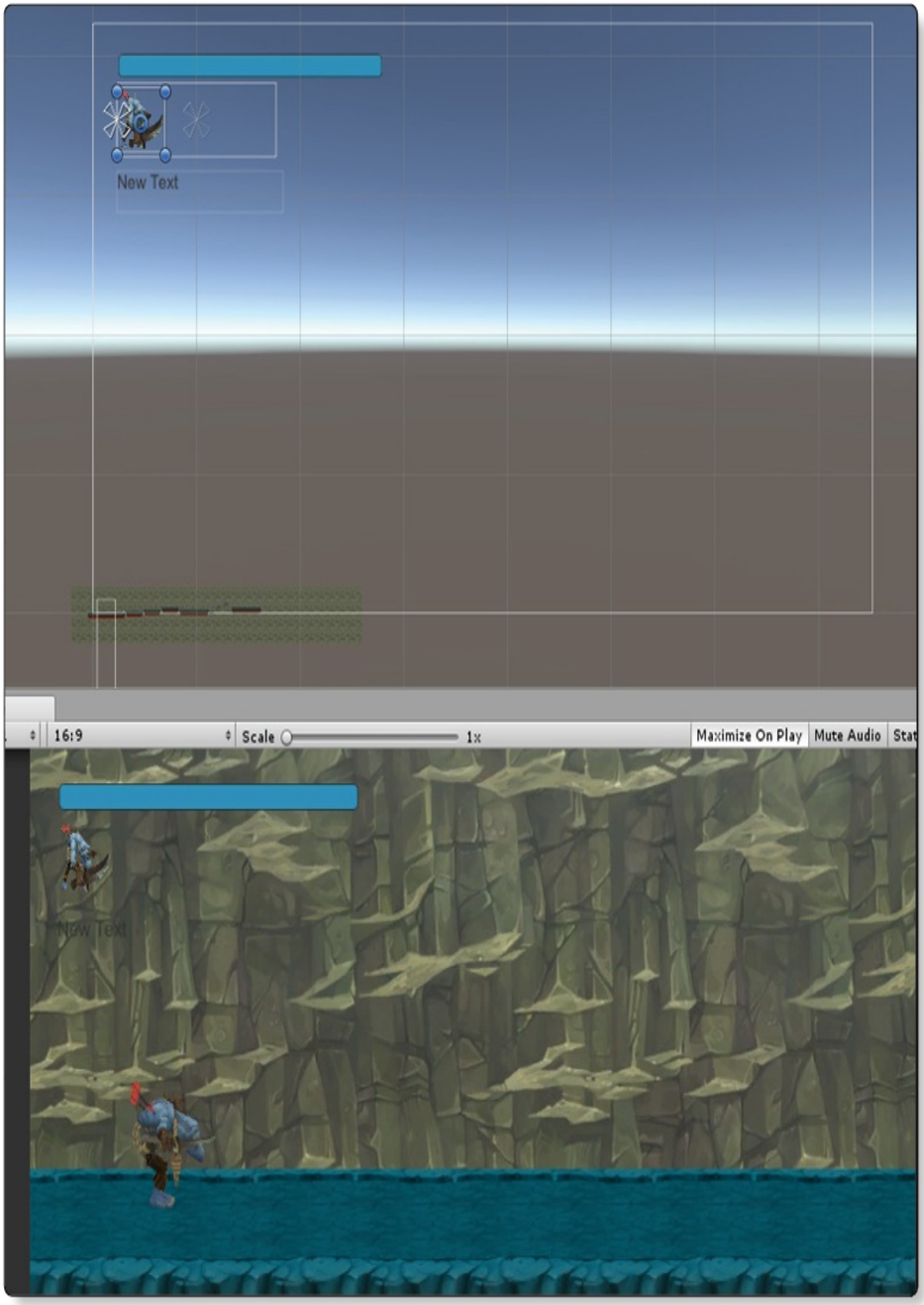
ghoulieLife



ghoulieMan

Gráfico para las vidas

Conviértala en Sprite2D, asígnela en el componente “Image” y colóquela como se muestra.

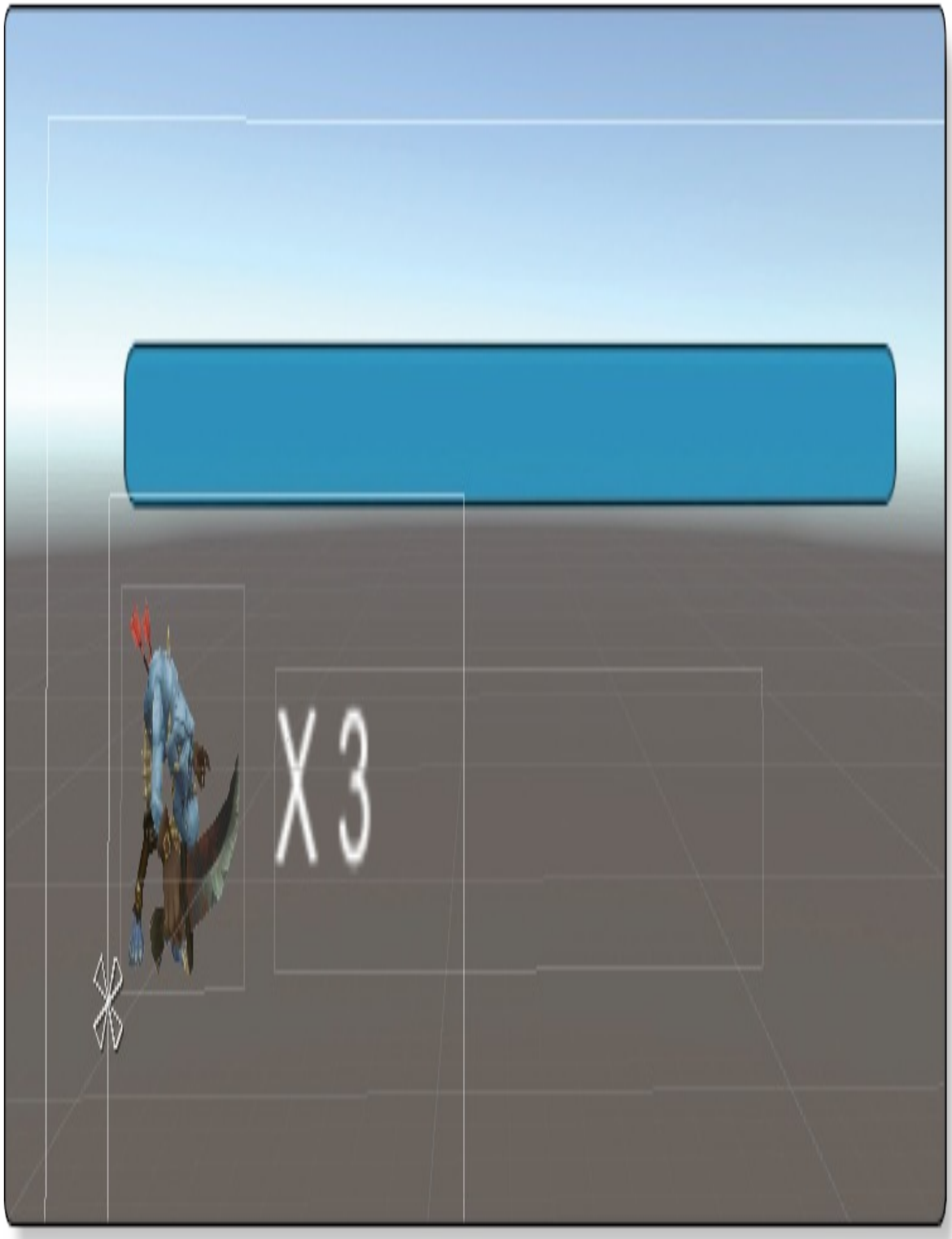

New Text
16:9
Scale
1x
Maximize On Play
Mute Audio
Stat
New Text

Colocación de los elementos GUI

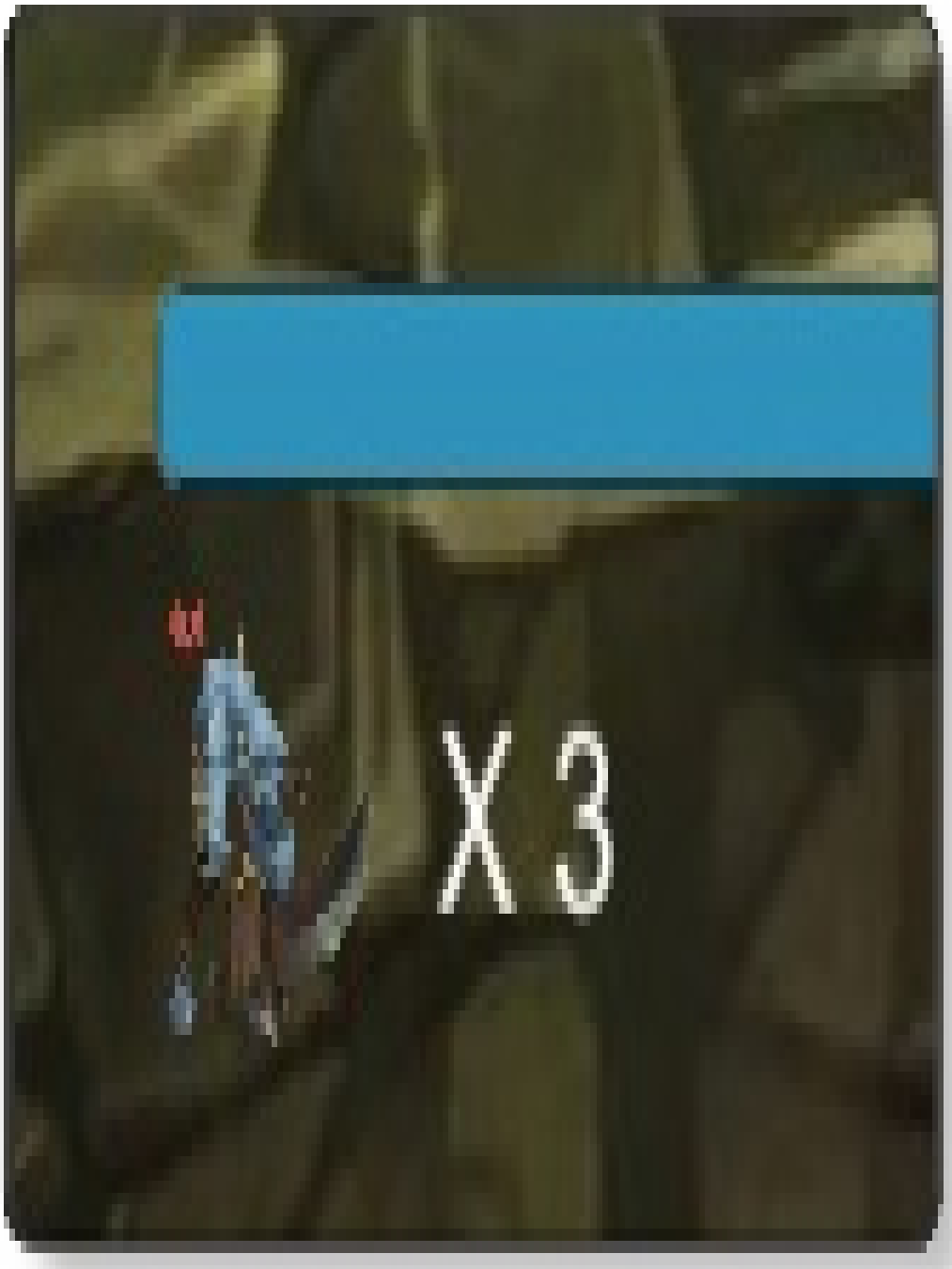
Seleccione el elemento “Text” y en “Inspector” asigne los valores que se muestran para formatear el texto.

Text (Script)
Text
X 3
Character
Font
Arial
Font Style
Normal
Font Size
21
Line Spacing
1
Rich Text
Paragraph
Alignment
Align By Geometry
Horizontal Overflow
Wrap
Vertical Overflow
Truncate
Best Fit
Color
Material
None (Material)
Raycast Target

Valores para “Text”


X 3

Formateo del texto


X3

Previsualización del texto en Game

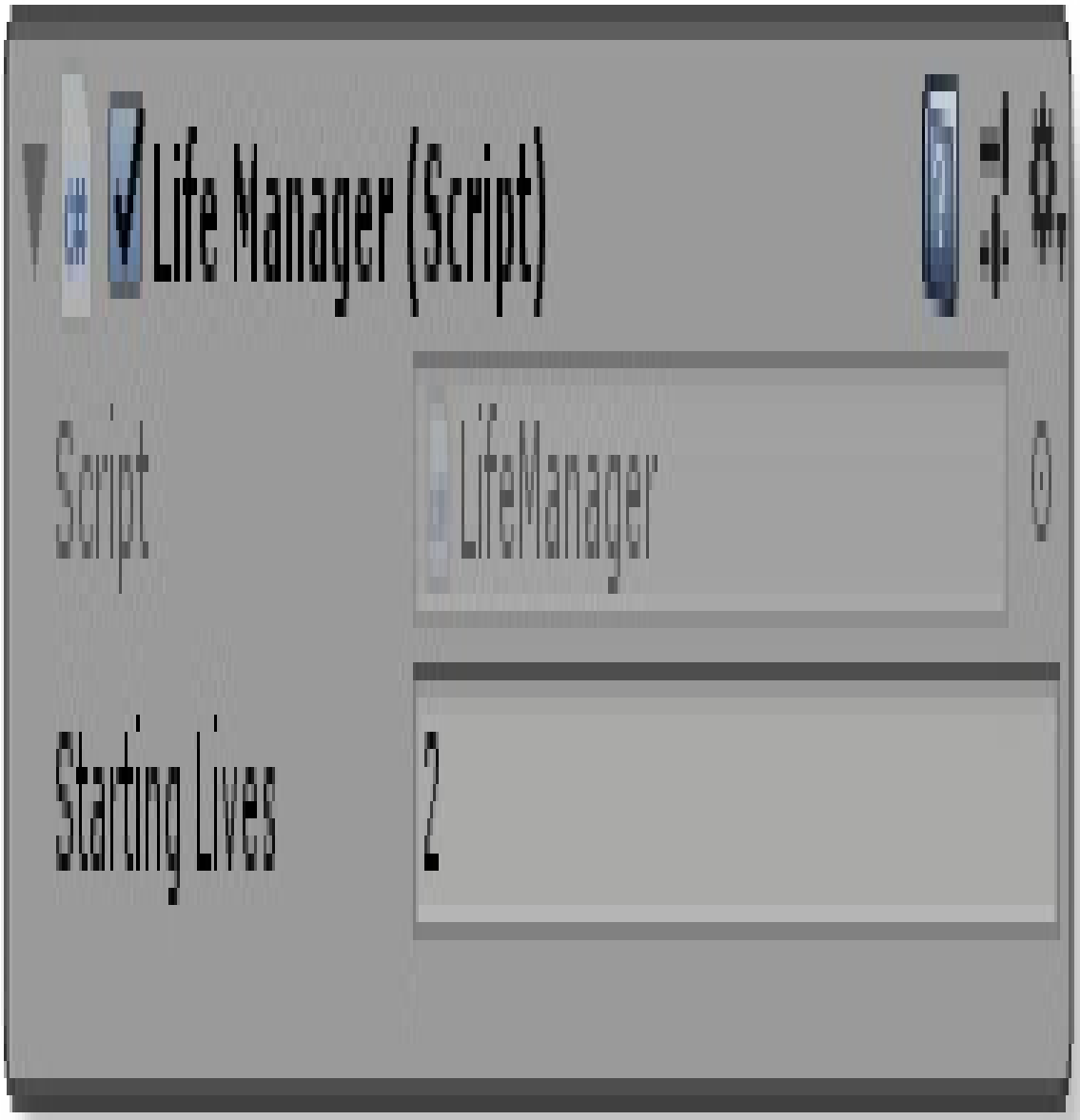
Haga un nuevo C#Script, nómbrelo “LifeManager” y asígnelo al elemento “Text”.

Abra el script creado y escriba el siguiente código:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class LifeManager : MonoBehaviour {

    public int startingLives;
    private int lifeCounter;
    private Text lifeText;

    // Use this for initialization
    void Start () {

        lifeText = GetComponent <Text> ();
        lifeCounter = startingLives;
    }

    // Update is called once per frame
    void Update () {

        lifeText.text = "X " + lifeCounter;
    }
}
```

Script LifeManager

Vaya a Unity, en el componente script asigne 2 vidas .


Life Manager (Script)
Script
LifeManager
Starting Lives
2

Asignar dos vidas

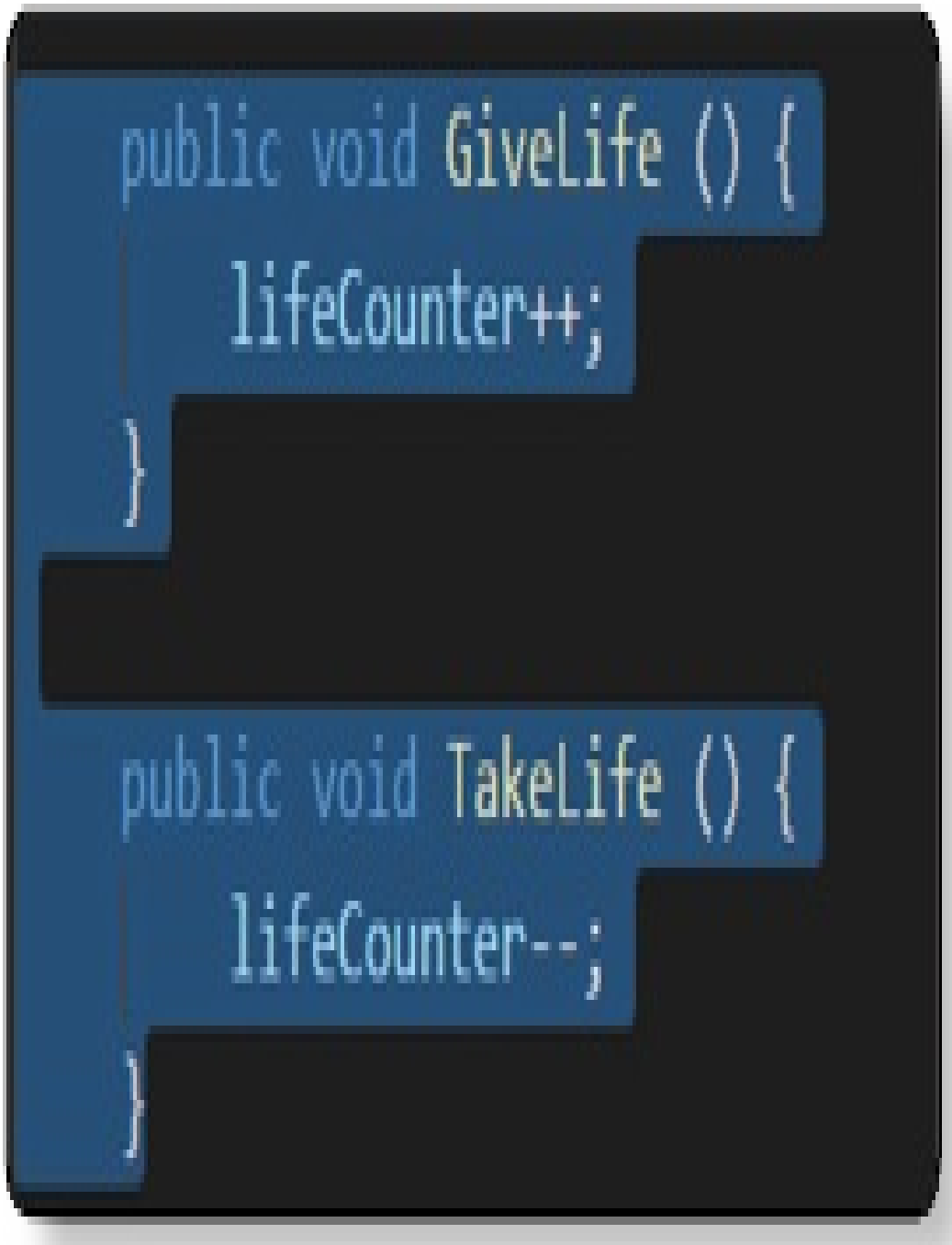
Previsualice el juego para observar la funcionalidad del sistema.

X2

Previsualización de las vidas en Game

A continuación, utilizaremos contadores dentro de funciones para añadir o restar vidas.

En el Script de LifeManager escriba las siguientes funciones.


public void GiveLife () {
lifeCounter++;
}
public void TakeLife () {
lifeCounter--;
}

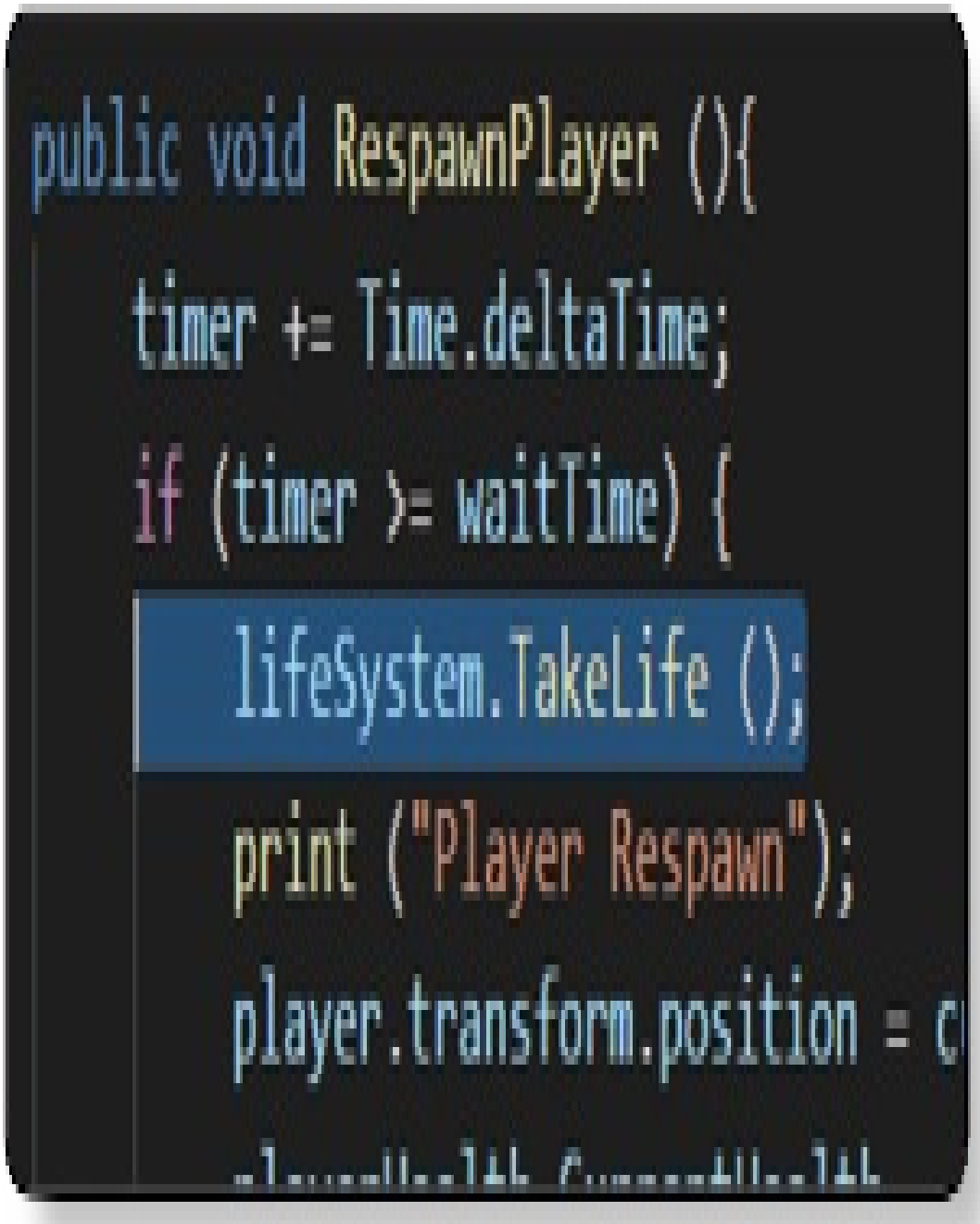
Contadores

Vaya al script “LevelManager” y haga e inicialice la variable que se muestra para mandar llamar las funciones creadas en el script “LifeManager”.

```
public Animator anim;

private LifeManager lifeSystem;

// Use this for initialization
void Start () {

    lifeSystem = FindObjectOfType<LifeManager> ();
    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealth>
```

Creación e inicialización de variables

Vaya a la función “RespawnPlayer” y mande llamar la función externa que se muestra.


public void RespawnPlayer (){
timer += Time.deltaTime;
if (timer >= waitTime) {
lifeSystem.TakeLife ();
print ("Player Respawn");
player.transform.position = c

Llamar una función externa

Previsualice el juego para observar que después de que muere el “Player” se le resta una vida en el indicador.

X1

Previsualización de la resta de vidas

Antes de finalizar el apartado debemos de crear una variable booleana para indicarnos cuando el “Player” se encuentra totalmente muerto, es decir sin vidas.

Vaya al script “PlayerHealth”, haga la siguiente variable e inicialícela.

public LevelManager levelMan
public bool isDead;
public int CurrentHealth {

Creación de variable booleana

```
void Start () {

    anim = GetComponent<Animator> (
    currentHealth = startingHealth;
    characterMovement = GetComponen
    audio = GetComponent <AudioSour
    levelManager = FindObjectOfType
    isDead = false;
```

Inicialización de variable booleana

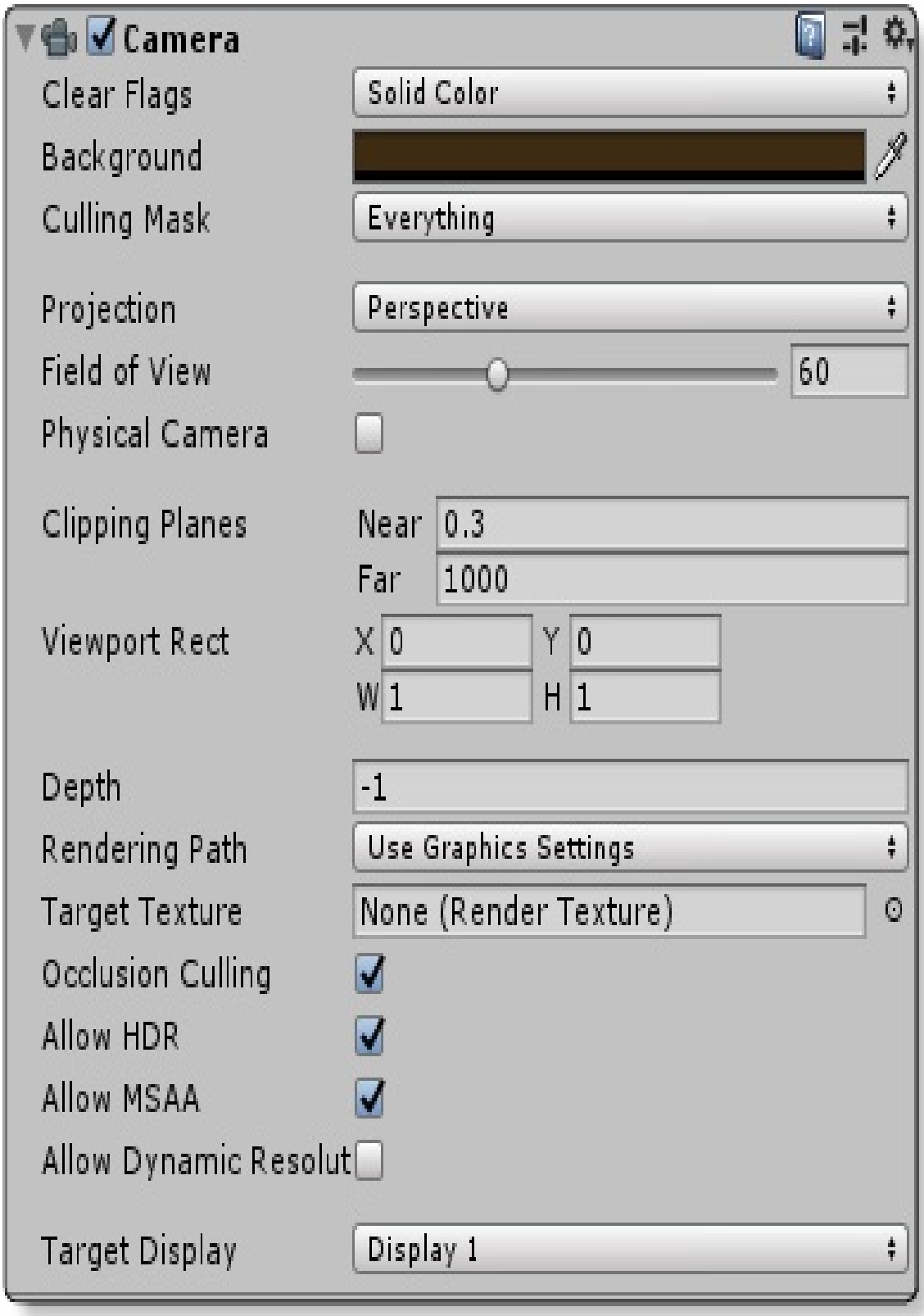
Creación de una pantalla de inicio

En este apartado vamos a crear una nueva pantalla de inicio del juego.

Así mismo crearemos y programaremos los botones necesarios para iniciar el juego o para apagarlo y crearemos un sistema para cargar escenas.

Haga una nueva escena (File/ New Scene) y nómbrela “StartScreen”.

Seleccione “Main Camera” y en “Inspector” asigne los valores que se muestran y cambie su “Background Color” al gusto.


Camera
Clear Flags
Solid Color
Background
Culling Mask
Everything
Projection
Perspective
Field of View
60
Physical Camera
Clipping Planes
Near
0.3
Far
1000
Viewport Rect
X
0
Y
0
W
1
H
1
Depth
-1
Rendering Path
Use Graphics Settings
Target Texture
None (Render Texture)
Occlusion Culling
Allow HDR
Allow MSAA
Allow Dynamic Resolut
Target Display
Display 1

Cambio de color del “Background”

Previsualización del cambio de color de fondo

Añada los siguientes elementos:

Prefab de la plataforma

Player con su animación inicial en “Idle”

Previsualización de elementos para la pantalla de inicio

Importe de la carpeta de “resources” el gráfico “ghoulieman” a la carpeta de “Sprites” y conviértalo en un sprite2D.

PSD
Ps
ghoulieLife
ghoulieMan
ghoulieMan

Archivo externo para el título

ghoulieLife
ghoulieMan

Creación de la sprite 2D del título

En Hierarchy haga un nuevo elemento “UI/ Image”.

StartScreen*
Main Camera
Directional Light
MobilePlatform
Player
Canvas
Image

Creación de una imagen UI

Con “Image” seleccionada vaya a Inspector, asigne en “SourceImage” el nuevo Sprite y colóquelo en la escena.

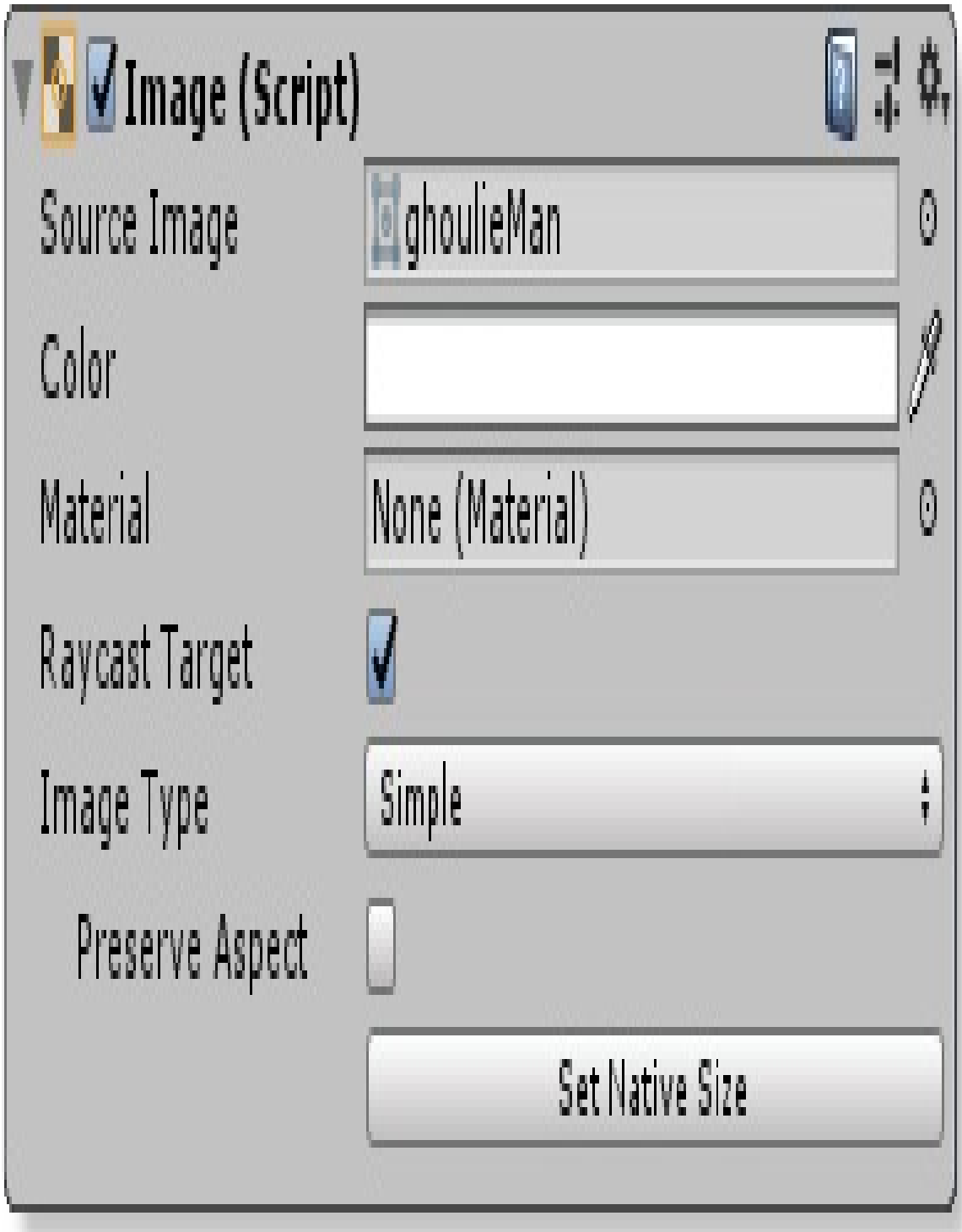
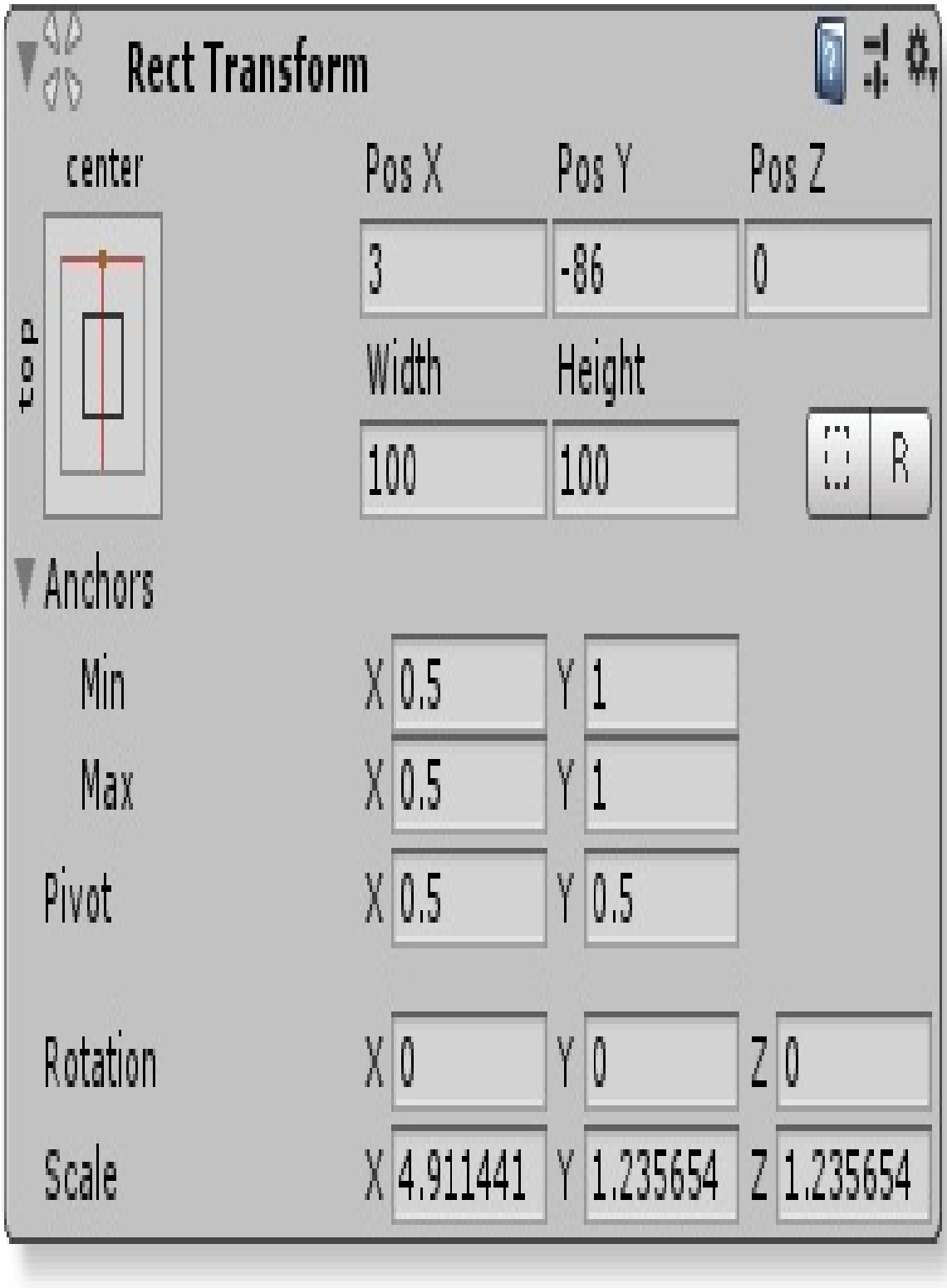


Image (Script)
Source Image
ghoulieMan
Color
Material
None (Material)
Raycast Target
Image Type
Simple
Preserve Aspect
Set Native Size

Valores de la UI Image


Rect Transform
center
top
Pos X
Pos Y
Pos Z
3
-86
0
Width
Height
100
100
R
Anchors
Min
X 0.5
Y 1
Max
X 0.5
Y 1
Pivot
X 0.5
Y 0.5
Rotation
X 0
Y 0
Z 0
Scale
X 4.911441
Y 1.235654
Z 1.235654

Anclaje de la imagen

GHOULIEMAN

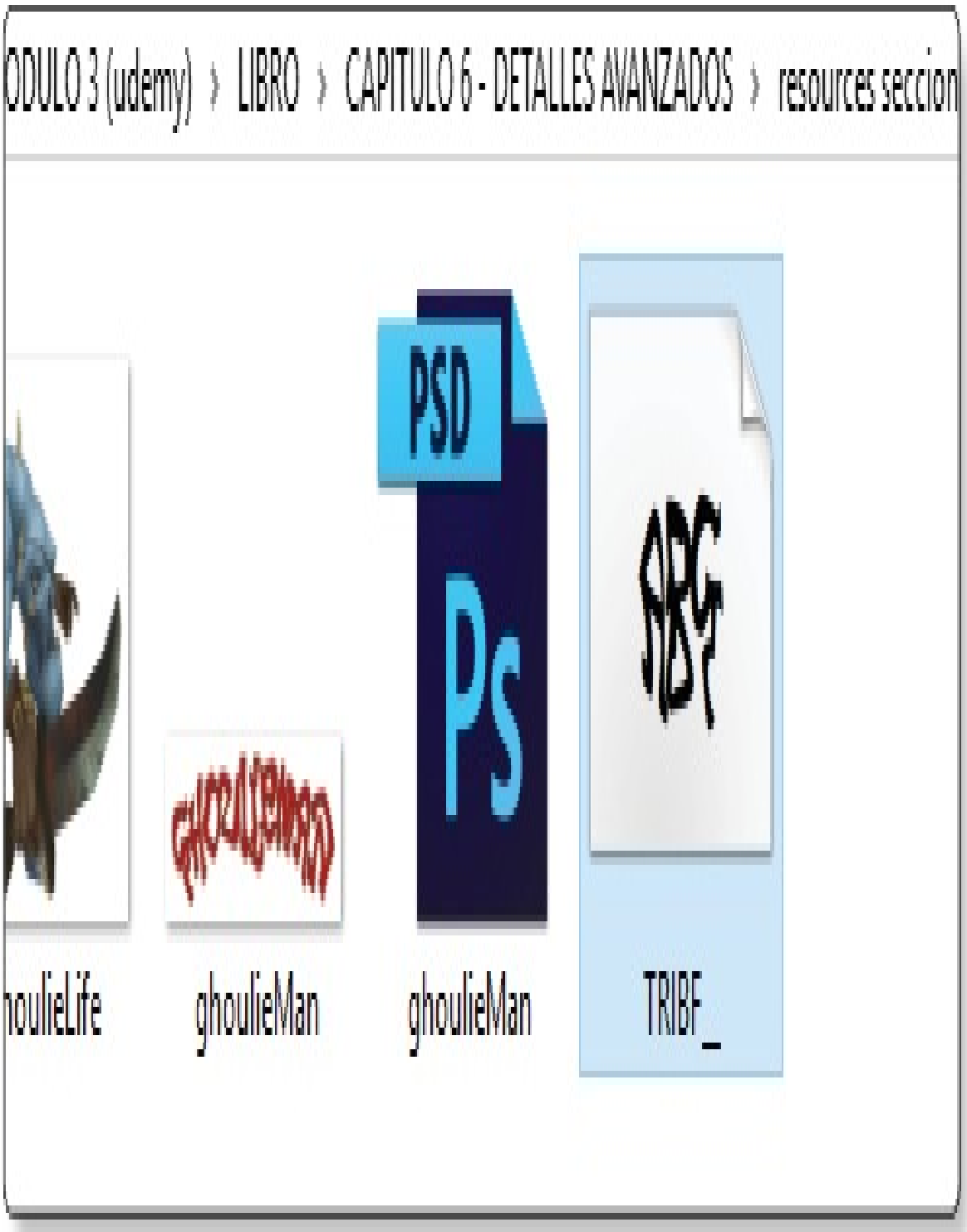
Colocación de la imagen en el editor

GHOULEMAS

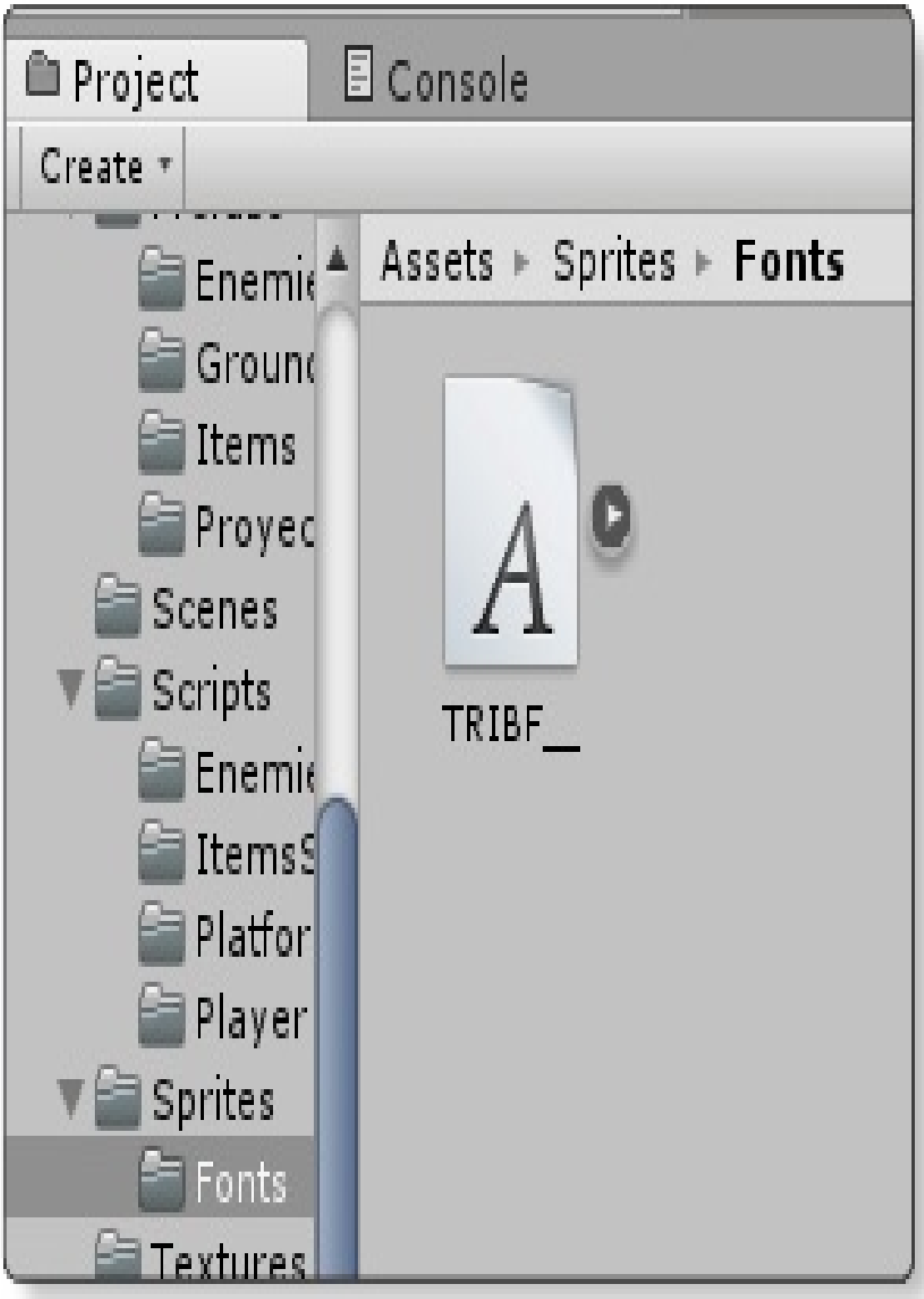
Previsualización de la pantalla de inicio

En el folder de “Sprites” haga un nuevo folder y nómbrelo “Fonts”.

Dentro del folder “Fonts” importe la fuente “TRIBF__” de la carpeta de “resorces”.


ODULO 3 (udemy) › LIBRO › CAPITULO 6 - DETALLES AVANZADOS › resources seccion
PSD
Ps
noulieLife
ghoulieMan
ghoulieMan
TRIBE_

Archivo externo de la fuente


Project
Console
Create
Assets ▸ Sprites ▸ Fonts
Enemie
Ground
Items
Proyec
Scenes
Scripts
Enemie
ItemsS
Platfor
Player
Sprites
Fonts
Textures
A
TRIBF_

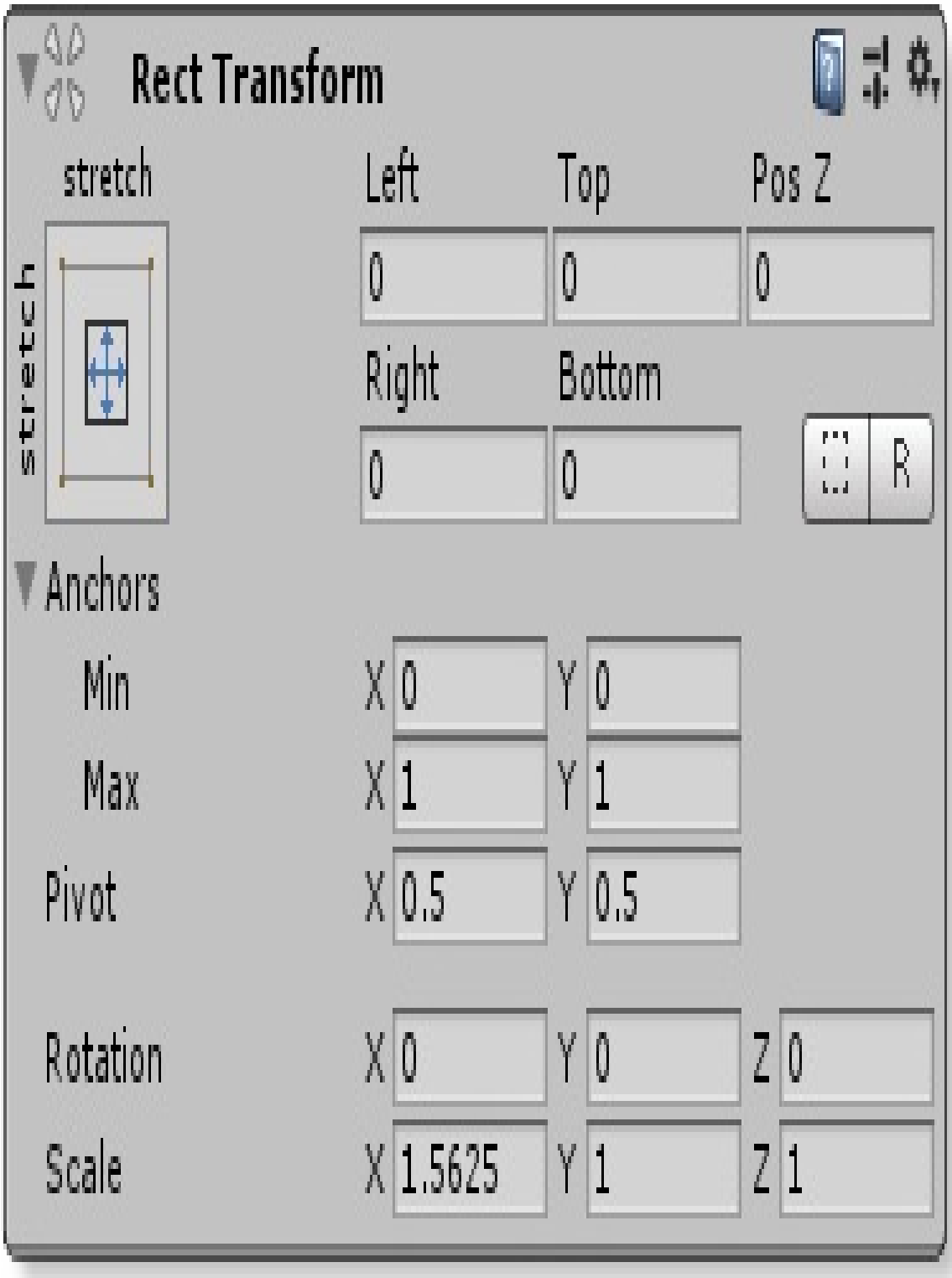
Importación de la fuente

Haga un “UI/ Button” y renómbrelo como “StartButton”.

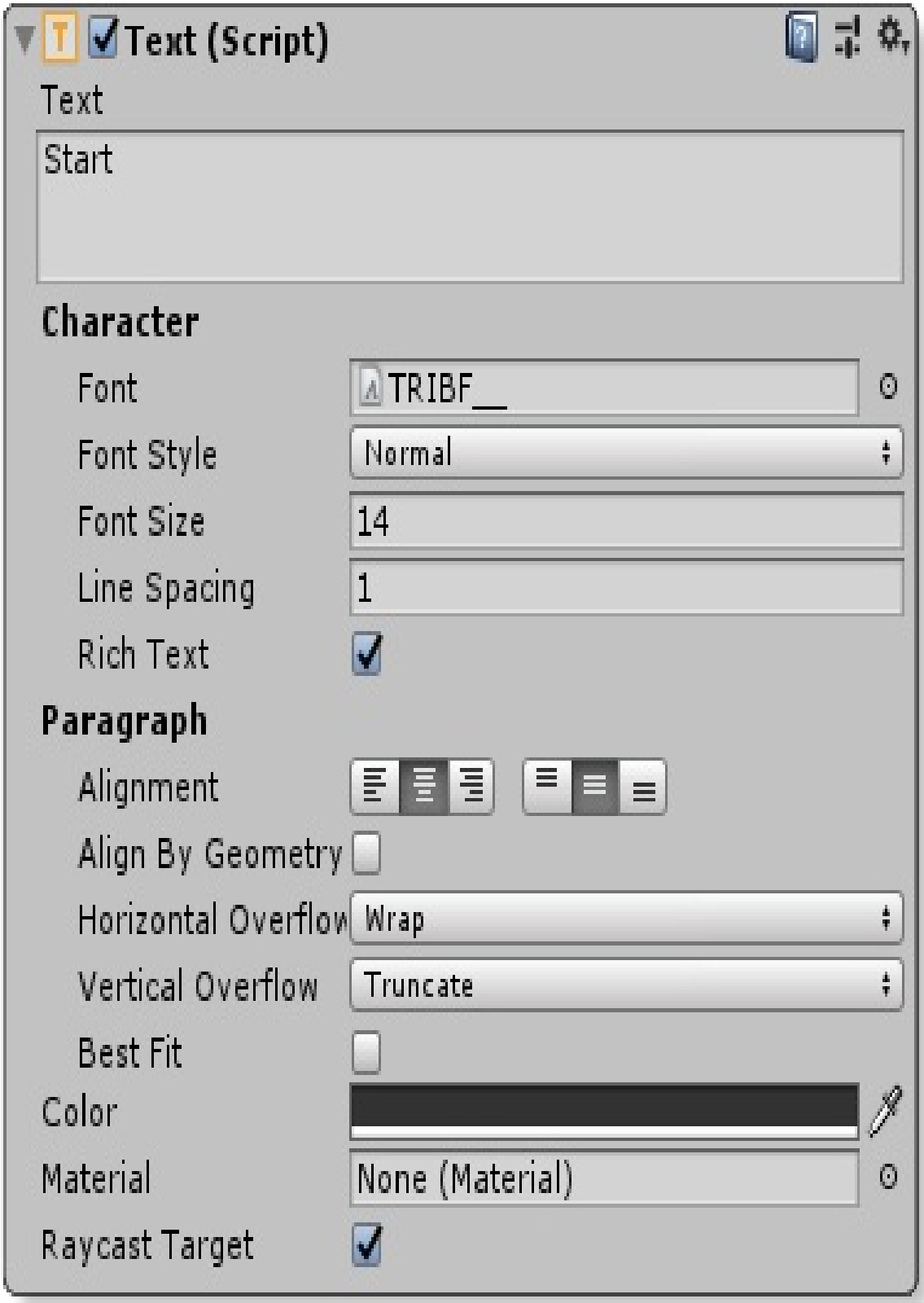
Seleccione su subelemento “Text” y en Inspector asigne los valores que se muestran.

StartButton
Text

Selección del elemento Text


Rect Transform
stretch
stretch
Left
Top
Pos Z
0
0
0
Right
Bottom
0
0
R
Anchors
Min
X 0
Y 0
Max
X 1
Y 1
Pivot
X 0.5
Y 0.5
Rotation
X 0
Y 0
Z 0
Scale
X 1.5625
Y 1
Z 1

Anclaje del elemento “Text”


Text (Script)
Text
Start
Character
Font
TRIBF__
Font Style
Normal
Font Size
14
Line Spacing
1
Rich Text
Paragraph
Alignment
Align By Geometry
Horizontal Overflow
Wrap
Vertical Overflow
Truncate
Best Fit
Color
Material
None (Material)
Raycast Target

Valores del texto

Seleccione el objeto “StartButton” y asigne los valores que se muestran.

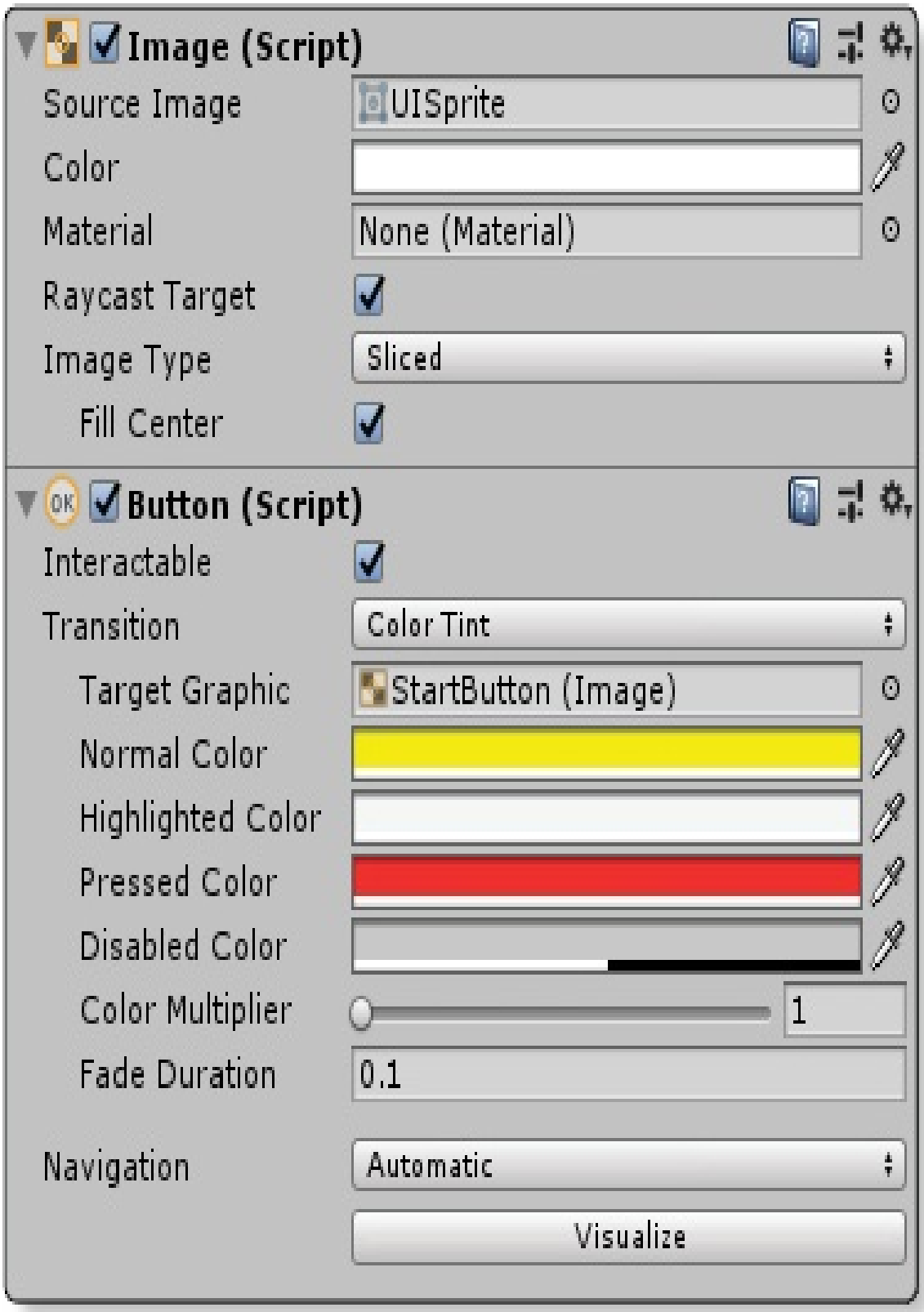


Image (Script)
Source Image
UISprite
Color
Material
None (Material)
Raycast Target
Image Type
Sliced
Fill Center
Button (Script)
Interactable
Transition
Color Tint
Target Graphic
StartButton (Image)
Normal Color
Highlighted Color
Pressed Color
Disabled Color
Color Multiplier
1
Fade Duration
0.1
Navigation
Automatic
Visualize

Valores del botón “Start”

GHOULIENAD
START

Colocación del botón “Start” en el editor

GHOULIEMAN
START

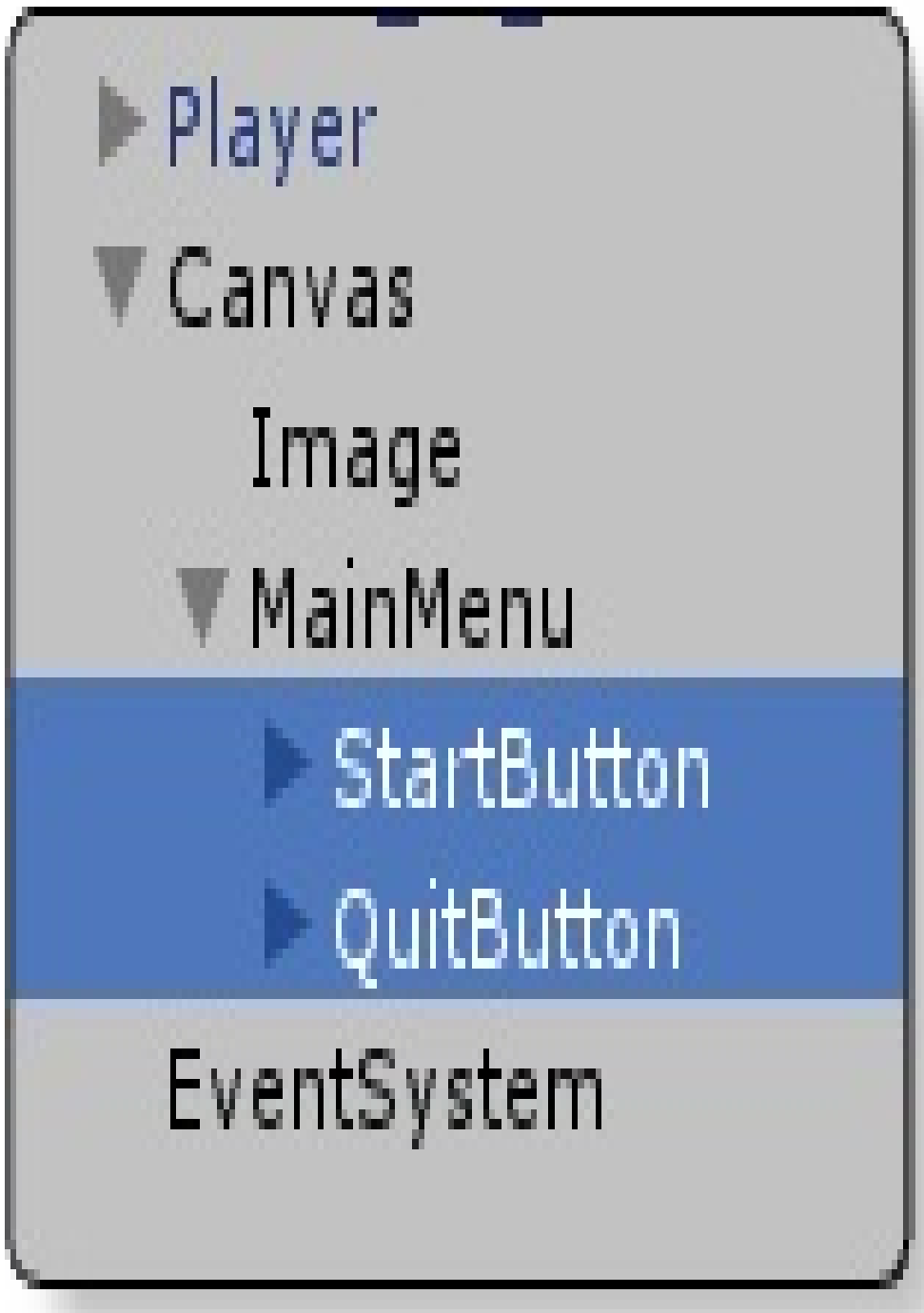
Previsualización del botón

Haga un duplicado de “StartButton”, renómbrelo “QuitButton” y colóquelo como se muestra.

GHOULEMAD
START
QUIT

Creación del botón “Quit”

Haga un nuevo objeto “EmptyObject”, renómbrelo “MainMenu” y haga hijos los objetos “StartButton” y “QuitButton”.


Player
Canvas
Image
MainMenu
StartButton
QuitButton
EventSystem

Emparentamiento de objetos

A continuación, se añadirán a la ventana de “BuildSettings” las escenas que queremos tener activas al momento de construir nuestro juego.

En el menú general asigne “File / Buid Settings” y añada las escenas de inicio y la del juego activo.

Scenes In Build
Scenes/StartScreen
0
Scenes/GhoulieMan421
1
Add Open Scenes

Añadir escenas activas en BuildSettings

Haga un nuevo C#Script, nómbrelo “MainMenuScript” y asígnelo al objeto “MainMenu”.

Abra el script “MainMenuScript” y escriba el siguiente código:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenuScript : MonoBehaviour {


    public void PlayGame () {

        SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex + 1);
    }
}
```

Función PlayGame

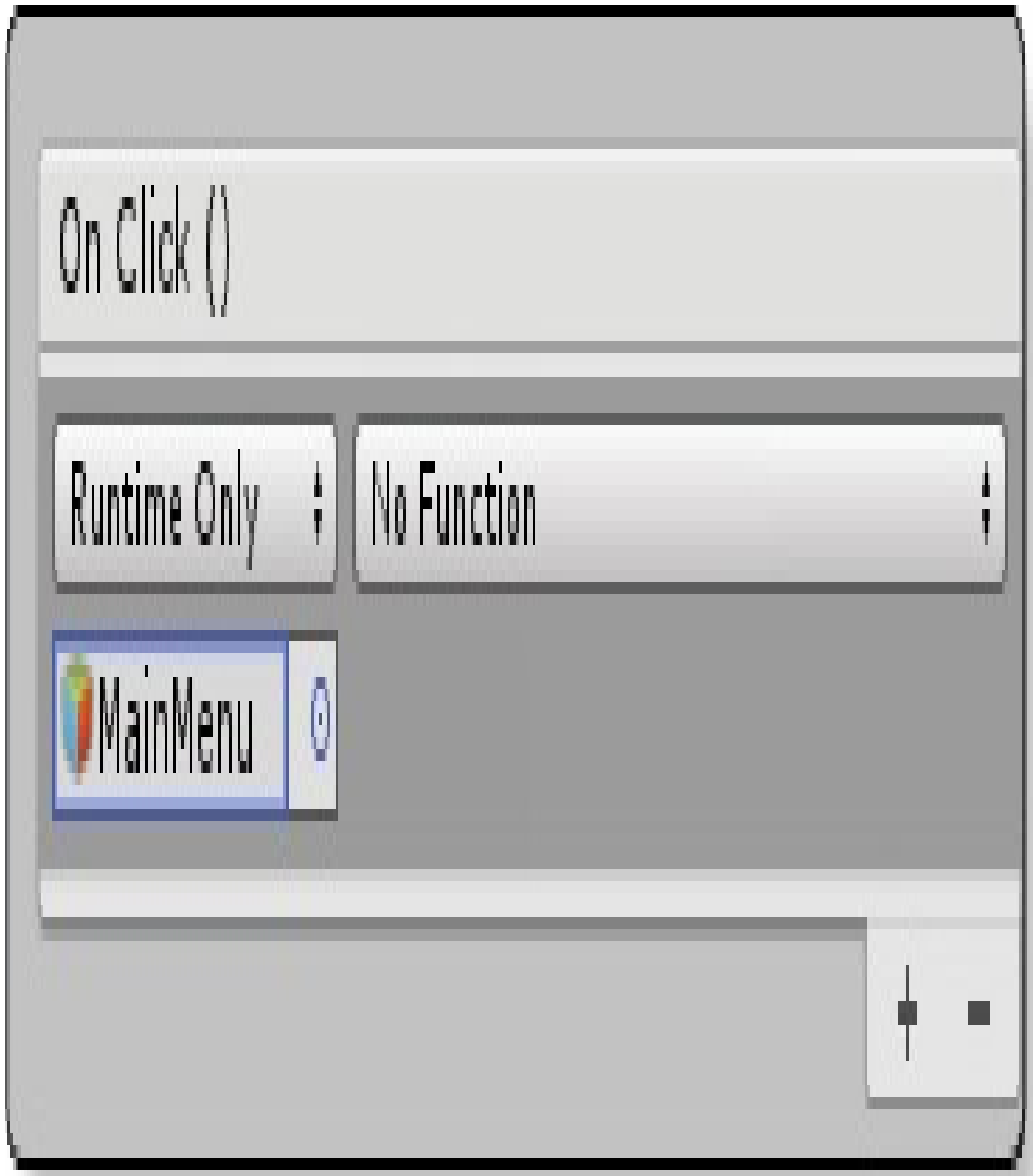
Inicialmente debemos mandar llamar la librería interna “SceneManagment” que nos permitirá la navegación entre escenas.

En seguida se creará la función “PlayGame” que cargará la siguiente escena que se encuentre activa, tomando en cuenta los siguientes factores:

La cuenta de las escenas activas comienza en cero.

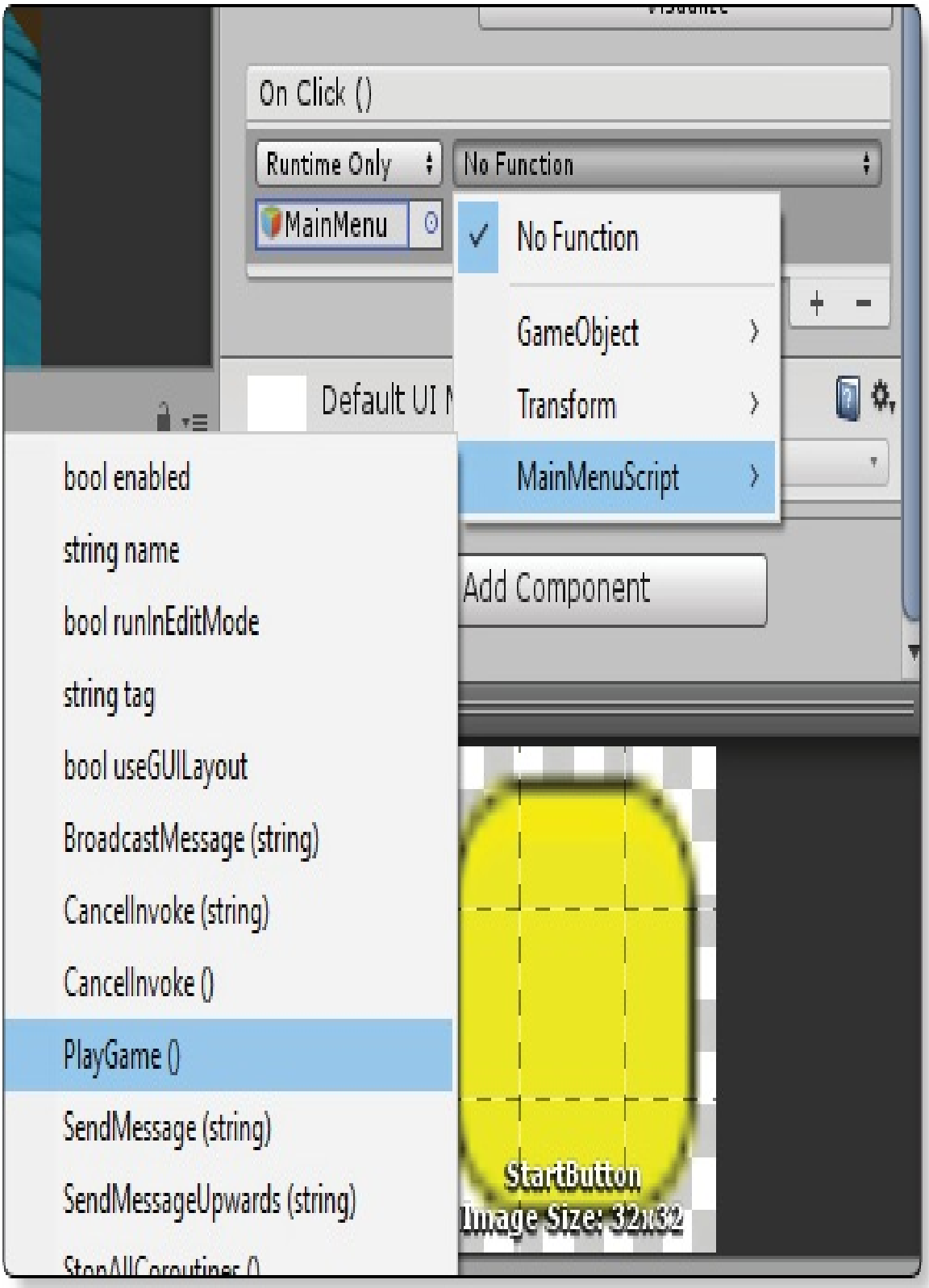
Al añadir el contador (+1) le sumará una unidad a la escena activa para cargar la siguiente, en este caso 0 + 1 = 1 y cargará la escena 01.

Vaya a Unity, seleccione el objeto “StartButton” y en su función “Onclick ()” asigne el objeto “MainMenu”, arrastrándolo de “Hierarchy” a “Inspector”.


On Click ()
Runtime Only
No Function
MainMenu

Asignación de objeto “MainMenu”

Clic en “No Function” y elija la función creada.


On Click ()
Runtime Only
No Function
MainMenu
No Function
GameObject
Transform
MainMenuScript
Add Component
bool enabled
string name
bool runInEditMode
string tag
bool useGUILayout
BroadcastMessage (string)
CancelInvoke (string)
CancelInvoke ()
PlayGame ()
SendMessage (string)
SendMessageUpwards (string)
StartButton
Image Size: 32x32

Asignación de función “PlayGame”

On Click ()
Runtime Only
MainMenu.PlayGame
MainMenu (Mai

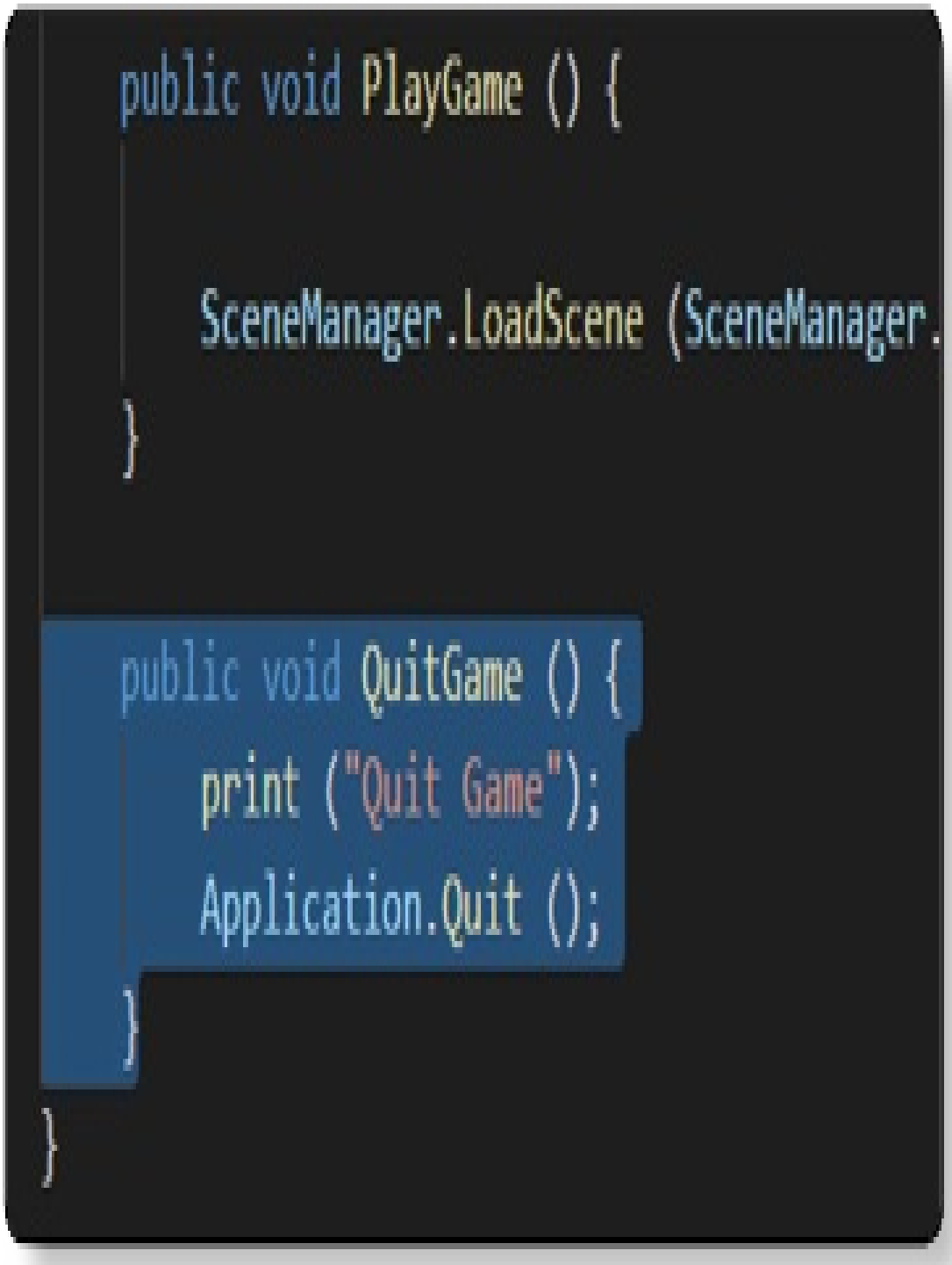
Función “PlayGame” asignada

Como muestra el ejemplo, podemos crear funciones específicas para asignarlas a eventos del ratón, en este caso se la asignamos a la función “On Click”.

Previsualice el juego en su pantalla de inicio para observar el evento “OnClick” del botón “Start”, inmediatamente nos lleva a la escena activa del juego.

A continuación, programaremos el botón de “Quit Game”.

Vaya al script “MainMenuScript” y escriba la siguiente función.


public void PlayGame () {
SceneManager.LoadScene (SceneManager.
}
public void QuitGame () {
print ("Quit Game");
Application.Quit ();
}
}

Función “QuitGame”

Con el comando “Application.Quit” mandamos llamar la función interna para cerrar la aplicación.

Seleccione el botón “QuitGame” y asigne la función creada.

On Click ()
Runtime Only
MainMenu.QuitGame
MainMenu (Mai

Asignación de función “QuitGame”

NOTA:

Al previsualizar el juego en Unity y pulsar el botón “Quit” no pasará nada, el comando “Application.Quit” únicamente funciona cuando corremos el juego externamente, es decir, una vez que hayamos generado su instalador.

Creación de una pantalla de GameOver

En este apartado vamos a crear y a programar una ventana de “GameOver” con una técnica diferente a la pantalla de inicio y su programación para que aparezca una vez que las vidas del “Player” lleguen a cero.

También vamos a mandar llamar la pantalla de inicio después de un lapso de tiempo en donde aparezca la pantalla “GameOver”.

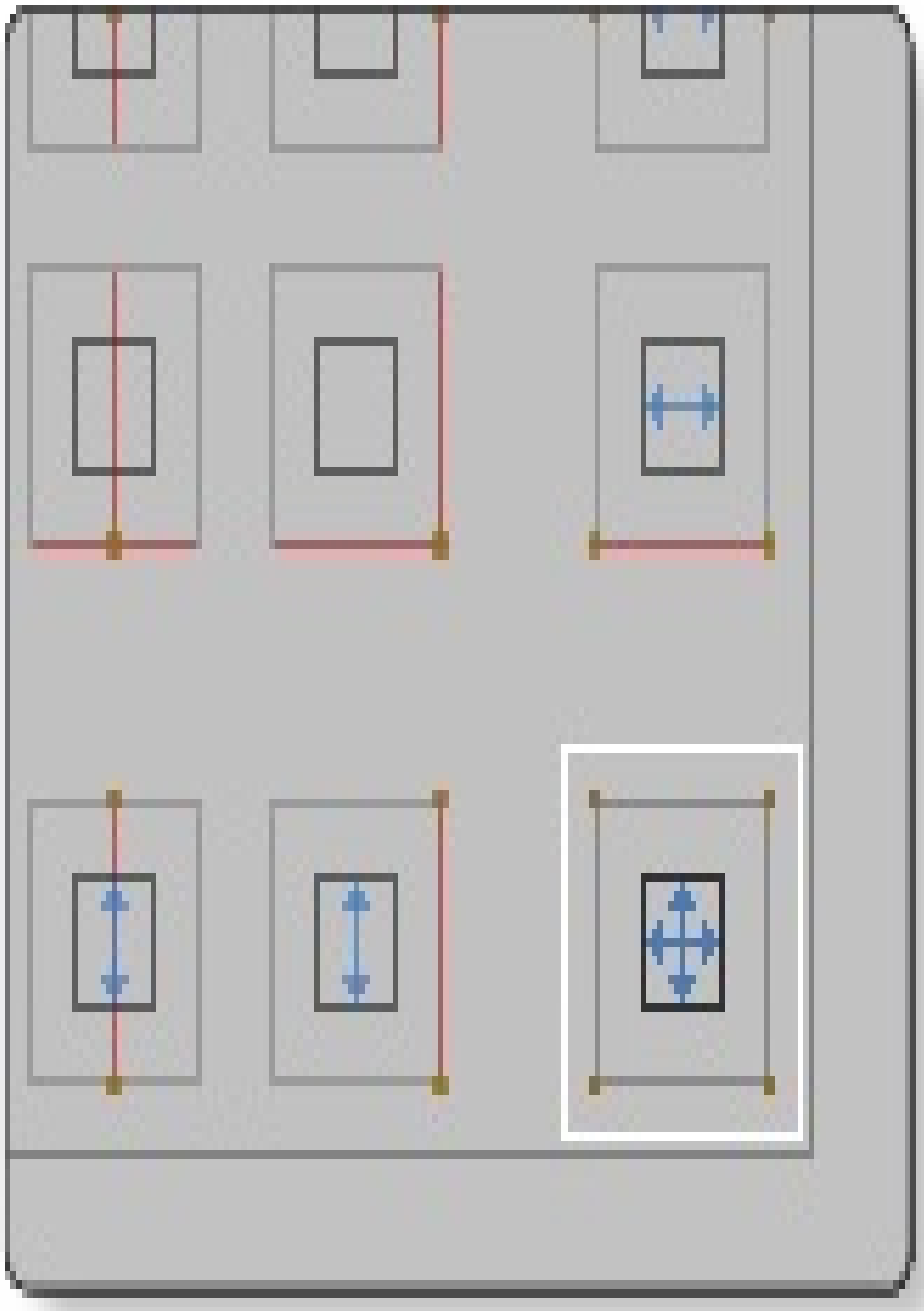
Al previsualizar el juego podemos observar que al perder vidas el contador comienza a generar números negativos, esto lo solucionaremos programando su pantalla de “Game Over”.

En “Hierarchy” seleccione el objeto “GUI”, asigne clic derecho y haga un nuevo “EmptyObject” y renómbrelo “GameOverScreen”.

GUI
HealthBar
LifeSystem
GameOverScreen
EventSystem

Creación de objeto “GameOverScreen”

Ancle el objeto en el “AnchorPresets” que se muestra.



Anclaje del nuevo objeto

Asigne un componente “UI / Image”, asigne un color rojo y expándalo.

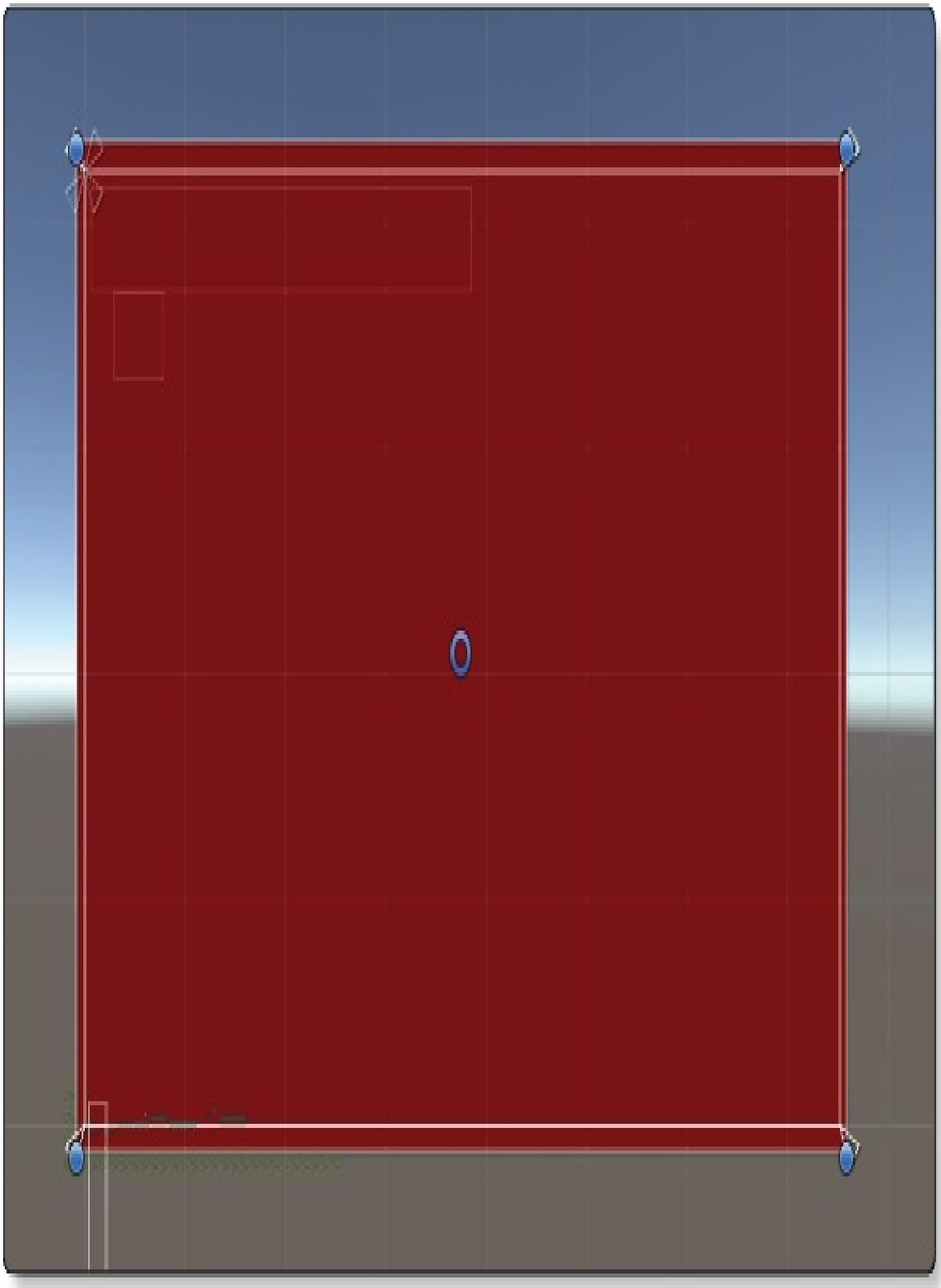


Imagen Background

Haga un nuevo elemento “UI / Image”, importe el gráfico correspondiente de la carpeta “resources”, conviértalo en “Sprite 2D” y colóquelo como se muestra.

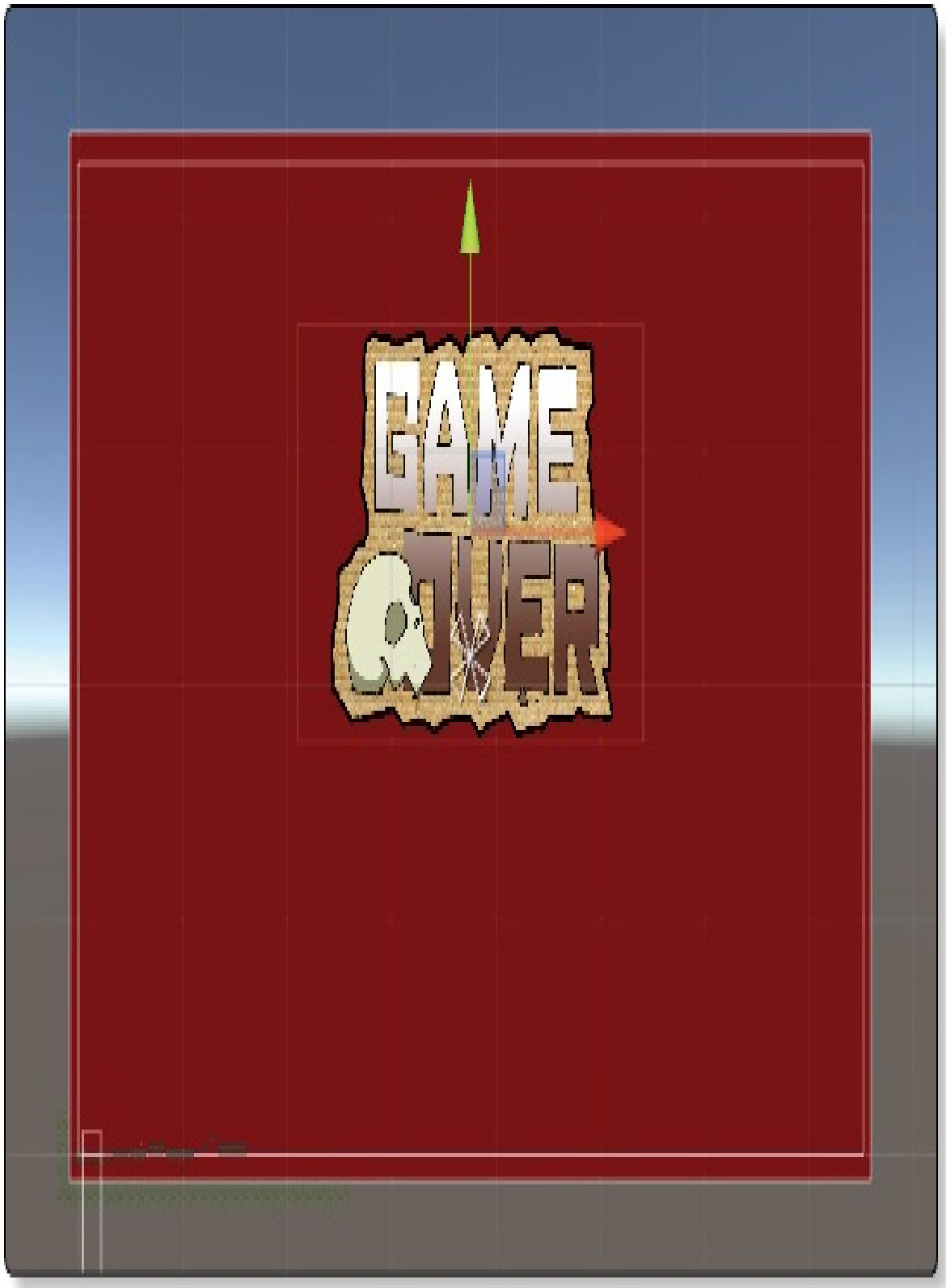

GAME
OVER

Gráfico GameOver

GUI
HealthBar
LifeSystem
GameOverScreen
Image
Image (1)
EventSystem

Emparentamiento correcto de los elementos

Seleccione el objeto “GameOverScreen” y en “Inspector” deshabilítelo.

Inspector
GameOverScreen
Tag
Untagged

Deshabilitación de objeto

Vaya al script “LifeManager”, haga las siguientes variables e inicialícelas.

```
public int startingLives;
private int lifeCounter;
private Text lifeText;

private GameObject player;
public GameObject gameOverScreen;

// Use this for initialization
void Start () {

    lifeText = GetComponent <Text> ();
    lifeCounter = startingLives;
    player = GameManager.instance.Player;
}
```

Creación e inicialización de variables

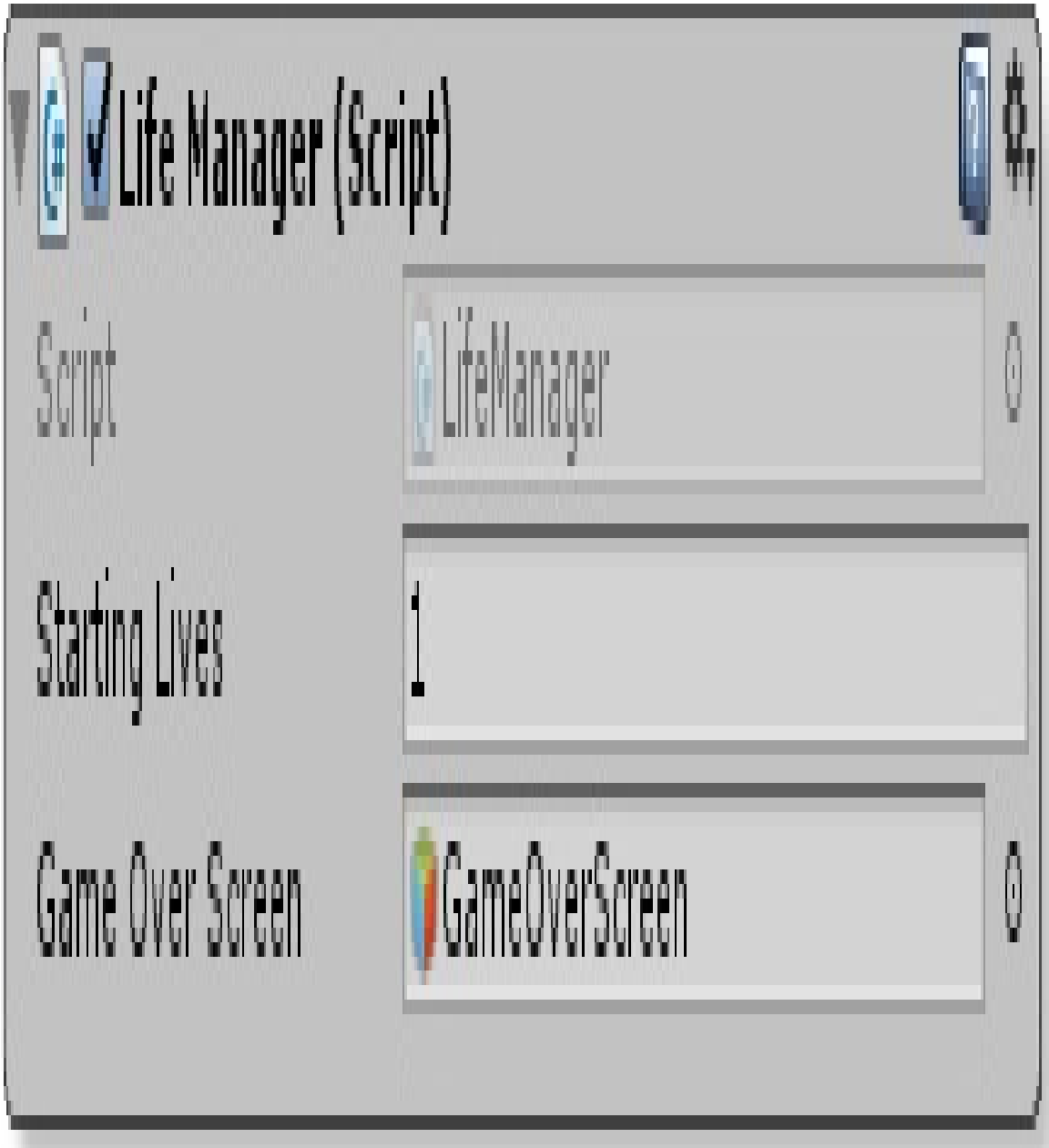
Escriba la siguiente estructura condicional en la función “Update”.

```
void Update () {

    lifeText.text = "X " + lifeCounter;

    if (lifeCounter <= 0) {

        gameOverScreen.SetActive (true);
        player.gameObject.SetActive (false);
    }
}
```

Estructura condicional

En esta estructura condicional estamos activando la pantalla de “GameOver” y deshabilitando a nuestro “Player” cuando su contador de vidas sea igual o menor que cero.

Vaya a Unity y asigne el objeto “GameOverScreen” en el script.


Life Manager (Script)
Script
LifeManager
Starting Lives
1
Game Over Screen
GameOverScreen

Asignación de objeto en el script

Previsualice el juego para observar que cuando las vidas del “Player” lleguen a cero, aparece la pantalla “GameOver”.

GAME
OVER

Previsualización de pantalla GameOver

PROBLEMA: La pantalla de GameOver se mantiene y no regresa al “Main Menu”

SOLUCIÓN: Vaya al script “LifeManager”, haga la siguiente variable y llame la librería.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LifeManager : MonoBehaviour {

    public int startingLives;
    private int lifeCounter;
    private Text lifeText;

    private GameObject player;
    public GameObject gameOverScreen;
    public float waitAterGameOver;
```

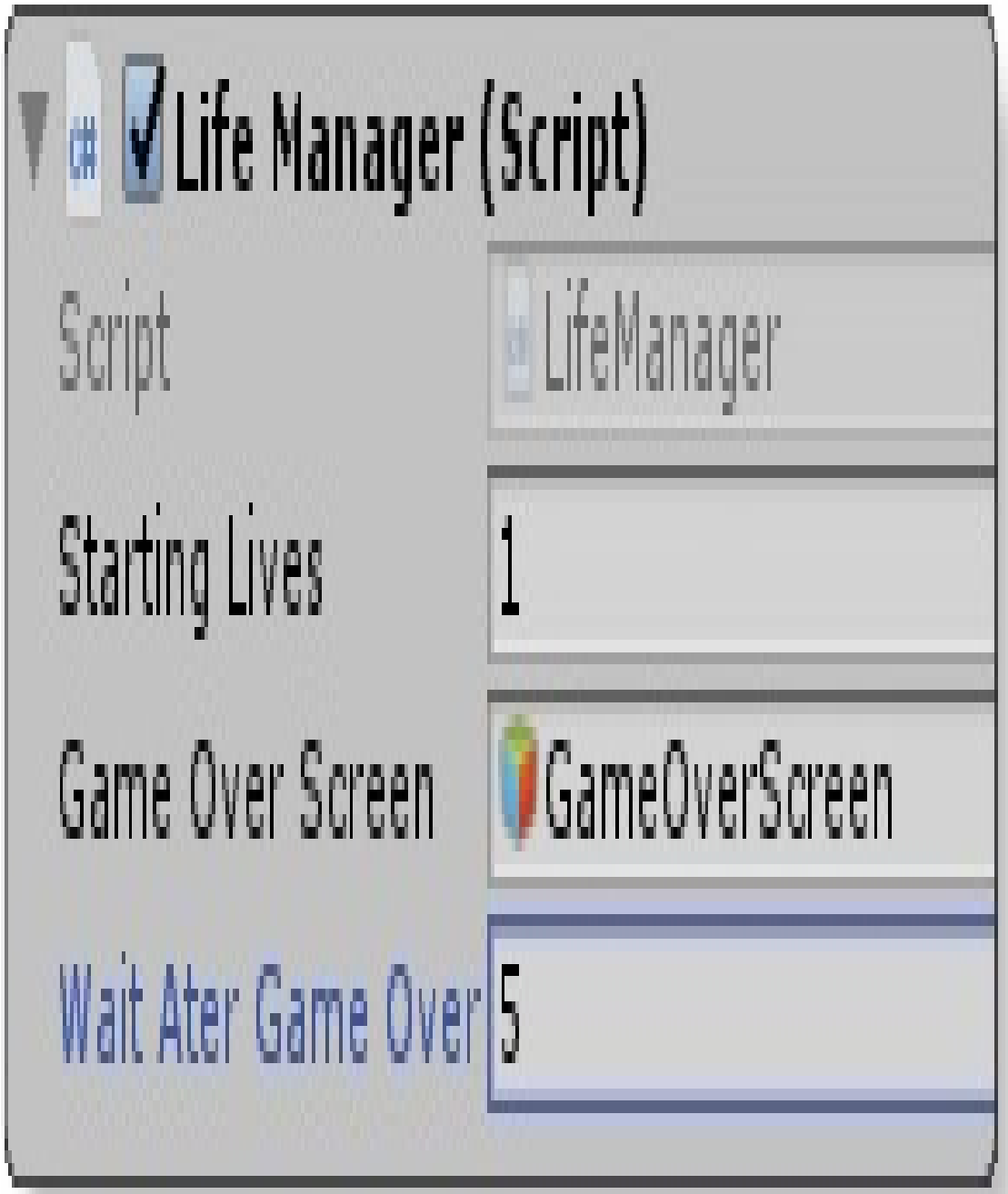
Importar biblioteca y creación de variable

Escriba la siguiente estructura condicional en la función “Update”.

```
        if (lifeCounter <= 0) {

            gameOverScreen.SetActive (true);
            player.gameObject.SetActive (false);
        }

        if (gameOverScreen.activeSelf) {
            waitAterGameOver -= Time.deltaTime;
        }

        if (waitAterGameOver < 0) {
            SceneManager.LoadScene (0);
        }
    }
```

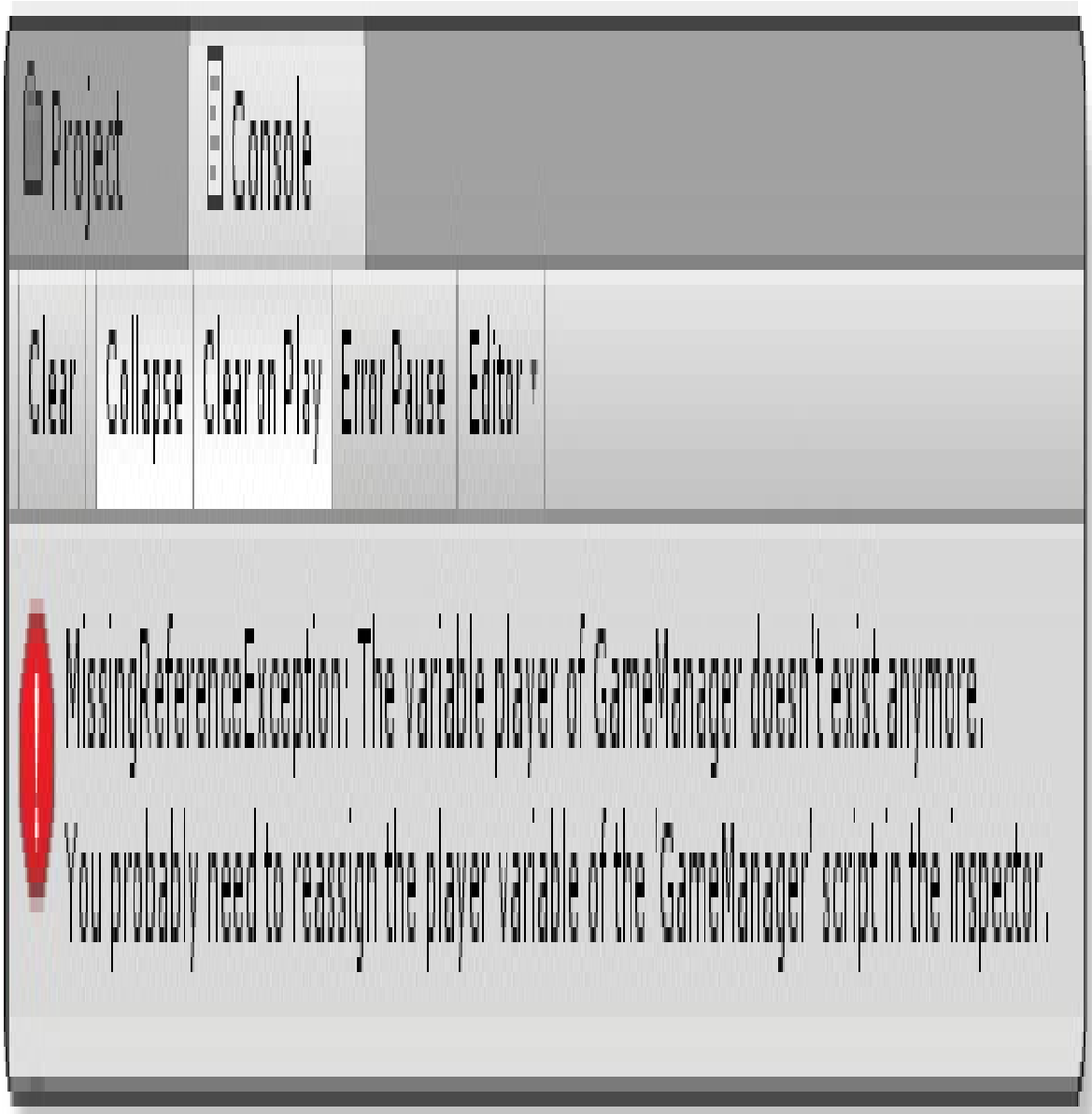
Estructuras condicionales

Vaya a Unity y en el componente “Script” asigne 5 a la variable “Wait After Game Over” para esperar 5 segundos antes de que se cargue la escena de inicio.


Life Manager (Script)
Script
LifeManager
Starting Lives
1
Game Over Screen
GameOverScreen
Wait Ater Game Over
5

Asignación de tiempo de espera

PROBLEMA: Después de empezar nuevamente el juego, en el componente script de “GameManager” no se asigna el “Player”.


Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
MissingReferenceException: The variable player of GameManager doesn't exist anymore.
You probably need to reassign the player variable of the GameManager script in the inspector.

Problema en consola

SOLUCIÓN: Vaya al script “GameManager” y en el Singleton borre o comente “DontDestroyOnLoad”.

```
void Awake () {

    if (instance == null) {
        instance = this;
    } else if (instance != this) {
        Destroy (gameObject);
    }
    //DontDestroyOnLoad (gameObject);
}
```

Comentar código

Creación y Programación de un Item de vida

En este apartado crearemos y programaremos un Item para añadir una vida al Player para que posteriormente lo asignemos al “Array” de los Items que dejan los enemigos al ser destruidos.

Seleccione el sprite utilizado para el indicador de las vidas del Player y colóquelo en la escena.

Escale el sprite y reposicione su pivote en SpriteEditor para colocarlo en la base.

KnifeSpawn

Sprite en escena

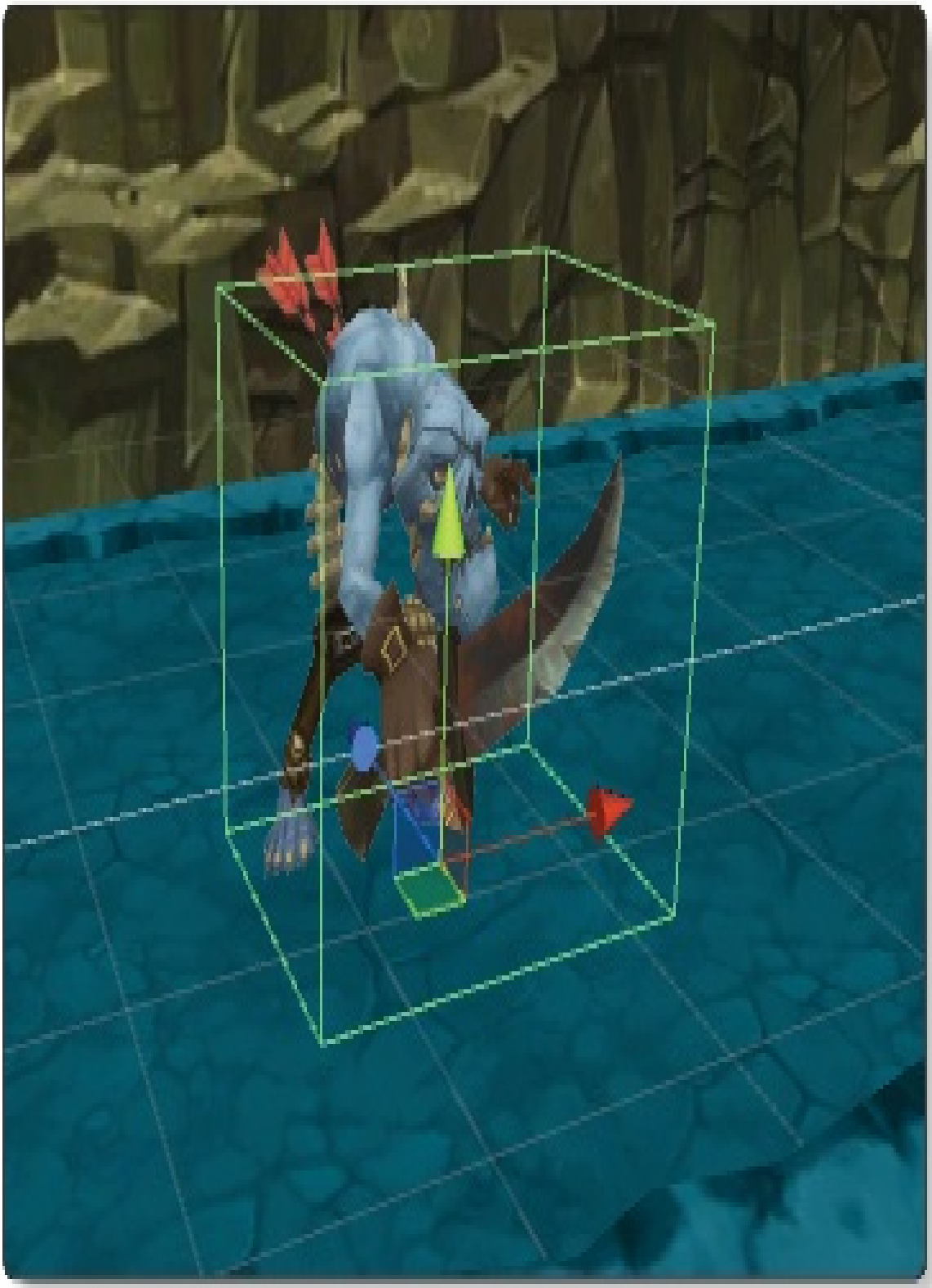
Haga un nuevo “EmptyObject”, renómbrelo “LifeItem” y colóquelo en la base del sprite.

Seleccione el sprite y hágalo hijo del objeto LifeItem

LifeItem
ghoulieLife

Emparentamiento de LifeItem

Seleccione el objeto “LifeItem”, asigne un “BoxCollider”, formatee la escala y asegúrese de marcar la opción “IsTrigger”.



Box Collider

Haga un nuevo C#Script, nómbrelo “LifeItemScript” y asígnelo al objeto “LifeItem”.

Abra el script y escriba el siguiente código:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LifeItem : MonoBehaviour {

    private GameObject player;
    private AudioSource audio;
    private LifeManager lifeManager;
    private SpriteRenderer spriteRenderer;
    private BoxCollider boxCollider;

    // Use this for initialization
    void Start () {

        player = GameManager.instance.Player;
        lifeManager = FindObjectOfType<LifeManager> ();
        spriteRenderer = GetComponentInChildren<SpriteRenderer> ();
        boxCollider = GetComponent<BoxCollider> ();
    }

    void OnTriggerEnter (Collider other) {
        if (other.gameObject == player) {
            PickLife ();
            print ("Life Collected");
        }
    }

    public void PickLife () {

        lifeManager.GiveLife ();
        spriteRenderer.enabled = false;
        boxCollider.enabled = false;
        Destroy (gameObject);
    }
}
```

LifeItemScript

Previsualice el juego para observar que cuando el Player toma el Item se añade una vida al contador.

Convierta el Item en Prefab.

TRABAJO CON ENEMIGO SPAWNER

INTRODUCCIÓN

En este capítulo abarcaremos las siguientes funcionalidades:

Crearemos un enemigo tipo “Spawner” o generador, creando el modelo a partir de modelos existentes en nuestra carpeta de “resources”.

Crearemos animaciones internas con la ventana “Animation” para programar sus comportamientos con código.

Crearemos un nuevo enemigo tipo “Spawn” con funcionalidad de movimiento hacia la izquierda y programaremos su animación.

Programaremos la funcionalidad de generación infinita de enemigos hasta que el objeto generador sea destruido.

Programaremos la música para los niveles de nuestro videojuego.

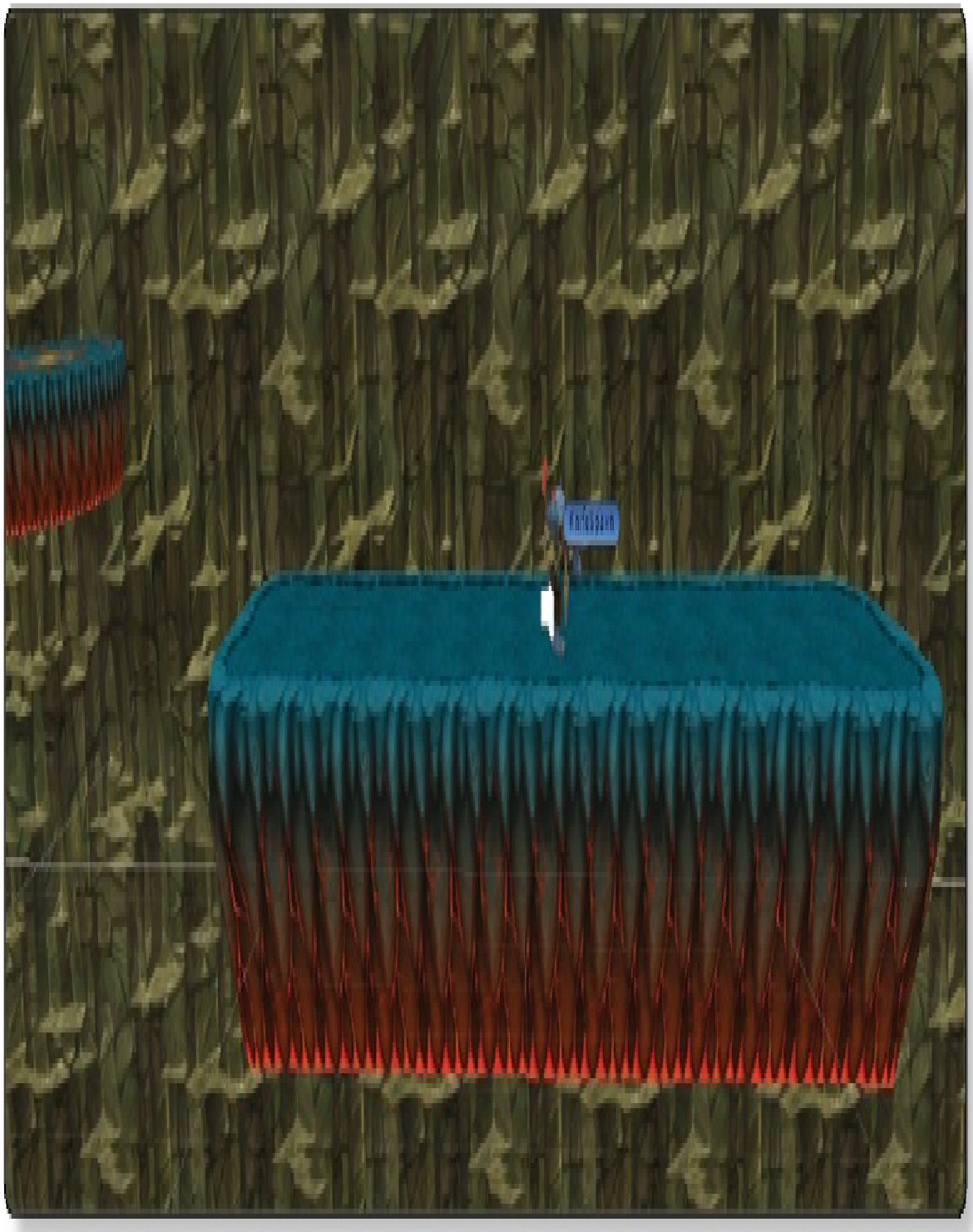
Programaremos la funcionalidad completa para pausar nuestro videojuego.

Creación de un enemigo tipo “Spawner”

En este apartado crearemos un enemigo tipo “Spawner” o bien “generador” de otros enemigos.

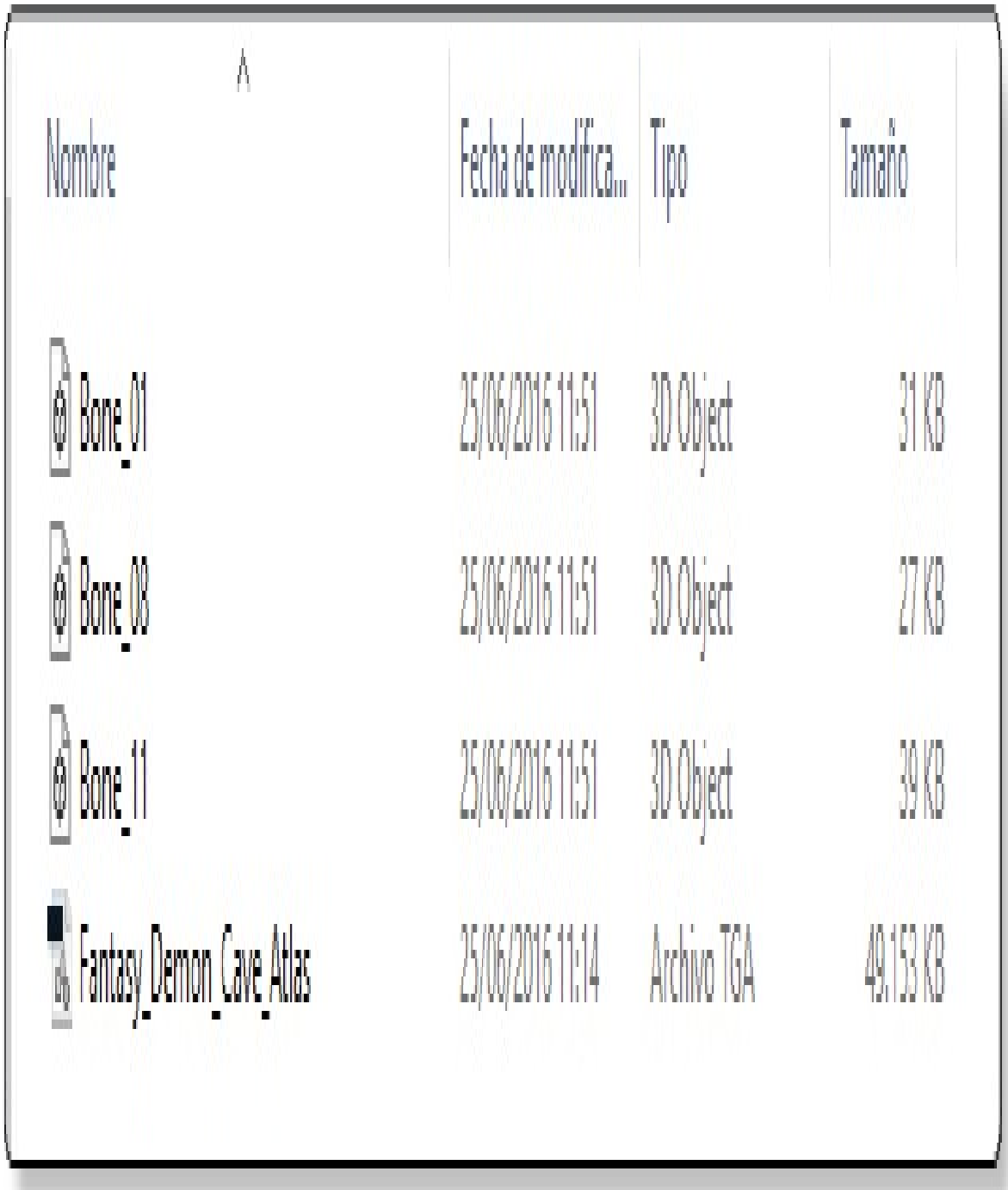
Dicho enemigo estará construido a partir de diferentes modelos que podremos encontrar en nuestra carpeta de “resources” para posteriormente añadir los componentes necesarios para su funcionalidad.

Añada una plataforma más al nivel después de las plataformas móviles y coloque al “Player” en ella.



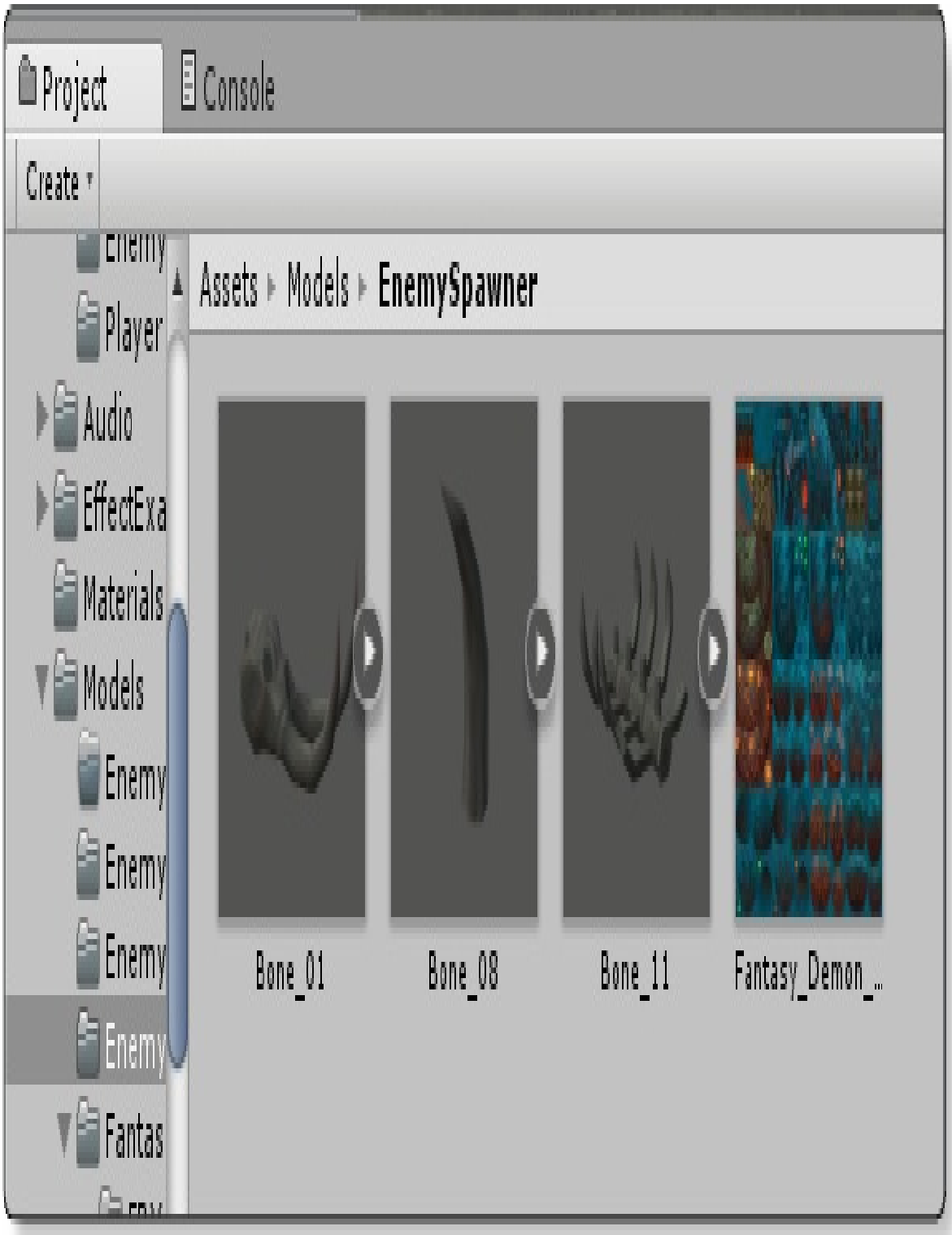
Expansión del nivel

De la carpeta “models” dentro del folder “resources” del capítulo, importe los modelos y la textura que se indica.



Nombre	Fecha de modifica...	Tipo	Tamaño
Bone.01	25/06/2016 11:51	3D Object	31 KB
Bone.08	25/06/2016 11:51	3D Object	27 KB
Bone.11	25/06/2016 11:51	3D Object	39 KB
Fantasy_Demon_Cave_Atlas	25/06/2016 11:14	Archivo TGA	49.153 KB

Folder models dentro de “resources”

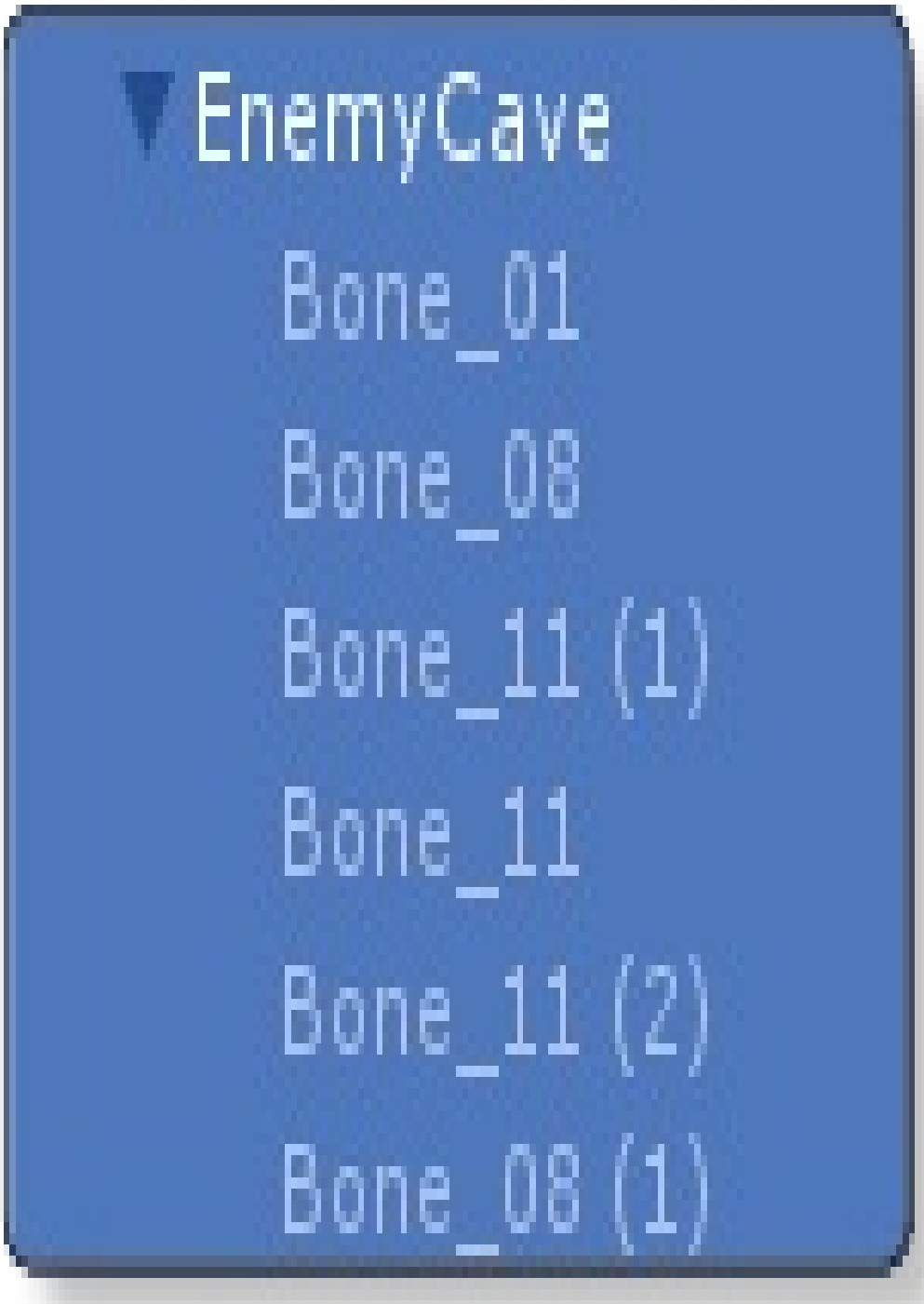

Project
Console
Create
Assets
Models
EnemySpawner
Player
Audio
Materials
Models
Enemy
Enemy
Enemy
Enemy
Bone_01
Bone_08
Bone_11
Fantasy_Demon_

Importación de modelos y texturas

Con los conceptos aprendidos haga la estructura que se muestra y asegúrese de texturizar los modelos.

Construcción de enemigo Spawner

Haga un nuevo “EmptyObject”, renómbrelo “EnemyCave” y hágalo padre de todos los elementos que conforman el enemigo.


EnemyCave
Bone_01
Bone_08
Bone_11 (1)
Bone_11
Bone_11 (2)
Bone_08 (1)

Emparentamiento de elementos

Con el objeto “EnemyCave” seleccionado escálelo.

KnifeSpawn

Colocación correcta de objeto EnemyCave

NOTA:

El pivote del objeto “EnemyCave” debe de estar en la base del objeto.

Seleccione el objeto “EnemyCave” y asigne su componente BoxCollider con los valores que se muestran.

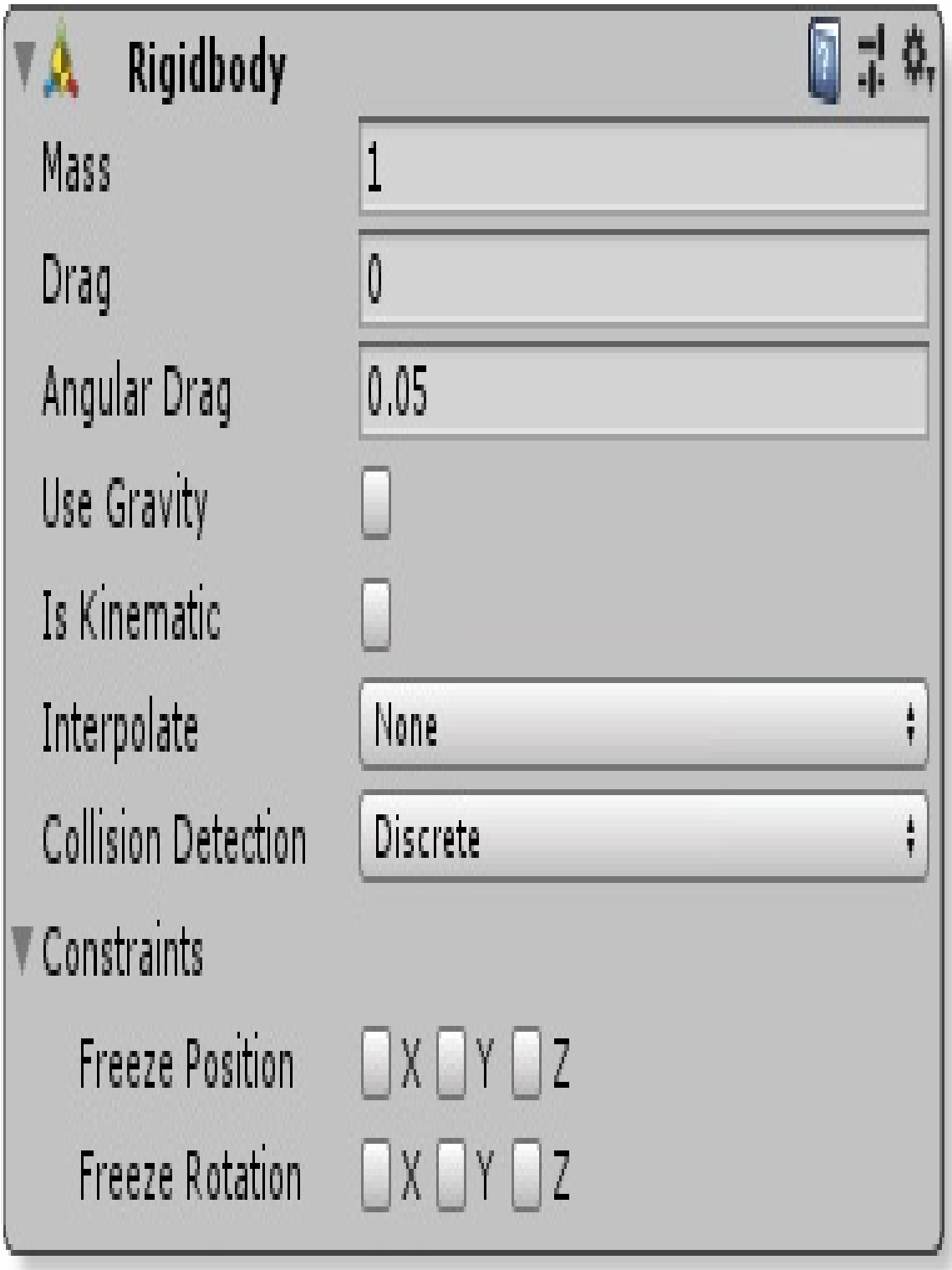
KnifeSpawn

BoxCollider.

Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0.67
Y 2.28
Z 0
Size
X 2.36
Y 4.4
Z 3.76

Valores de BoxCollider

Asigne un componente “RigidBody” con los valores que se muestran.


Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Z
Freeze Rotation
X
Y
Z

Valores de RigidBody.

Creación de las animaciones para el enemigo tipo “Spawner”

En este apartado trabajaremos creando las animaciones de daño y muerte de nuestro enemigo tipo Spawner.

Anteriormente hemos trabajado con animaciones ya existentes para nuestros personajes. En este caso no las tenemos ya que hemos creado un nuevo enemigo particular.

Conoceremos la ventana “Animation” para hacer las animaciones respectivas con llaves de animación para posteriormente guardarlas como clips de animación y mandarlas llamar con “Animator”.

Seleccione el objeto EnemyCave y abra la ventana Animation (Window / Animation / Animation) y clic en el botón “Create” para hacer una nueva animación.

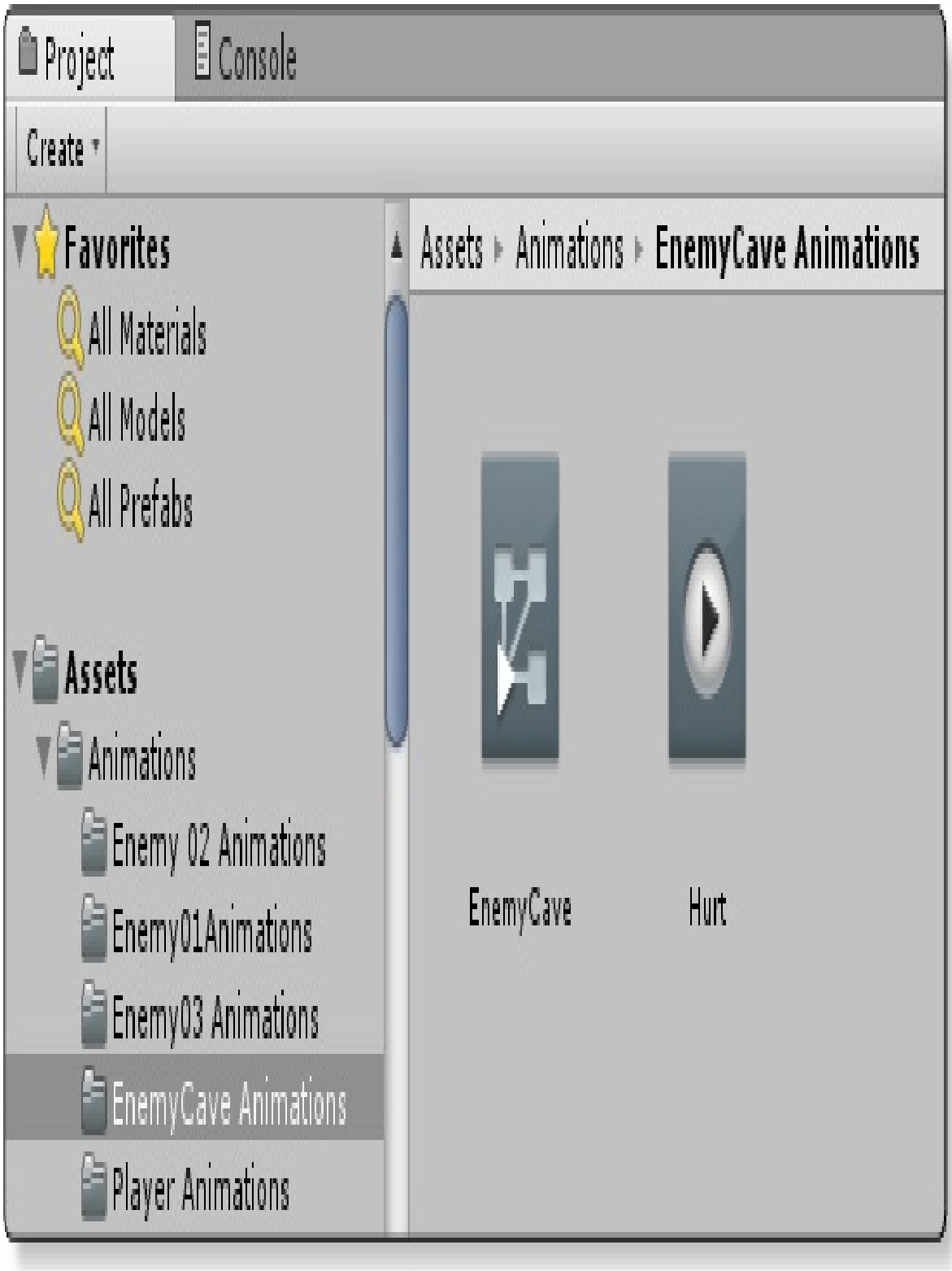
Animation
Preview
Sample
To begin animating EnemyCave, create an Animation Clip.
Create

Ventana Animation

Asigne el nombre “Hurt” a la animación.

NOTA:

Se creará un AnimatorController y la animación Hurt automáticamente.


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
Enemy 02 Animations
Enemy01Animations
Enemy03 Animations
EnemyCave Animations
Player Animations
Assets ▸ Animations ▸ EnemyCave Animations
EnemyCave
Hurt

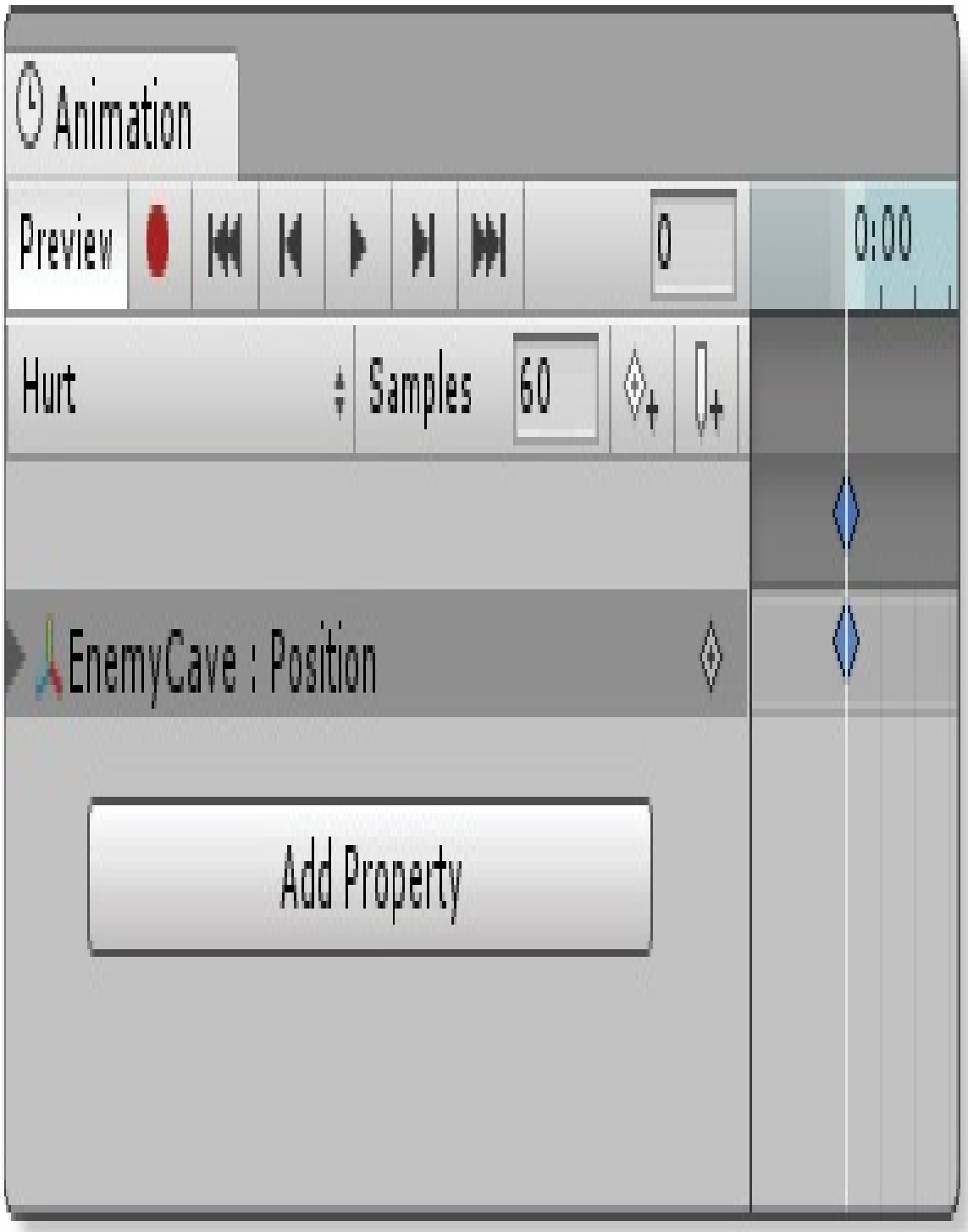
Creación de AC y de clip.

Para la animación de “Hurt” vamos a mover un par de veces el objeto de atrás hacia adelante, para ello en la ventana de “Animation” debemos de asignar una propiedad o canal de transformación conforme a su posición.

Clic en el botón “Add Property”, expanda Transform y con el icono (“+”) relativo a su posición añada el canal.

Animation
Preview
0
0:00
0:05
0:10
Hurt
Samples
60
Add Property
Transform
Position
Rotation
Scale
Box Collider

Asignación de propiedad de posición.


Animation
Preview
0
0:00
Hurt
Samples
60
EnemyCave : Position
Add Property

Propiedad de posición y llave inicial.

NOTA:

Automáticamente se creará la primera llave de animación para el canal o propiedad seleccionado, representada por un rombo azul.

A continuación, se deberá de crear la segunda llave con otra posición en otro momento de la línea de tiempo.

Accione el icono de grabación (Enable/ disable keyframe recording mode).

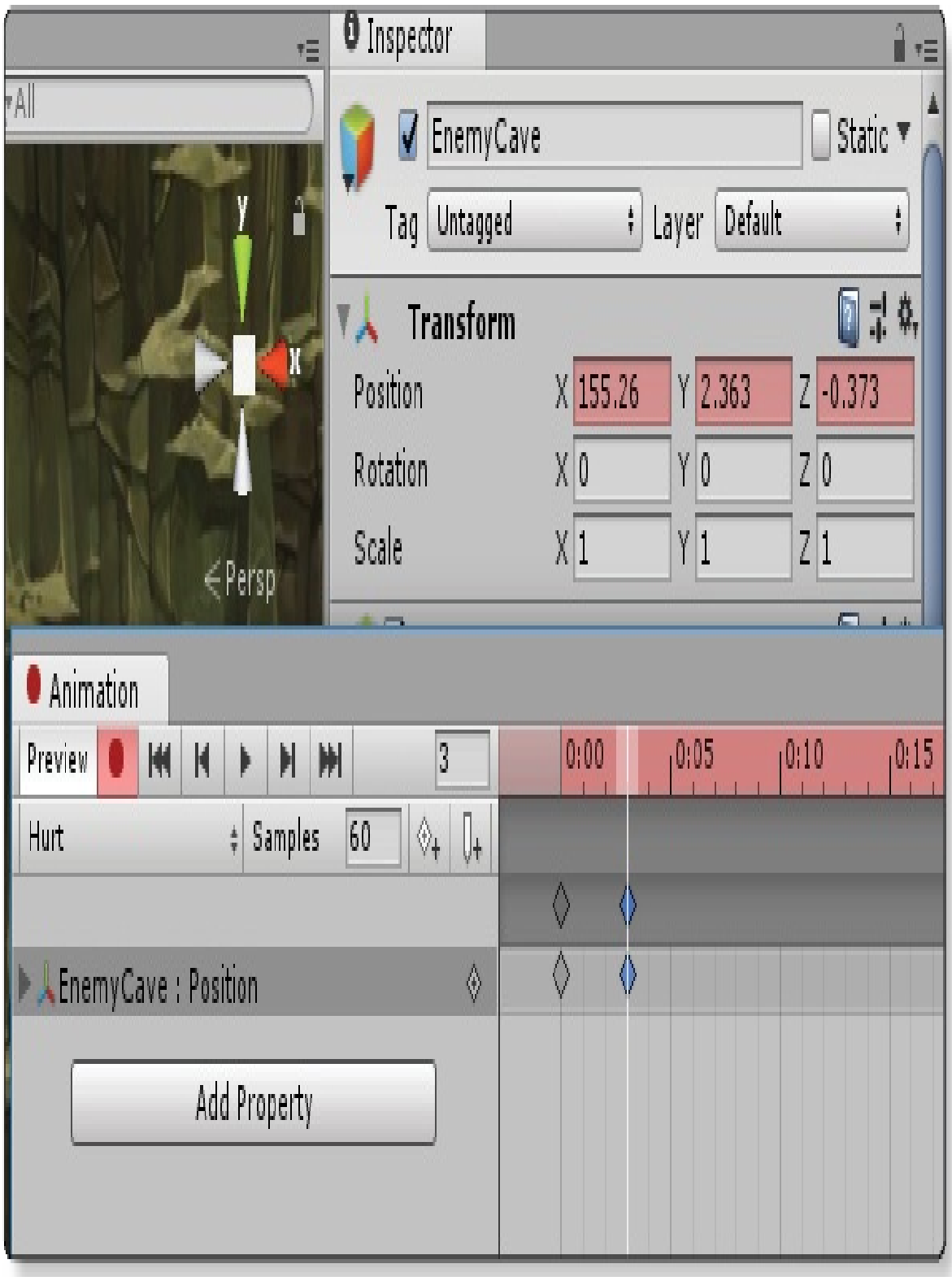
Animation
Preview
0
0:00
Samples
60
Enable/disable keyframe recording mode.
EnemyCave : Position
Add Property

Botón de grabación.

NOTA:

Al activar el icono de grabación podrá observar que la parte superior del canal se vuelve roja, indicando que cualquier movimiento en otro lapso de tiempo se convertirá automáticamente en llave.

Vaya al cuadro 3 y mueva ligeramente hacia la derecha el objeto EnemyCave.


Inspector
All
EnemyCave
Static
Tag
Untagged
Layer
Default
Transform
Position
X 155.26
Y 2.363
Z -0.373
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Persp
Animation
Preview
3
0:00
0:05
0:10
0:15
Hurt
Samples
60
EnemyCave : Position
Add Property

Creación de la segunda llave

NOTA:

Al mover hacia la derecha EnemyCave podemos observar que:

Se crea la segunda llave de animación.

Los indicadores de Transform en Inspector se vuelven rojos indicando una animación existente.

KnifeSpawn

Movimiento de objeto hacia la derecha

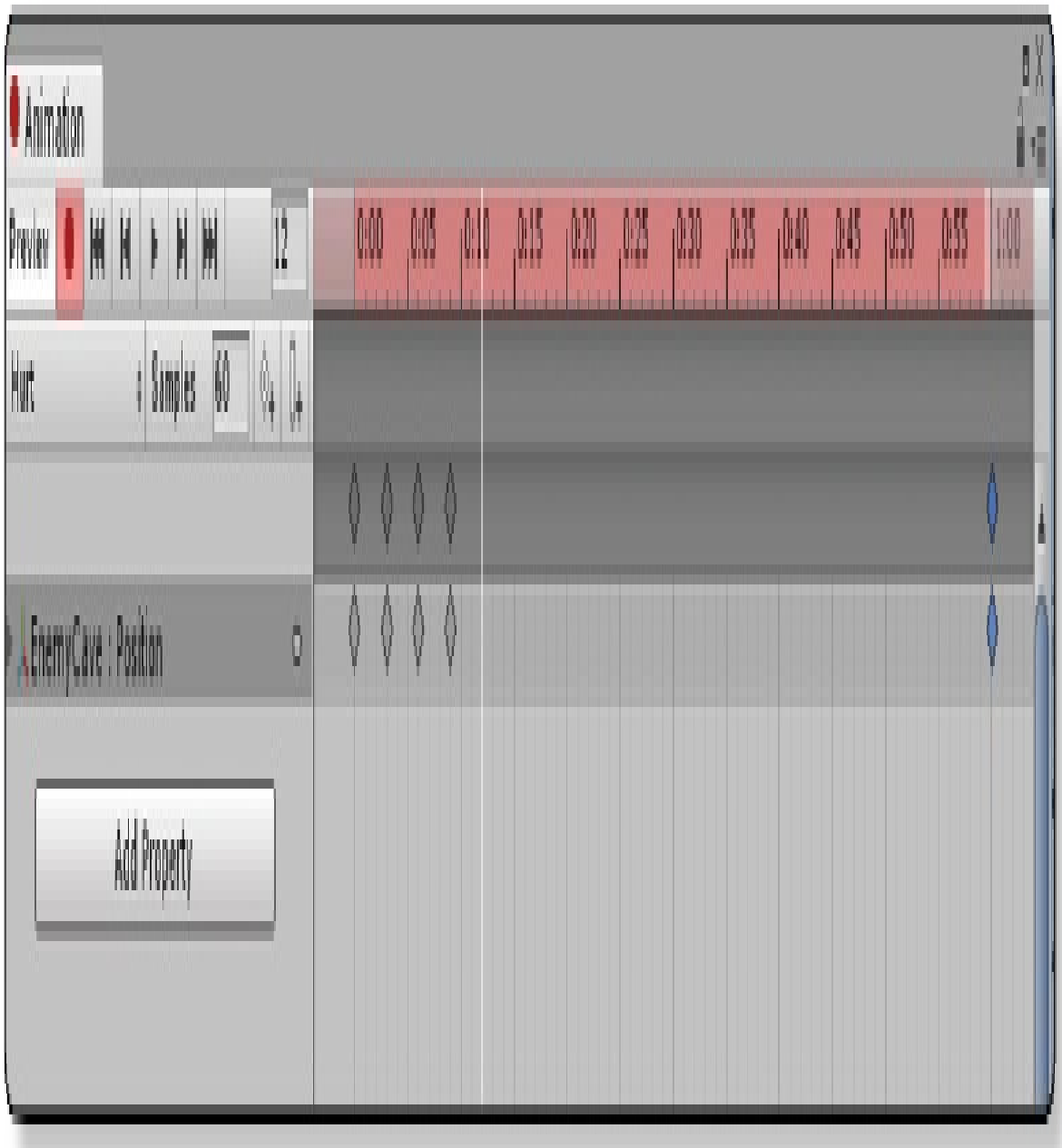
En la ventana “Animation” seleccione las dos llaves creadas y cópielas (Control + C.)

Coloque el puntero en el tiempo 6 y pegue las llaves.

Animation
Preview
32
0:00
0:05
0:10
Hurt
Samples
60
EnemyCave : Position
Add Property

Pegado de llaves.

Seleccione la última llave y arrástrela al tiempo 12.


Animation
Preview
Hunt
Samples
0:00
0:05
0:10
0:15
0:20
0:25
0:30
0:35
0:40
0:45
0:50
0:55
1:00
EnemyCave : Position
Add Property

Visualización de la última llave.

Animation
Preview
Hunt
Samples
Add Property

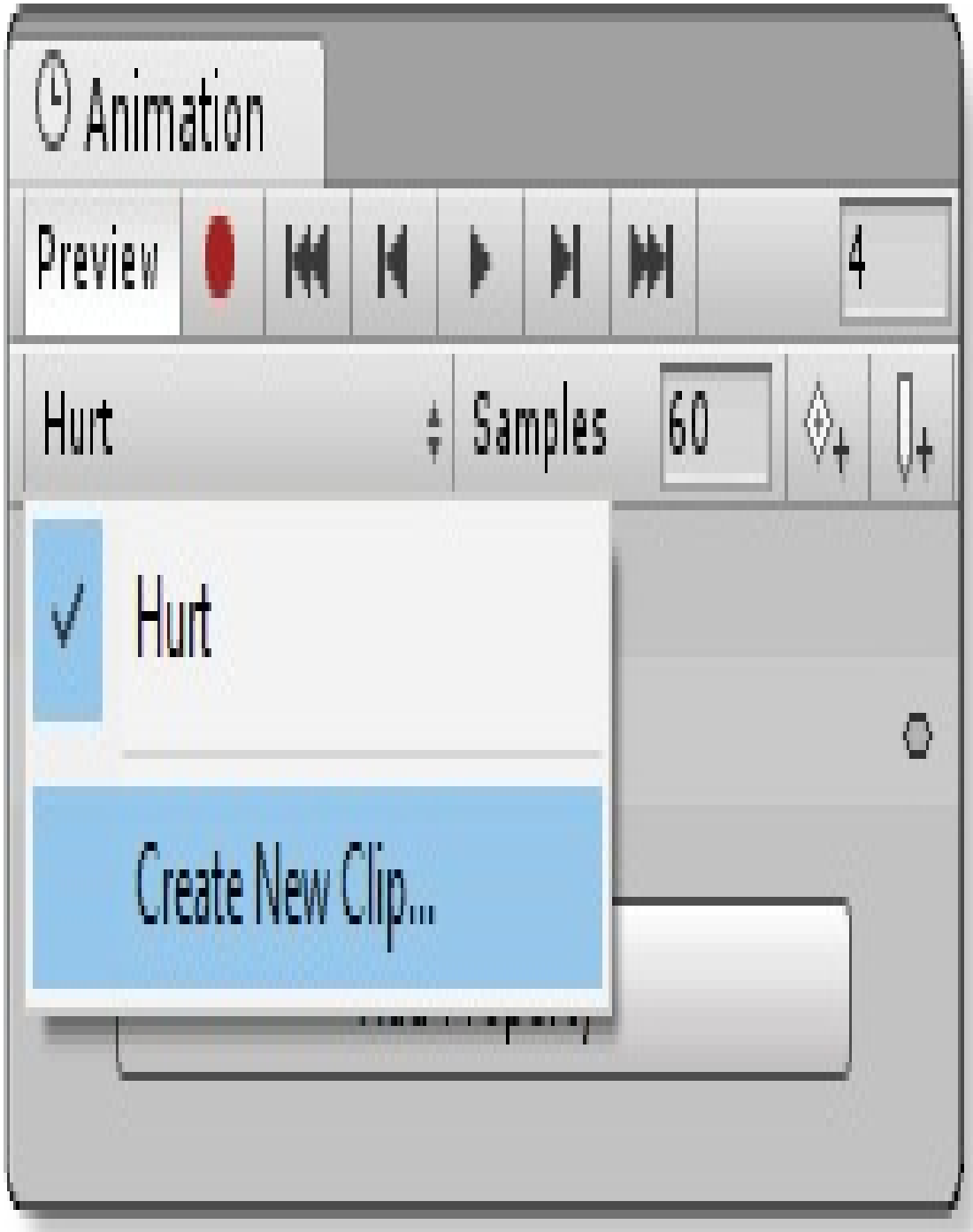
Reposición de la última llave

Apague el botón de grabación.

Previsualice la animación con el botón “Play” o de reproducción de la ventana “Animation”.

De esta manera ya hemos completado la animación de “Hurt” a continuación crearemos la animación de “EnemyDie” modificando la transformación y rotación de todos los elementos que conforman el objeto “EnemyCave” para crear la animación de su destrucción.

Pulse la animación de “Hurt” en la ventana de “Animation” y clic en el botón “Create New Clip”.


Animation
Preview
4
Hurt
Samples
60
Hurt
Create New Clip...

Creación de un nuevo Clip de animación

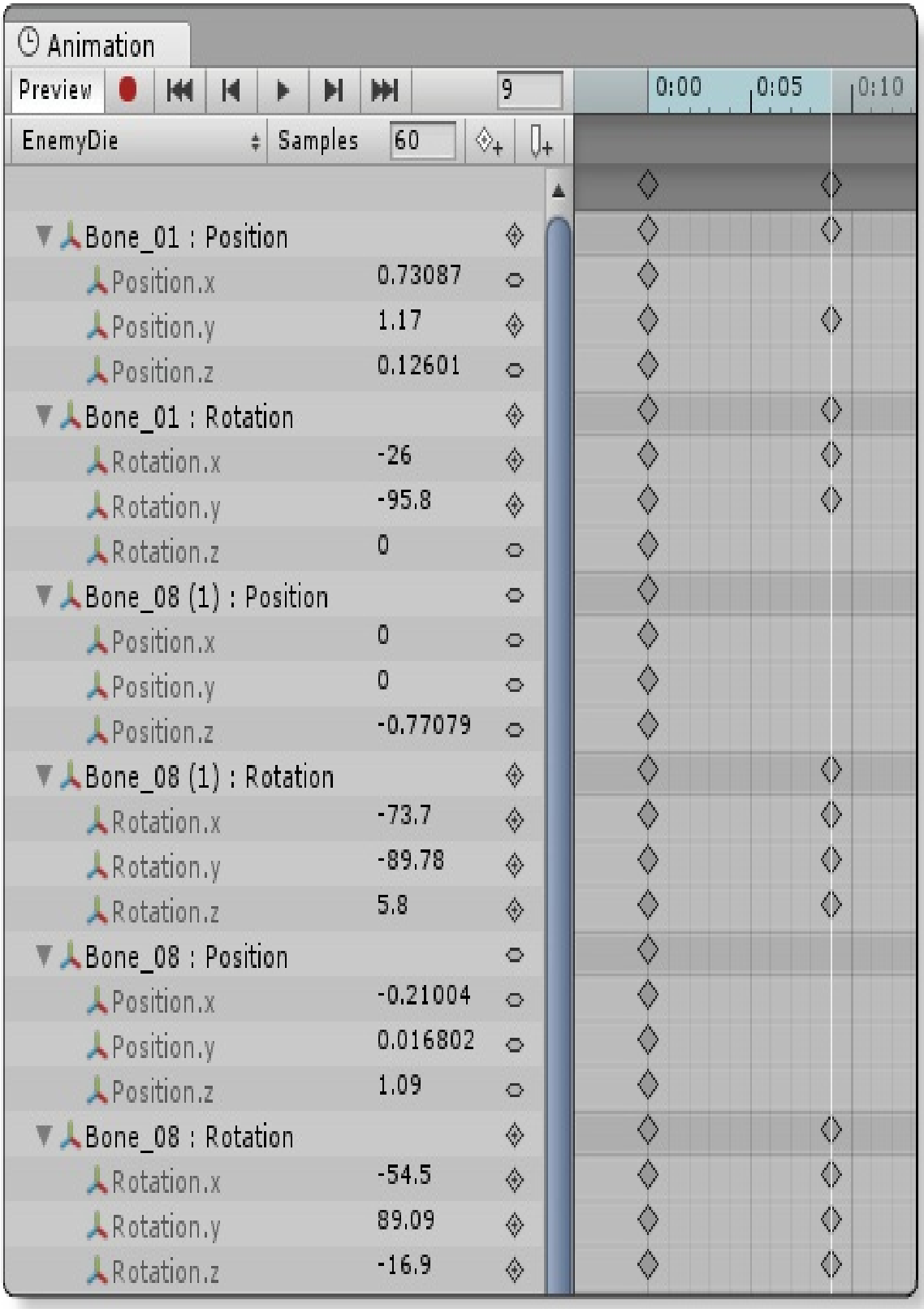
Nombre el nuevo clip como “EnemyDie”.

Con los conceptos aprendidos haga las llaves necesarias para mover todos los elementos que conforman EnemyCave en solo dos pasos, o bien dos llaves. Las de sus posiciones y rotaciones iniciales y sus posiciones y rotaciones finales en el cuadro 9.

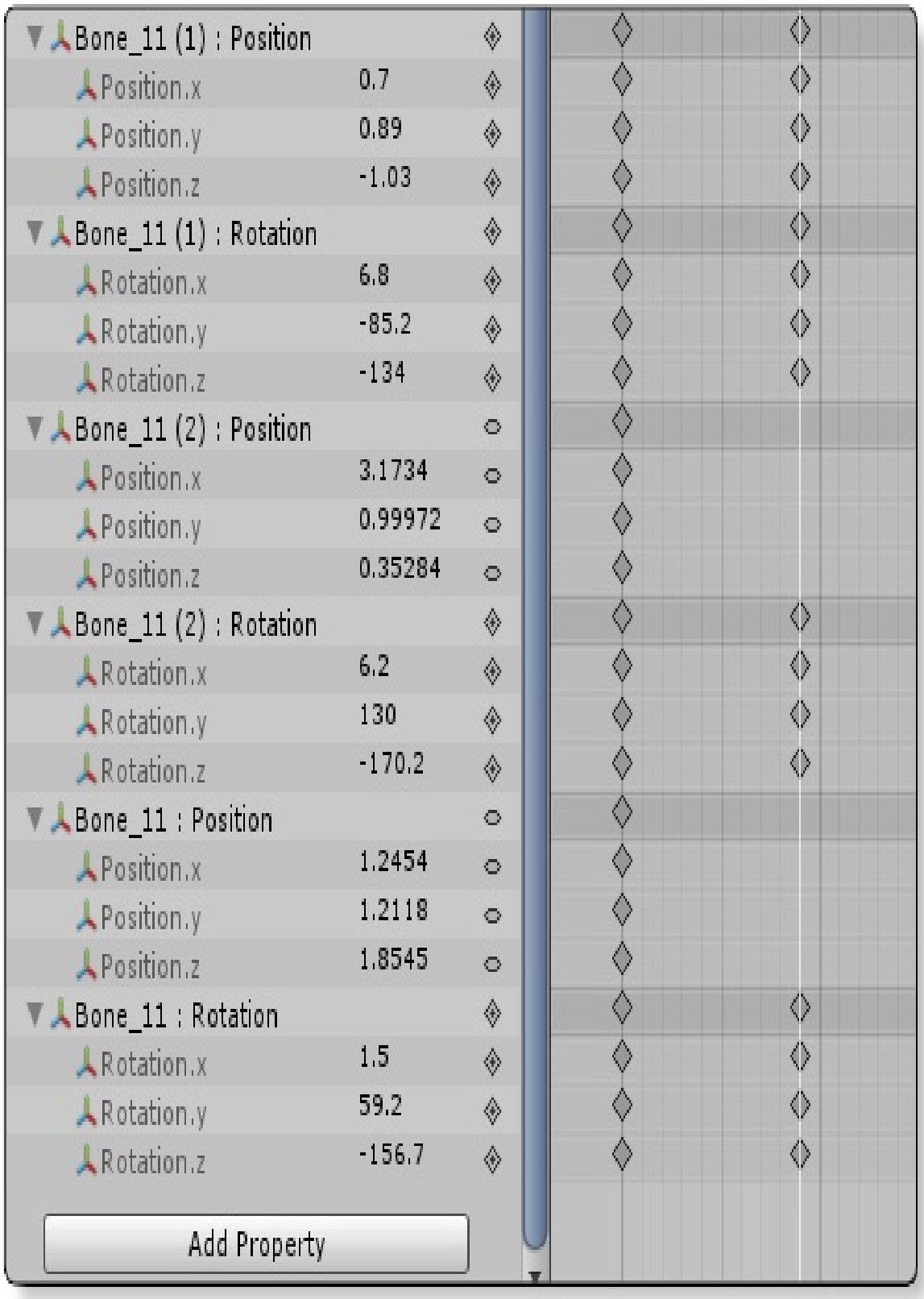
A continuación, se presenta el cuadro guía de las posiciones y rotaciones de cada elemento conforme a sus llaves finales.

KnifeSpawn

Previsualización de ultimas llaves de animación.


Animation
Preview
9
0:00
0:05
0:10
EnemyDie
Samples
60
Bone_01 : Position
Position.x 0.73087
Position.y 1.17
Position.z 0.12601
Bone_01 : Rotation
Rotation.x -26
Rotation.y -95.8
Rotation.z 0
Bone_08 (1) : Position
Position.x 0
Position.y 0
Position.z -0.77079
Bone_08 (1) : Rotation
Rotation.x -73.7
Rotation.y -89.78
Rotation.z 5.8
Bone_08 : Position
Position.x -0.21004
Position.y 0.016802
Position.z 1.09
Bone_08 : Rotation
Rotation.x -54.5
Rotation.y 89.09
Rotation.z -16.9

Ultimas llaves de posición y rotación de objetos (I).


Bone_11 (1) : Position
Position.x 0.7
Position.y 0.89
Position.z -1.03
Bone_11 (1) : Rotation
Rotation.x 6.8
Rotation.y -85.2
Rotation.z -134
Bone_11 (2) : Position
Position.x 3.1734
Position.y 0.99972
Position.z 0.35284
Bone_11 (2) : Rotation
Rotation.x 6.2
Rotation.y 130
Rotation.z -170.2
Bone_11 : Position
Position.x 1.2454
Position.y 1.2118
Position.z 1.8545
Bone_11 : Rotation
Rotation.x 1.5
Rotation.y 59.2
Rotation.z -156.7
Add Property

Ultimas llaves de posición y rotación de objetos (II.)

Previsualice la animación con el botón de reproducción de Animation.

Programación de las animaciones para el enemigo tipo “Spawner”

En este apartado programaremos los comportamientos del enemigo tipo “Spawner” para que:

Cada vez que sea lastimado que corra su animación de “Hurt”.

Programar un sistema de salud para el enemigo “EnemyCave”.

Al ser destruido el enemigo, correrá su animación de “EnemyDie”, se destruirá y arrojará un ítem.

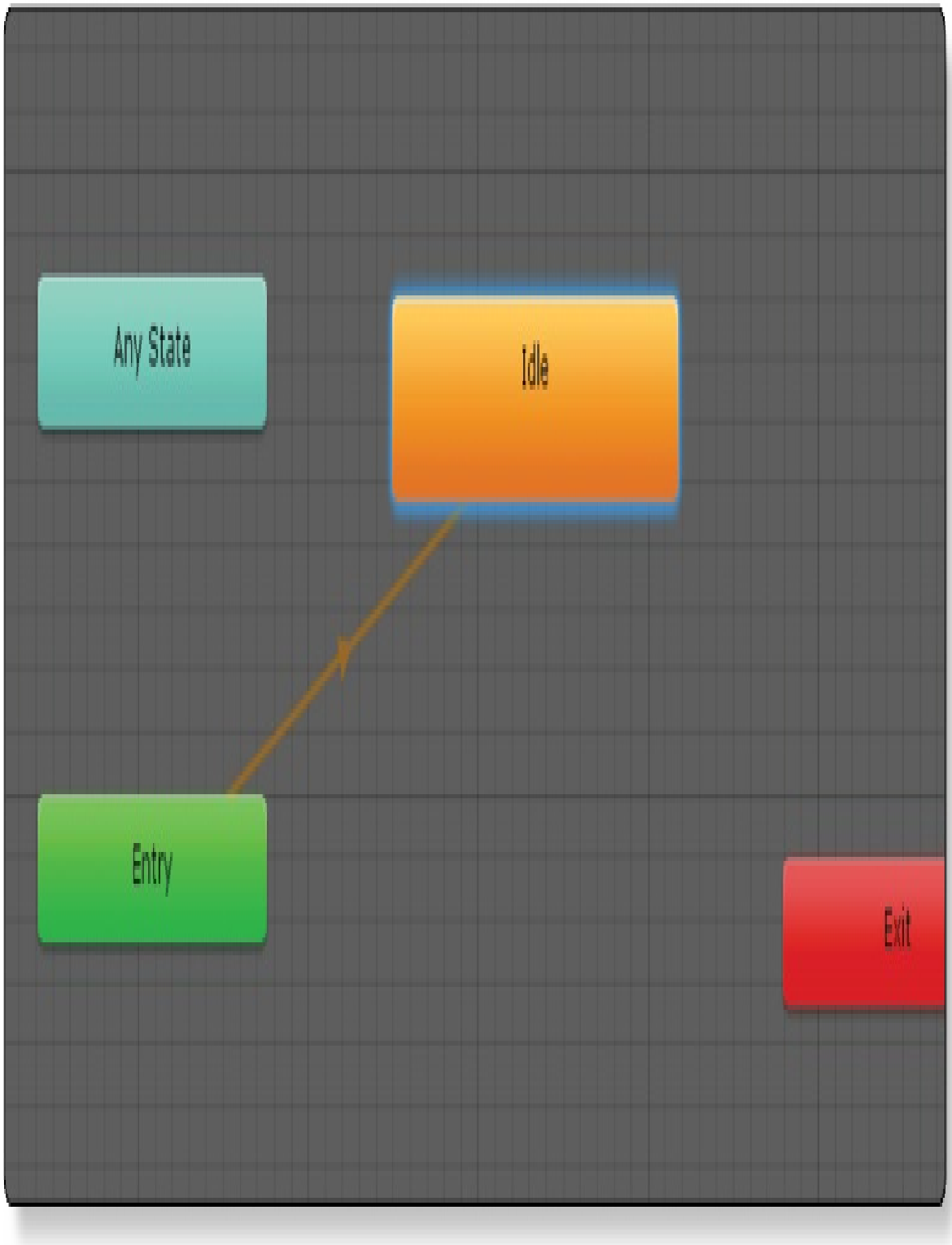
Se añadirán audios a sus comportamientos.

Asigne a EnemyCave un componente Animator y asigne su controlador.

Animator
Controller
EnemyCave
Avatar
None (Avatar)
Apply Root Motion

Asignación de Animation Controller.

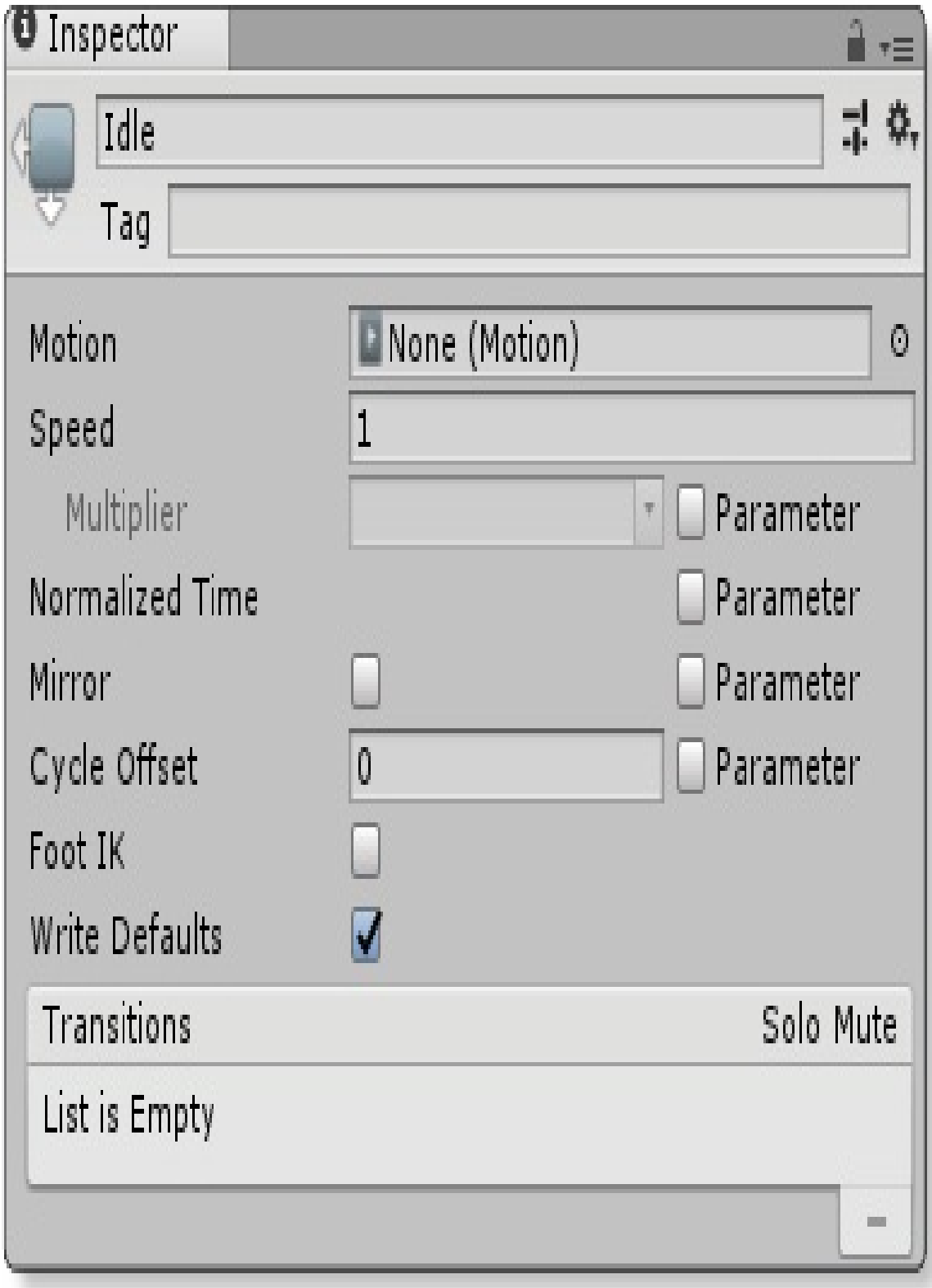
Con EnemyCave seleccionado vaya a “Animator”, haga un nuevo “State”, nómbrelo “Idle”.


Any State
Idle
Entry
Exit

Creación de estado Idle

NOTA:

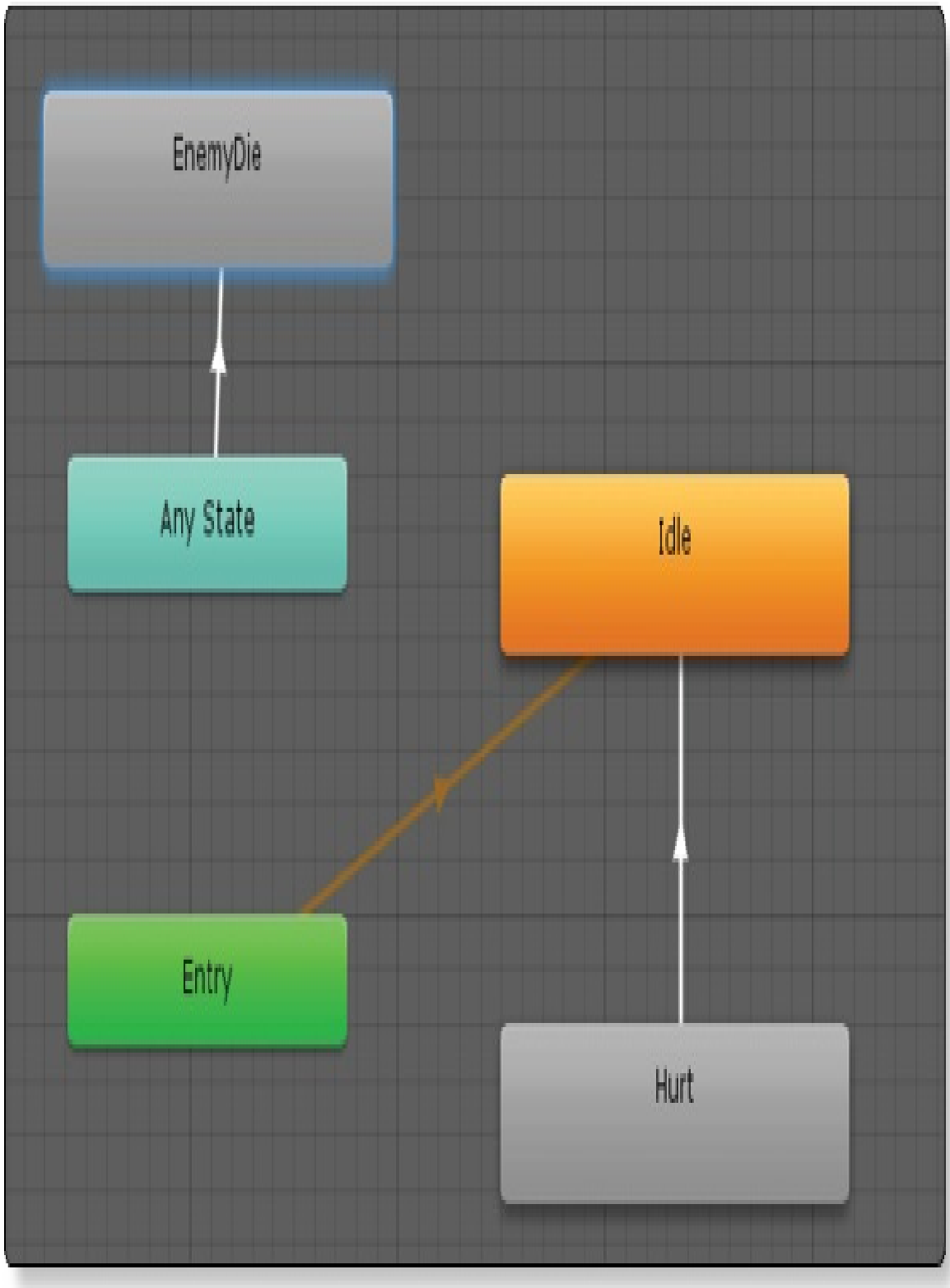
Al ser el único estado, éste tendrá el color naranja referente a su estado por “default”.


Inspector
Idle
Tag
Motion
None (Motion)
Speed
1
Multiplier
Parameter
Normalized Time
Parameter
Mirror
Parameter
Cycle Offset
0
Parameter
Foot IK
Write Defaults
Transitions
Solo Mute
List is Empty

Valores del estado Idle.

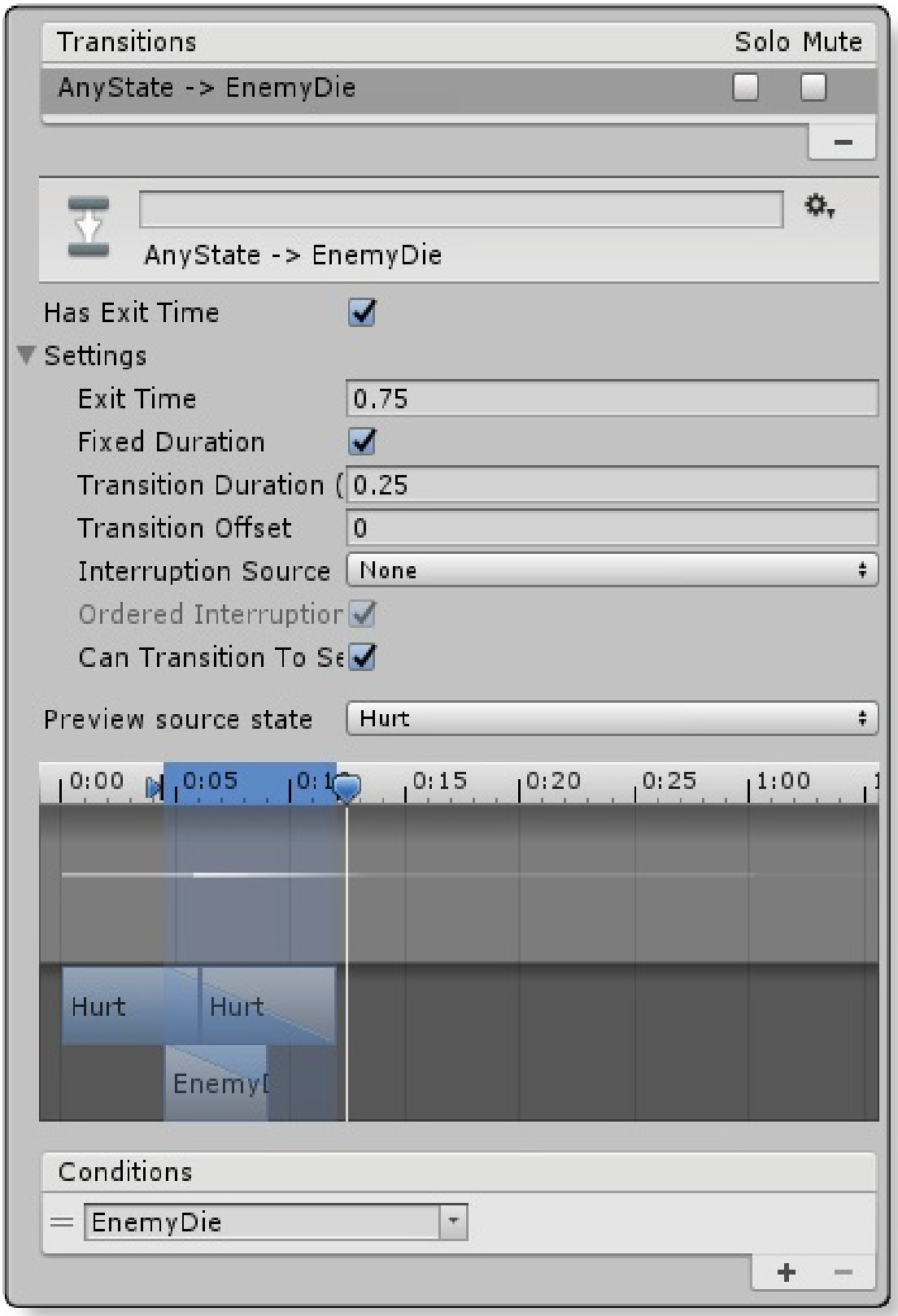
Añada el clip de animación de “Hurt” con una transición de “Hurt” a “Idle” sin parámetro.

Añada el clip de “EnemyDie” , añadiendo una transición de “AnyState” a “EnemyDie”.


EnemyDie
Any State
Idle
Entry
Hurt

Asignación de clips de animación en “Animator”.

Haga un parámetro tipo “Trigger”, nómbrelo “EnemyDie” y asígnelo a la nueva transición con los valores que se muestran.


Transitions
Solo
Mute
AnyState -> EnemyDie
AnyState -> EnemyDie
Has Exit Time
Settings
Exit Time
0.75
Fixed Duration
Transition Duration (
0.25
Transition Offset
0
Interruption Source
None
Ordered Interruptior
Can Transition To Se
Preview source state
Hurt
0:00
0:05
0:15
0:20
0:25
1:00
Hurt
Hurt
Enemy[
Conditions
EnemyDie

Valores de transición de muerte.

Haga un nuevo C#Script, nómbrelo “Cave01Health” y asígnelo al objeto “EnemyCave”.

Abra el script, escriba e inicialice las variables que se muestran.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Cave01Health : MonoBehaviour {

    [SerializeField] private int startingHealth = 50;
    [SerializeField] private float timeSinceLastHit = 0.2f;
    [SerializeField] private float dissapearSpeed = 2f;
    [SerializeField] private int currentHealth;

    private float timer = 0f;
    private Animator anim;
    private bool isAlive;
    private Rigidbody rigidbody;
    private BoxCollider boxCollider;
    private bool dissapearEnemy = false;
    private AudioSource audio;
    public AudioClip hurtAudio;
    public AudioClip killAudio;
    private DropItem dropItem;
    public GameObject explosionEffect;

    public bool IsAlive {
        get {return isAlive; }
    }

    // Use this for initialization
    void Start () {

        rigidbody = GetComponent <Rigidbody> ();
        boxCollider = GetComponent <BoxCollider> ();
        anim = GetComponent <Animator> ();
        isAlive = true;
        currentHealth = startingHealth;
        audio = GetComponent<AudioSource> ();
        dropItem = GetComponent<DropItem> ();
    }
```

Creación e inicialización de variables.

Escriba el siguiente código para la funcionalidad del enemigo.

```
// Update is called once per frame
void Update () {

    timer += Time.deltaTime;
}

void OnTriggerEnter (Collider other) {
    if (timer >= timeSinceLastHit && !GameManager.instance.GameOver) {
        if (other.tag == "PlayerWeapon") {
            takeHit ();
            timer = 0f;
        }
    }
}

void takeHit () {

    if (currentHealth > 0) {

        GameObject newexplosionEffect = (GameObject)Instantiate
        (explosionEffect, transform.position, transform.rotation);
        Destroy (newexplosionEffect, 1);

        anim.Play ("Hurt");
        currentHealth -= 10;
        audio.PlayOneShot (hurtAudio);
    }

    if (currentHealth <= 0) {
        isAlive = false;
        KillEnemy ();
    }
}
```

Código de funcionalidad I.

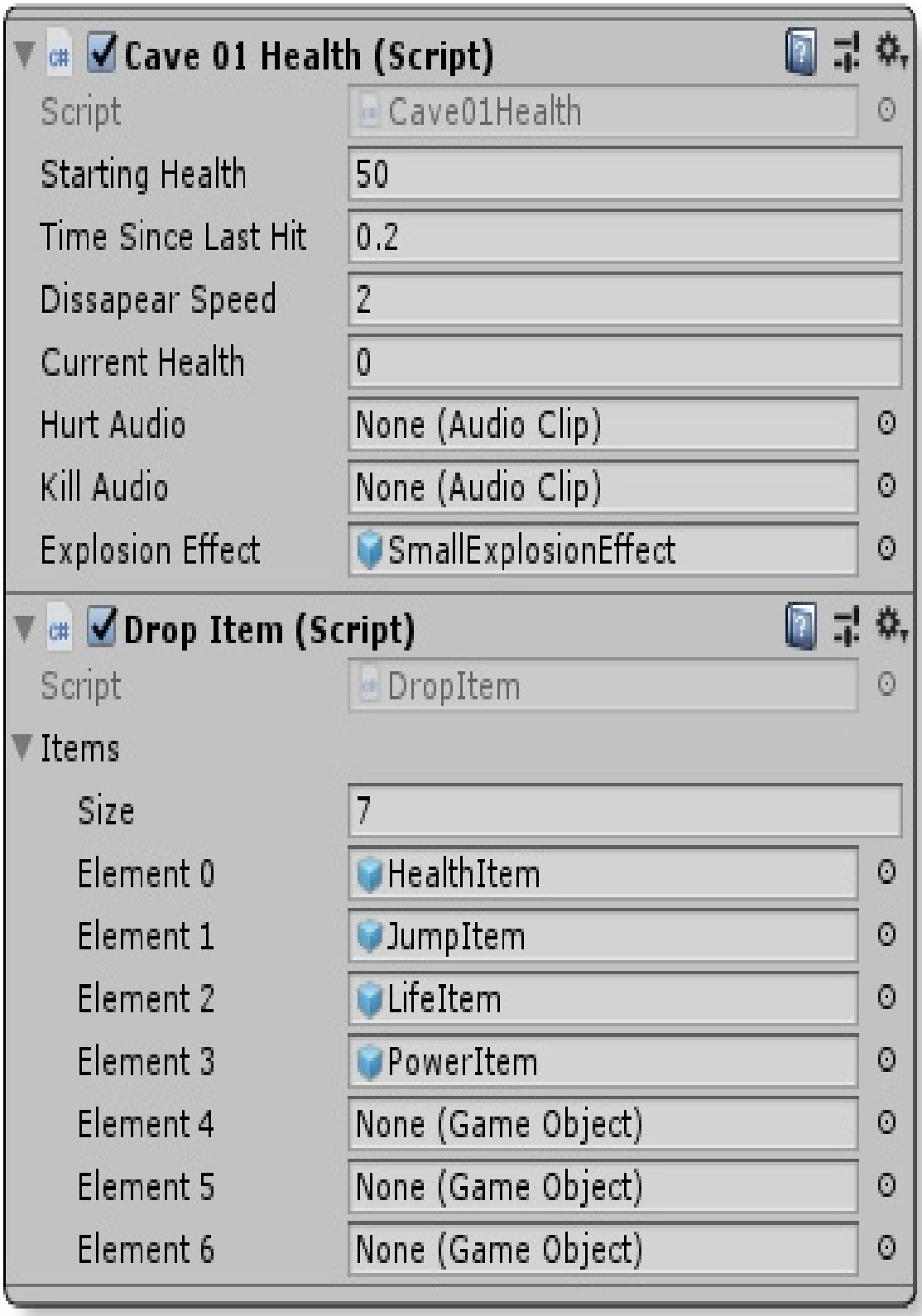
```
    void KillEnemy () {

        boxCollider.enabled = false;
        anim.SetTrigger ("EnemyDie");
        audio.PlayOneShot (killAudio);

        StartCoroutine (removeEnemy ());
        dropItem.Drop ();

    }

    IEnumerator removeEnemy() {

        yield return new WaitForSeconds (2f);
        Destroy (gameObject);
    }
}
```

Código de funcionalidad II.

En Unity asigne el script “EnemyDrop” al objeto “EnemyCave”.

Previsualice el juego para observar el correcto funcionamiento de los scripts.

Previsualización de funcionalidad.


Cave 01 Health (Script)
Script
Cave01Health
Starting Health
50
Time Since Last Hit
0.2
Dissapear Speed
2
Current Health
0
Hurt Audio
None (Audio Clip)
Kill Audio
None (Audio Clip)
Explosion Effect
SmallExplosionEffect
Drop Item (Script)
Script
DropItem
Items
Size
7
Element 0
HealthItem
Element 1
JumpItem
Element 2
LifeItem
Element 3
PowerItem
Element 4
None (Game Object)
Element 5
None (Game Object)
Element 6
None (Game Object)

Valores en Unity para los componentes Script.

NOTA:

Puede crear un “EmptyObject” para asignarlo en los elementos vácios para evitar la alerta “NullObject”.

Creación y programación del enemigo tipo “Spawn”

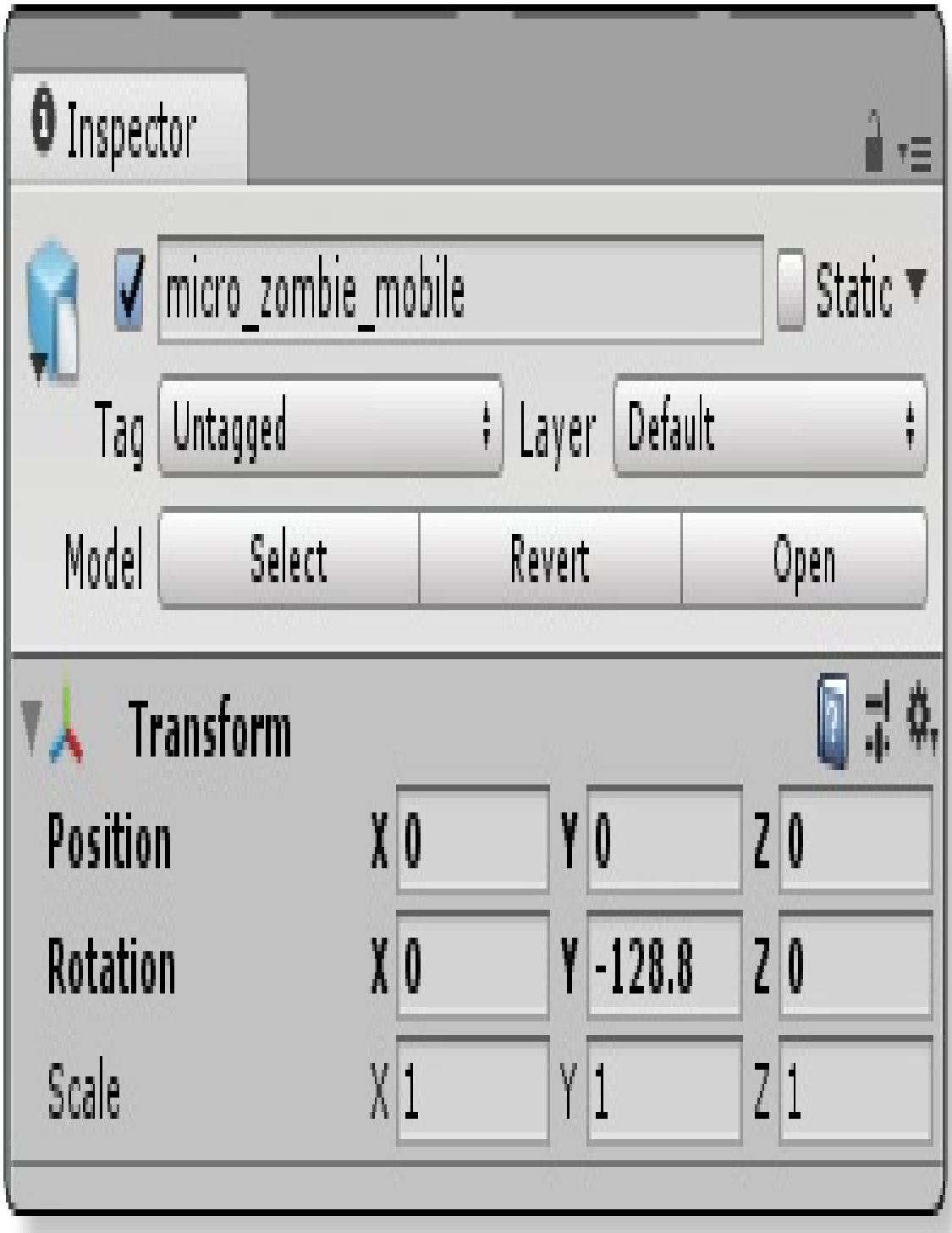
En este apartado crearemos y programaremos el enemigo tipo “Spawn”, mismo que será regenerado infinitamente por el enemigo EnemyCave hasta que este sea destruido.

Con los conceptos aprendidos:

Importe el personaje “Zombie” de la carpeta de “resources” del capítulo a Unity, así como su textura.

Colóquelo en la escena como se muestra.

Colocación de Zombie.


Inspector
micro_zombie_mobile
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y -128.8
Z 0
Scale
X 1
Y 1
Z 1

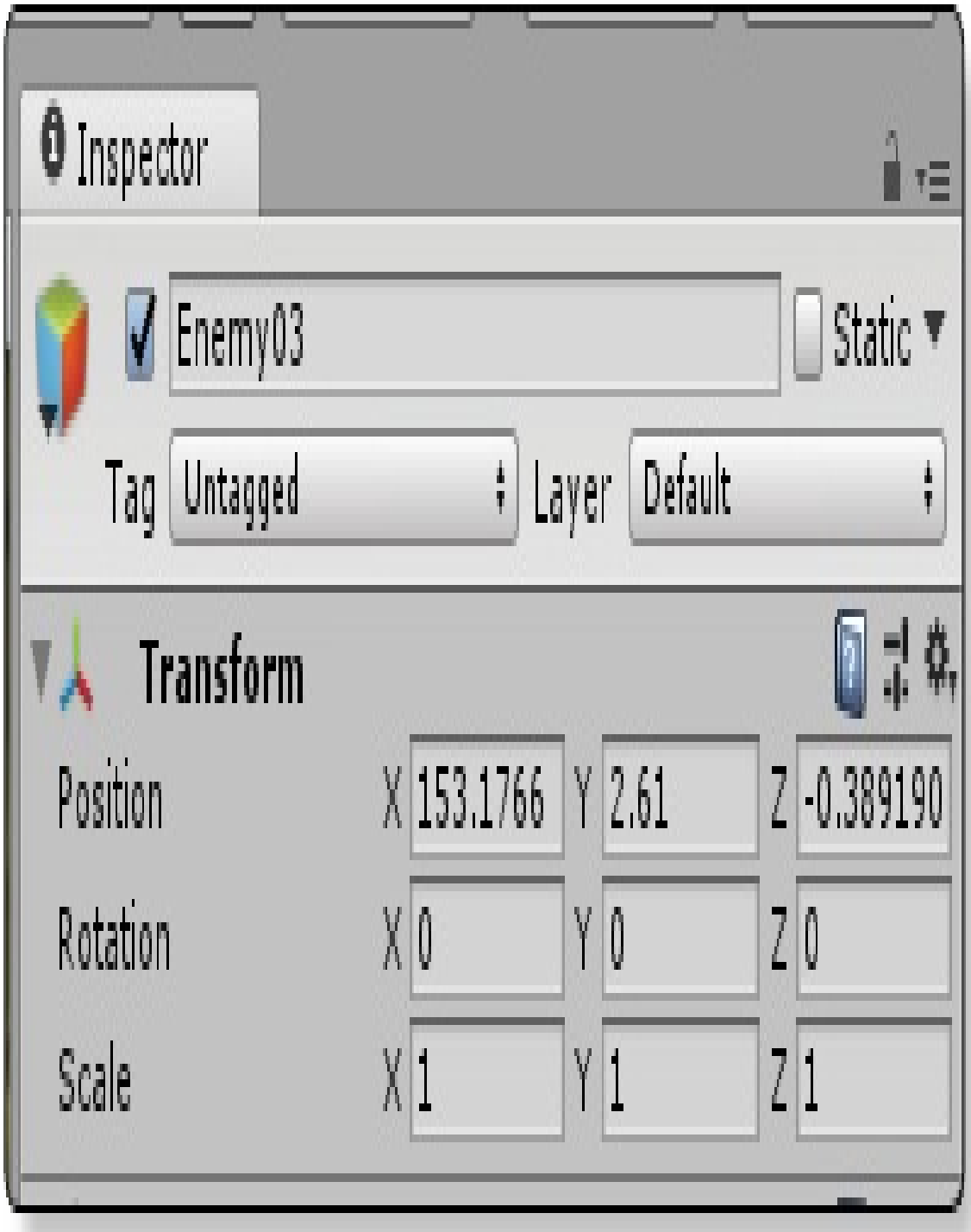
Correcta rotación.

Haga un nuevo “EmptyObject”, renómbrelo “Enemy03” colóquelo en la base del nuevo enemigo y empareje.

Enemy03
micro_zombie_mobile

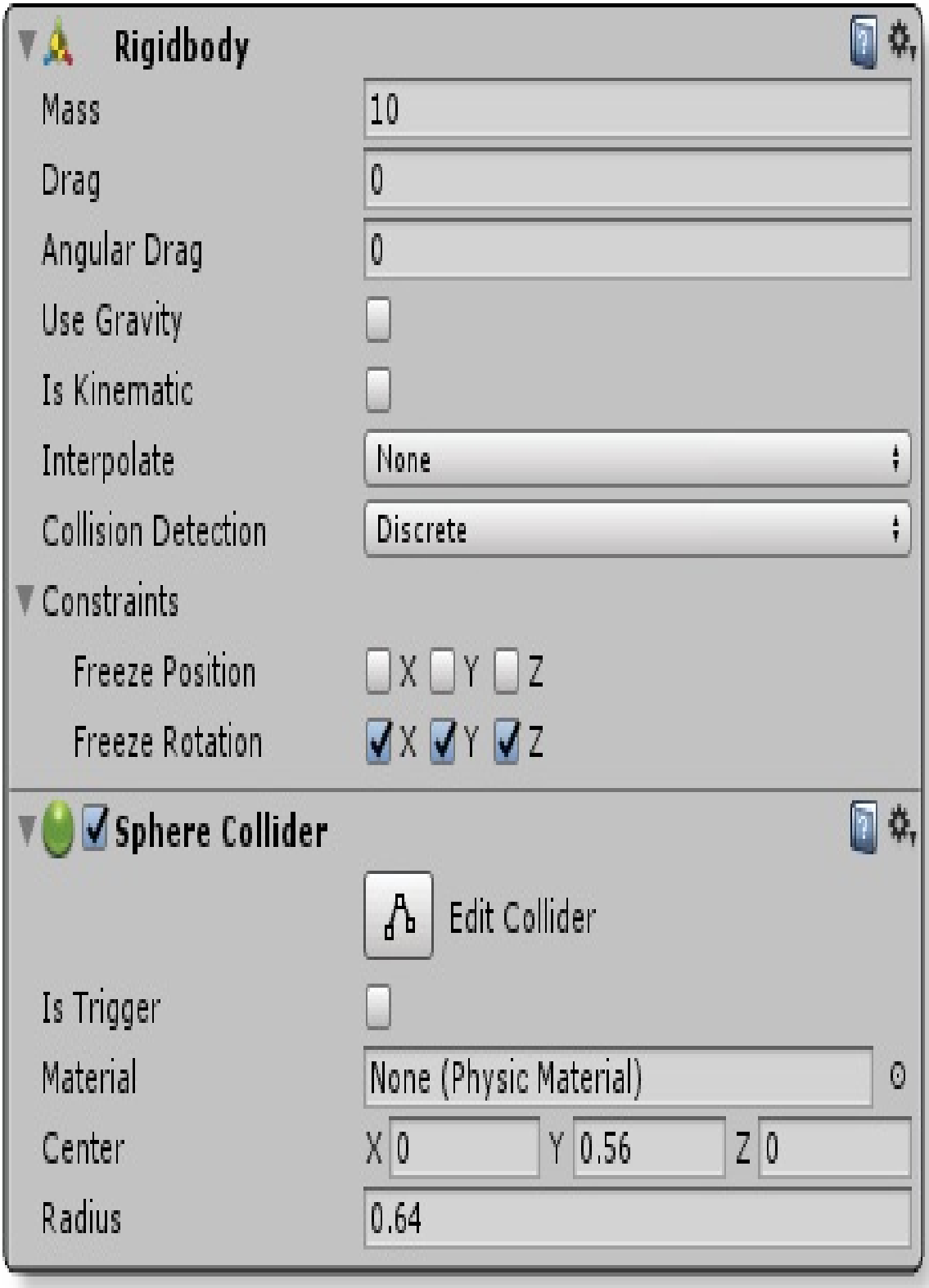
Emparentamiento de Enemy03.

Colocación de Enemy03.

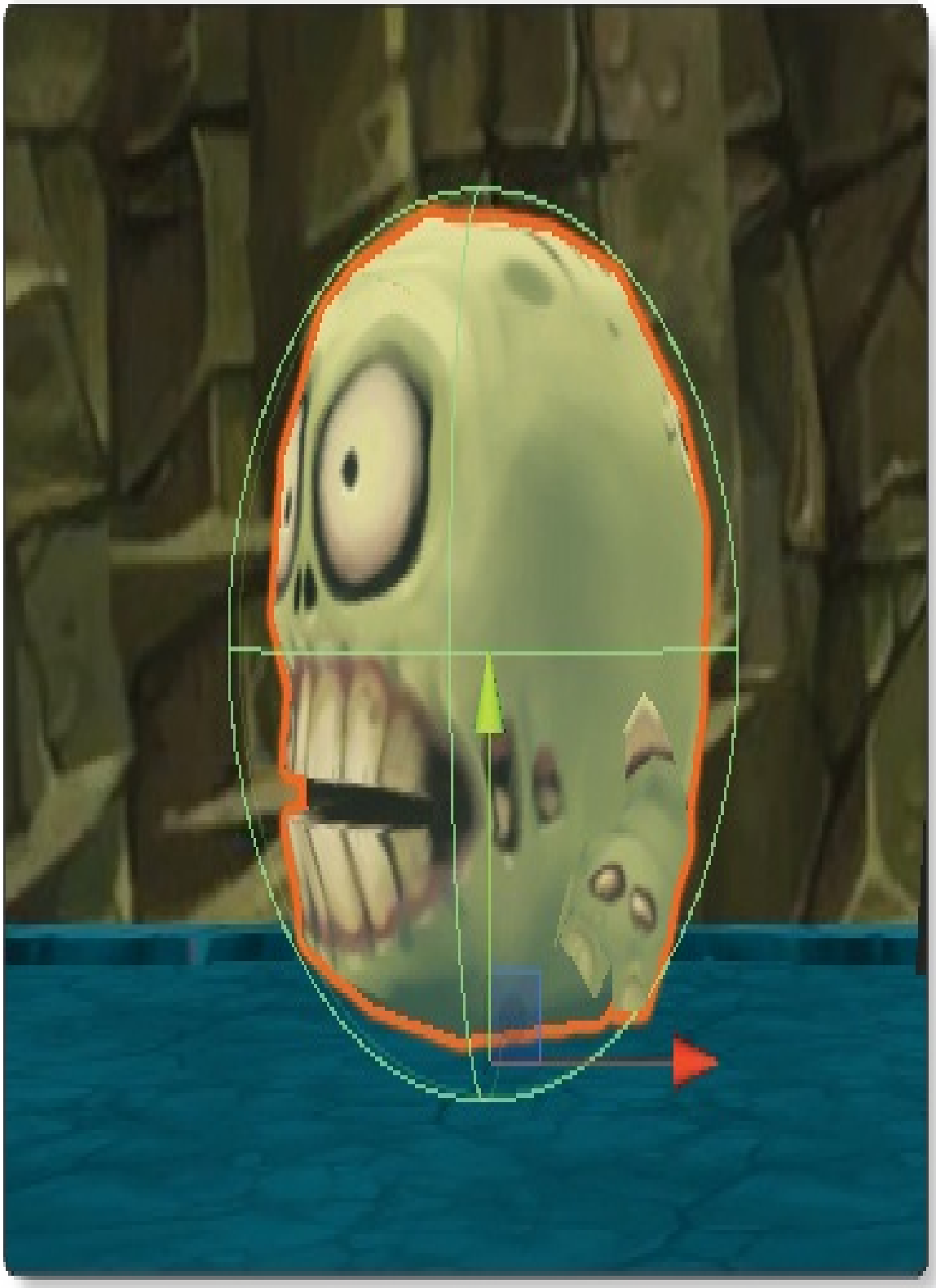

Inspector
Enemy03
Static
Tag
Untagged
Layer
Default
Transform
Position
X 153.1766
Y 2.61
Z -0.389190
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1

Rotaciones en cero de Enemy03.

Asigne los componentes “SphereCollider” y “RigidBody”.


Rigidbody
Mass
10
Drag
0
Angular Drag
0
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X
Y
Z
Freeze Rotation
X
Y
Z
Sphere Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 0.56
Z 0
Radius
0.64

Valores de componentes.



SphereCollider.

Con los conceptos aprendidos, haga los siguientes elementos para Enemy03.

Su AnimationController y nómbrelo “Enemy03AC”.

Su clip de animación relativo a “Bite”

frame 215-245: bite.

Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
Assets
Animations
Enemy02 Animations
Enemy03Animations
Enemy03 Animations
EnemyCave Animations
Assets ▸ Animations ▸ Enemy03 Animations
Enemy03AC
micro_zombie_no...
brian
root
zombi
brian
Box
micro_zombie_...

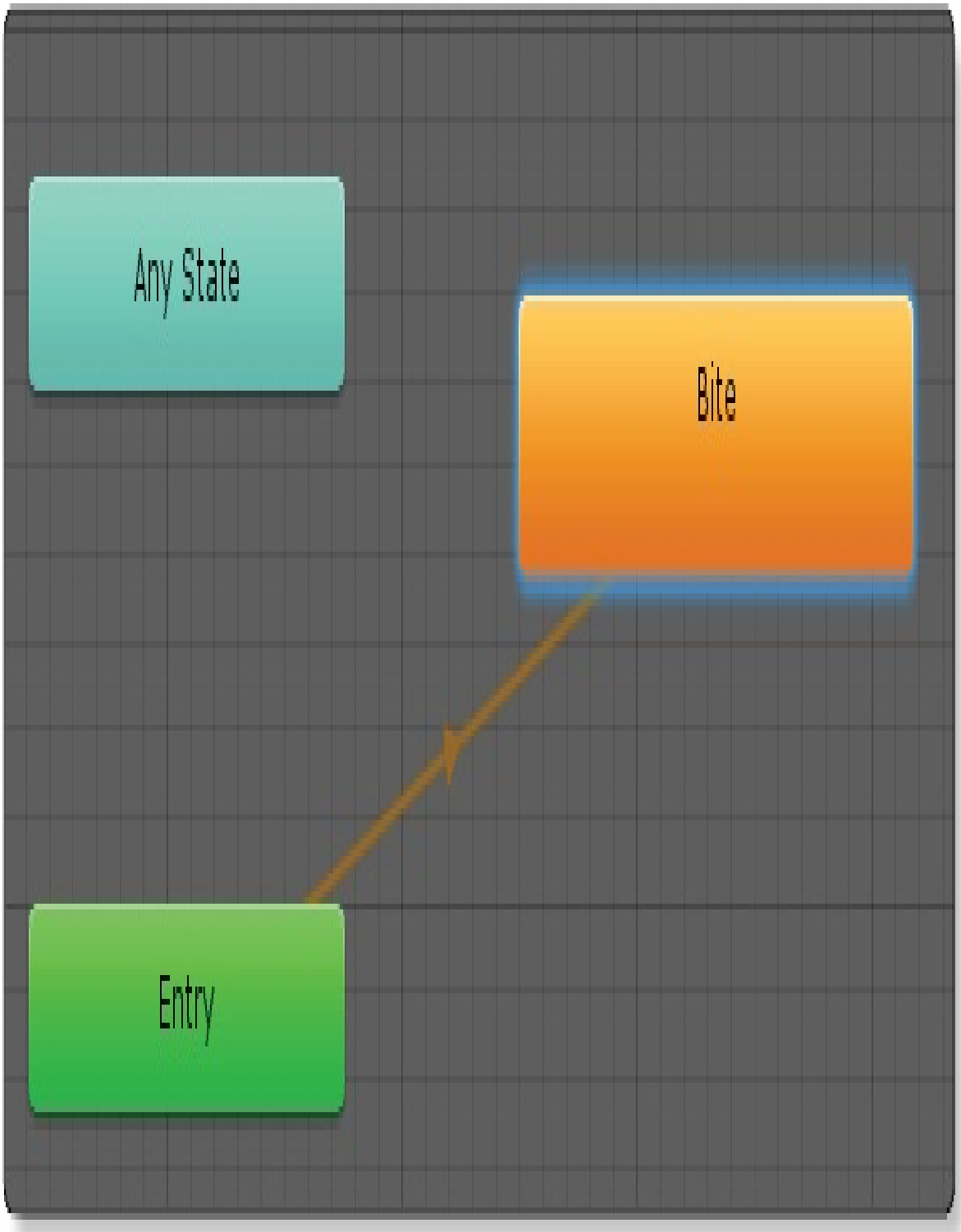
Creación de clip animado Bite.

Con el objeto “Enemy03” seleccionado, asigne un nuevo componente “Animator” y asigne su AnimationController y su “Avatar”.

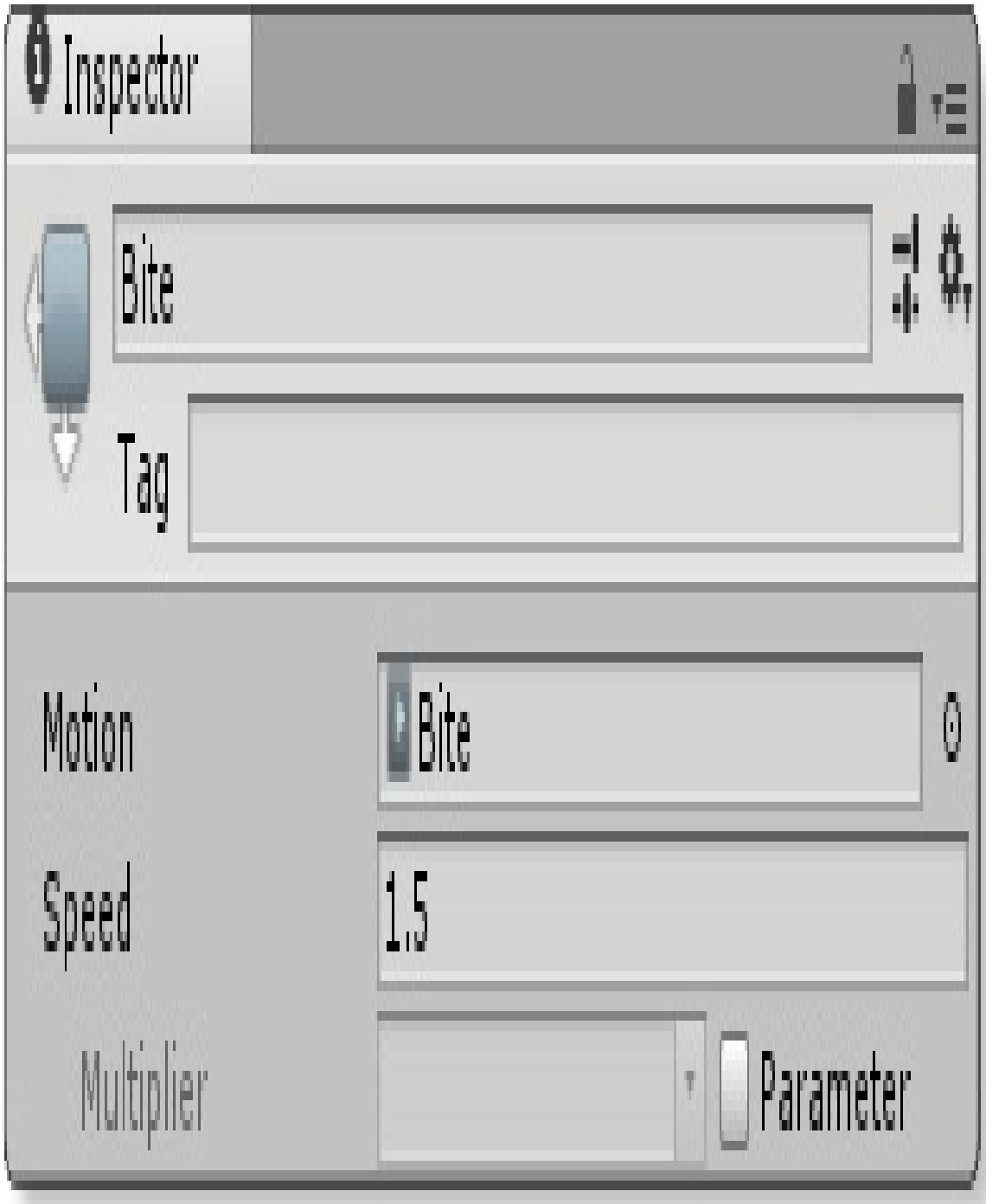
Animator
Controller
Enemy03AC
Avatar
micro_zombie_mobileAvatar
Apply Root Motion
Update Mode
Normal
Culling Mode
Always Animate

Asignación de elementos en componente Animator.

Vaya a “Animator”, asigne como “DefaultState” el clip de animación “Bite” y asigne una velocidad de 1.5.


Any State
Bite
Entry

DefaultState en Animator.


Inspector
Bite
Tag
Motion
Bite
Speed
1.5
Multiplier
Parameter

Cambio de velocidad de clip de animación.

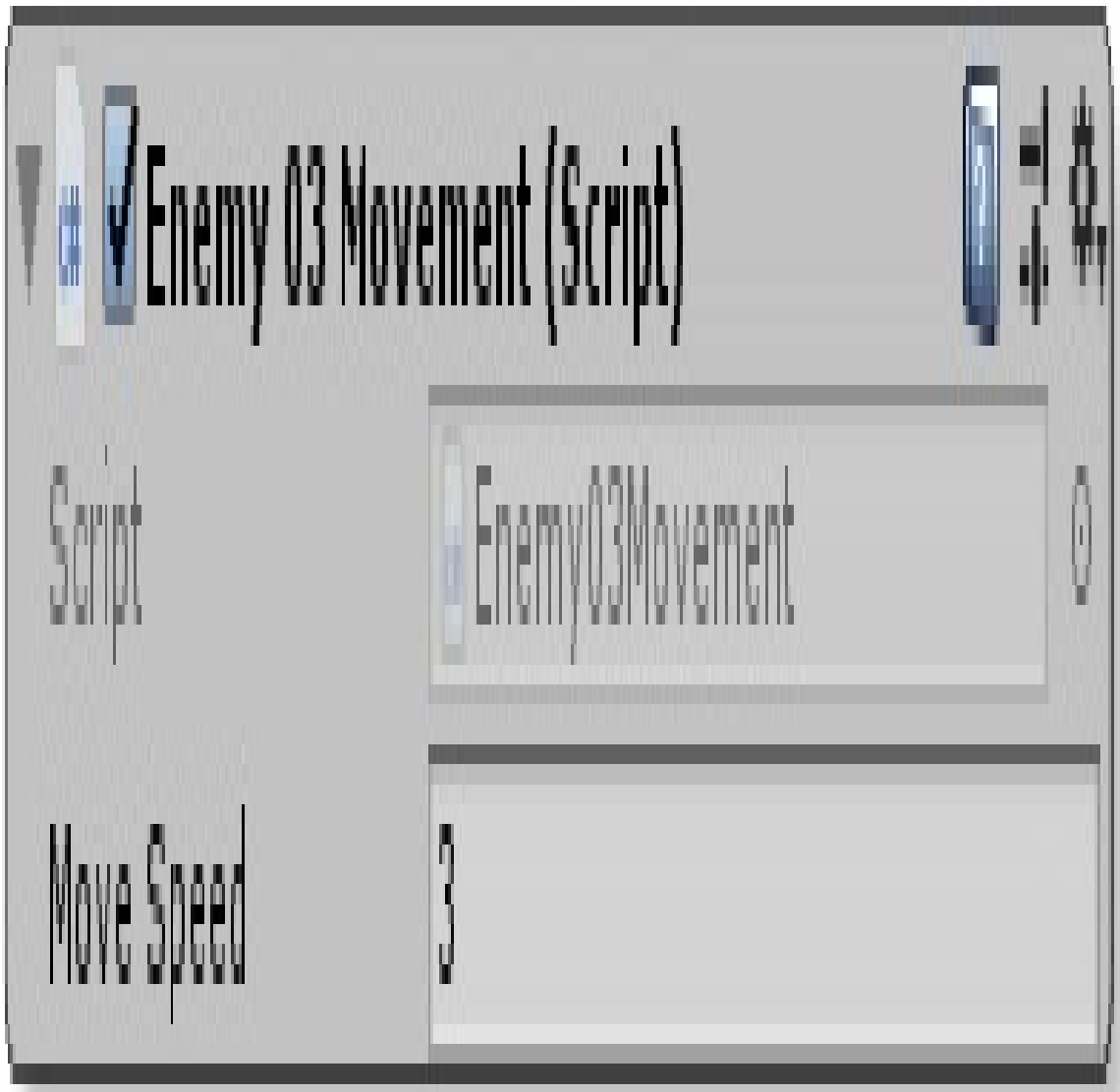
Haga un nuevo C#Script, nómbrelo “Enemy03Movement” y asígnelo a “Enemy03”.

Abra el script y escriba el siguiente código:

```
public class Enemy03Movement : MonoBehaviour {

    public float moveSpeed;
    private Rigidbody rigidBody;
    private Transform transform;

    // Use this for initialization
    void Start () {

        rigidBody = GetComponent<Rigidbody> ();
        transform = GetComponent<Transform> ();
    }

    void FixedUpdate () {

        Vector2 Vel = rigidBody.velocity;
        Vel.x = - moveSpeed;
        rigidBody.velocity = Vel;
    }
}
```

Script Enemy03Movement.

Vaya a Unity y asigne manualmente la velocidad de 3 en el componente script.


Enemy 03 Movement (Script)
Script
Enemy03Movement
Move Speed
3

Asignación manual de la velocidad.

Previsualice el juego para observar que el enemigo se mueve hacia la izquierda con su respectiva animación.

Previsualización de Enemy03.

A continuación, se programará la funcionalidad del daño al Enemy03.

Haga un nuevo Script, nómbrelo Enemy03Health y asígnelo a “Enemy03.”

Abra el script, escriba e inicialice las variables que se indican.

```
public class Enemy03Health : MonoBehaviour {

    [SerializeField] private int startingHealth = 10;
    [SerializeField] private int currentHealth;

    private Rigidbody rigidbody;
    private SphereCollider sphereCollider;
    private AudioSource audio;
    public AudioClip killAudio;
    public GameObject explosionEffect;

    // Use this for initialization
    void Start () {

        rigidbody = GetComponent <Rigidbody> ();
        sphereCollider = GetComponent <SphereCollider> ();
        currentHealth = startingHealth;
        audio = GetComponent<AudioSource> ();
    }
```

Creación e inicialización de variables.

Escriba el siguiente código para la funcionalidad del enemigo.

```
void OnTriggerEnter (Collider other) {
    if ( !GameManager.instance.GameOver){
        if (other.tag == "PlayerWeapon"){
            takeHit ();
        }
    }
}

void takeHit () {
    if (currentHealth > 0){
        GameObject newexplosionEffect =
        (GameObject)Instantiate (explosionEffect, transform.position, transform.rotation);
        Destroy (newexplosionEffect, 1);
        currentHealth -= 10;
    }

    if (currentHealth <= 0){
        KillEnemy ();
    }
}

void KillEnemy (){
    sphereCollider.enabled = false;
    audio.PlayOneShot (killAudio);
    Destroy (gameObject);

}
```

Código de script Enemy03Health

Vaya a Unity y asigne los elementos que se muestran en el componente Script.

Enemy 03 Health (Script)
Script
Enemy03Health
Starting Health
10
Current Health
0
Kill Audio
15zombie_explode
Explosion Effect
SmallExplosionEffect

Asignación manual de elementos para componente script.

Asigne un componente “AudioSource” a “Enemy03”.

Previsualice el juego para observar que el enemigo muere cuando colisiona con el proyectil del “Player”.

Previsualización de destrucción de Enemy03.

PROBLEMA: Necesitamos que el enemigo le haga daño al “Player” cuando lo toca.

SOLUCIÓN: En el componente “SphereCollider” de “Enemy03” active la opción “IsTrigger” y asigne su “Tag” a “Weapon”.

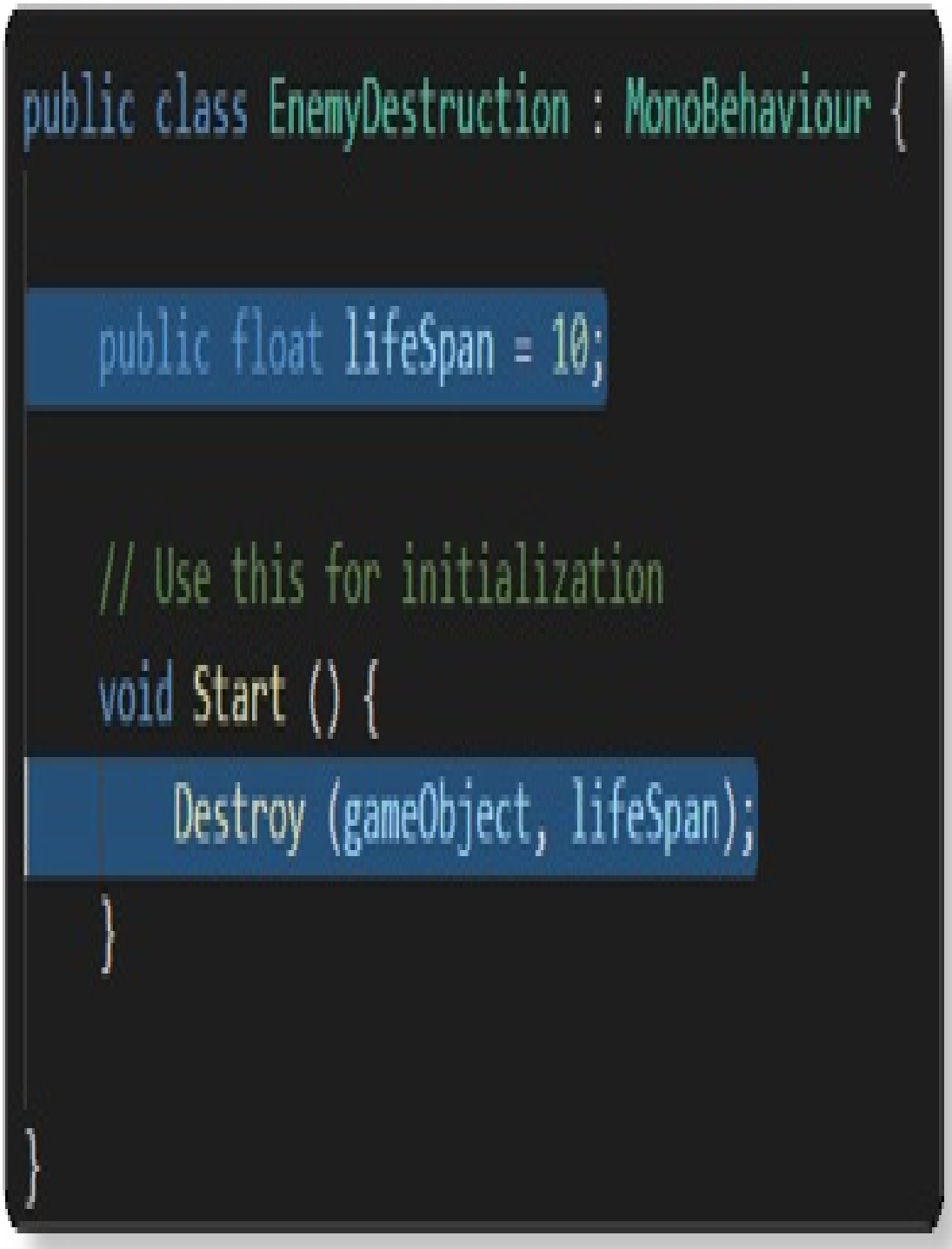
Previsualización de daño a Player.

Inspector
Enemy03
Tag
Weapon
Lay
Transform

Tag Weapon.

PROBLEMA: Ya que este enemigo va a aparecer infinitamente hasta que destruyamos “EnemyCave”, debemos de autodestruirlo después de un lapso de tiempo.

SOLUCIÓN: Haga un nuevo C#Script, nómbrelo “EnemyDestruction”, asígnelo a “Enemy03” y escriba el siguiente código:


public class EnemyDestruction : MonoBehaviour {
public float lifeSpan = 10;
// Use this for initialization
void Start () {
Destroy (gameObject, lifeSpan);
}
}

Script EnemyDestruction.

Previsualice el juego para observar que el enemigo se destruye después de 10 segundos.

Convierta “Enemy03” en Prefab y elimínelo de la escena.

Generación de enemigos

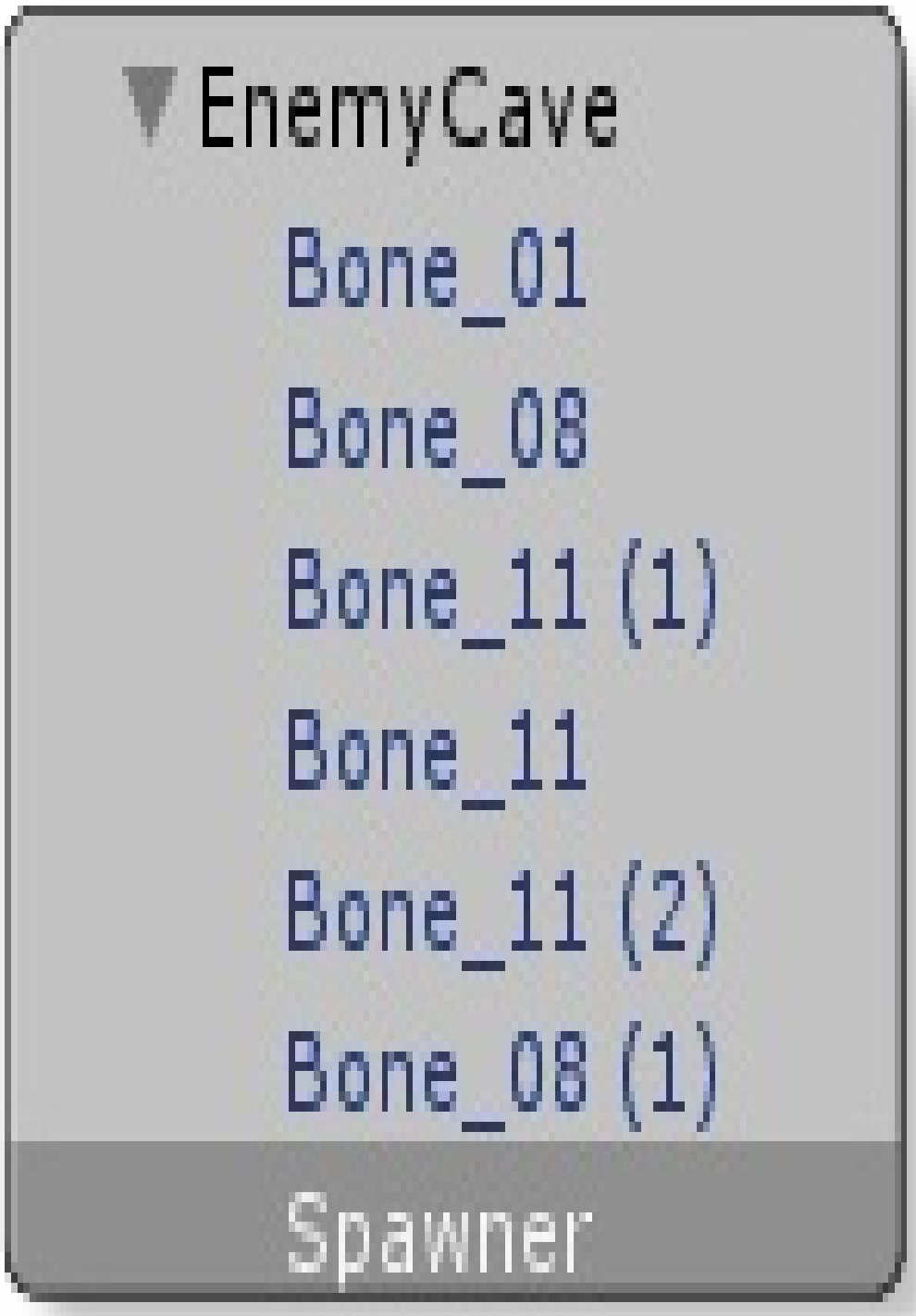
En este apartado crearemos y programaremos un objeto “Spawner” o generador de enemigos, los enemigos a generar serán los creados en el apartado anterior y programaremos su funcionalidad de generación infinita hasta que el enemigo tipo cueva sea destruido.

Haga un nuevo “EmptyObject”, nómbrelo “Spawner”, asigne un “label” color azul y colóquelo donde se muestra.

Spawner

Colocación de objeto Spawner.

Haga el objeto “Spawner” hijo del objeto “EnemyCave”.


EnemyCave
Bone_01
Bone_08
Bone_11 (1)
Bone_11
Bone_11 (2)
Bone_08 (1)
Spawner

Emparentamiento de Spawner.

Haga un nuevo C#Script, nómbrelo “EnemySpawner” y asígnelo al objeto “Spawner”.

Abra el script, haga e inicialice las variables que se muestran.

```
    [SerializeField] private float range = 15f;
    [SerializeField] private float timeBetweenSpawn = 1f;

    private GameObject player;
    private bool playerInRange;

    public Transform enemySpawn;
    public Rigidbody enemyPrefab;

    Rigidbody clone;

    // Use this for initialization
    void Start () {

        enemySpawn = GameObject.Find ("Spawner").transform;
        player = GameManager.instance.Player;
    }
```

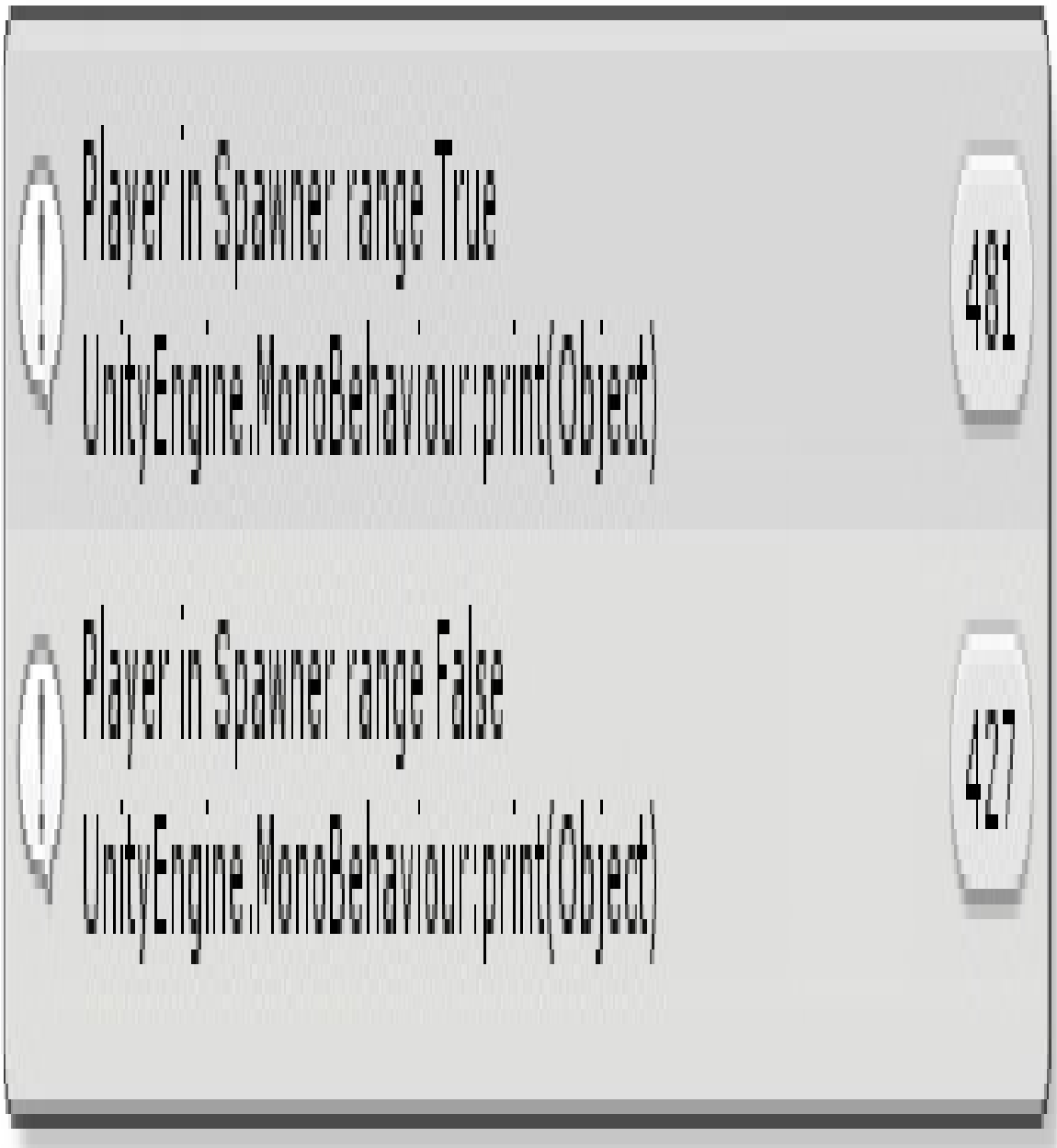
Creación e inicialización de variables.

Escriba el siguiente código en la función “Update” para comprobar si el “Player” se encuentra en rango.

```
void Update () {

    if (Vector3.Distance (transform.position, player.transform.position) < range) {
        playerInRange = true;
    } else {
        playerInRange = false;
    }
    print ("Player in Spawner range " + playerInRange);

}
```

Comprobación de rango.

Previsualice el juego para observar en la consola si nuestro “Player” se encuentra en rango del “Spawner” a medida que se acerca o se aleja de él.


Player in Spawner range True
UnityEngine.MonoBehaviour:print(Object)
481
Player in Spawner range False
UnityEngine.MonoBehaviour:print(Object)
427

Previsualización de comprobación de rango.

Escriba el siguiente código en el script “EnemySpawner” para crear la función de la generación de enemigos.

```
        print ("Player in Spawner range " + playerInRange);
    }


    public IEnumerator SpawnEnemies () {

        if (playerInRange && !GameManager.instance.GameOver) {

            clone = Instantiate (enemyPrefab, enemySpawn.position, enemySpawn.rotation)
            as Rigidbody;
            yield return new WaitForSeconds (timeBetweenSpawn);
        }

        yield return null;
        StartCoroutine (SpawnEnemies ());
    }
}
```

Función IEnumerator.

Mande llamar la “Coroutine” en la función “Start”.

```
void Start () {

    enemySpawn = GameObject.Find ("Spawner").transform;
    player = GameManager.instance.Player;

    StartCoroutine (SpawnEnemies ());
}
```

Llamar la “Coroutine”

Vaya a Unity, seleccione el objeto “Spawner” y asigne en Inspector los elementos que se indican.

Enemy Spawner (Script)
Script
enemySpawner
Range
10
Time Between Spawn
3
Enemy Spawn
Spawner (Transform)
Enemy Prefab
Enemy03 (Rigidbody)

Asignación de elementos.

Seleccione el objeto “EnemyCave” y conviértalo en Prefab.

Previsualice el juego para observar el funcionamiento del “Spawner”.

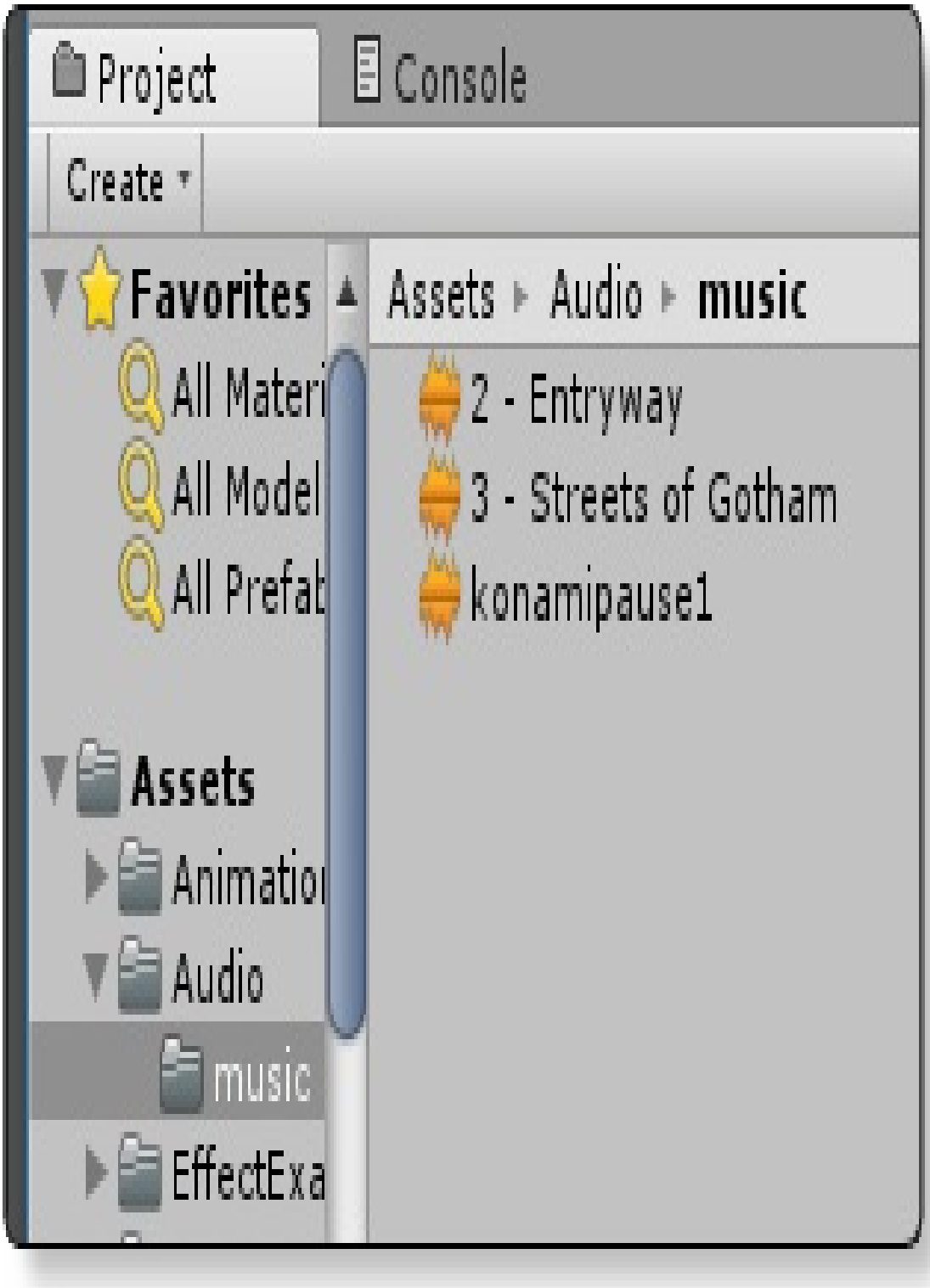
X3

Previsualización de funcionalidad de Spawner.

Añadir música al juego

En este apartado añadiremos componentes “AudioSource” a los estados de juego para añadirles música.

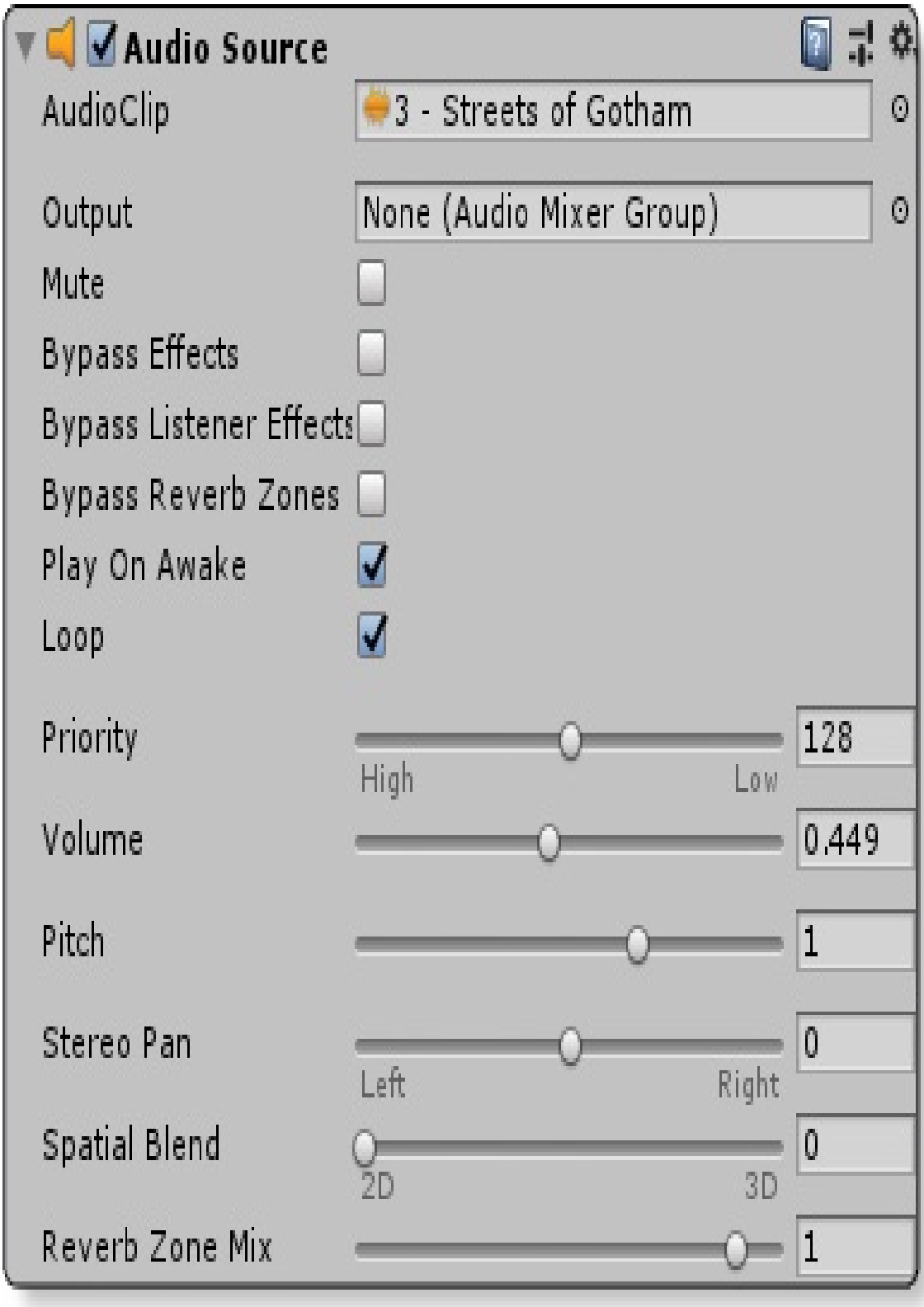
De la carpeta de “resources” del capítulo importe al proyecto los archivos mp3 que se muestran.


Project
Console
Create
Favorites
All Materi
All Model
All Prefat
Assets
Animatio
Audio
music
EffectExa
Assets ▸ Audio ▸ music
2 - Entryway
3 - Streets of Gotham
konamipause1

Importación de archivos de audio.

En “Hierarchy” haga un nuevo “EmptyObject” y renómbrelo como “AudioManager”.

Asigne un componente “AudioSource” y asigne el audio mp3, activando la casilla “Loop”.


Audio Source
AudioClip
3 - Streets of Gotham
Output
None (Audio Mixer Group)
Mute
Bypass Effects
Bypass Listener Effects
Bypass Reverb Zones
Play On Awake
Loop
Priority
128
High
Low
Volume
0.449
Pitch
1
Stereo Pan
0
Left
Right
Spatial Blend
0
2D
3D
Reverb Zone Mix
1

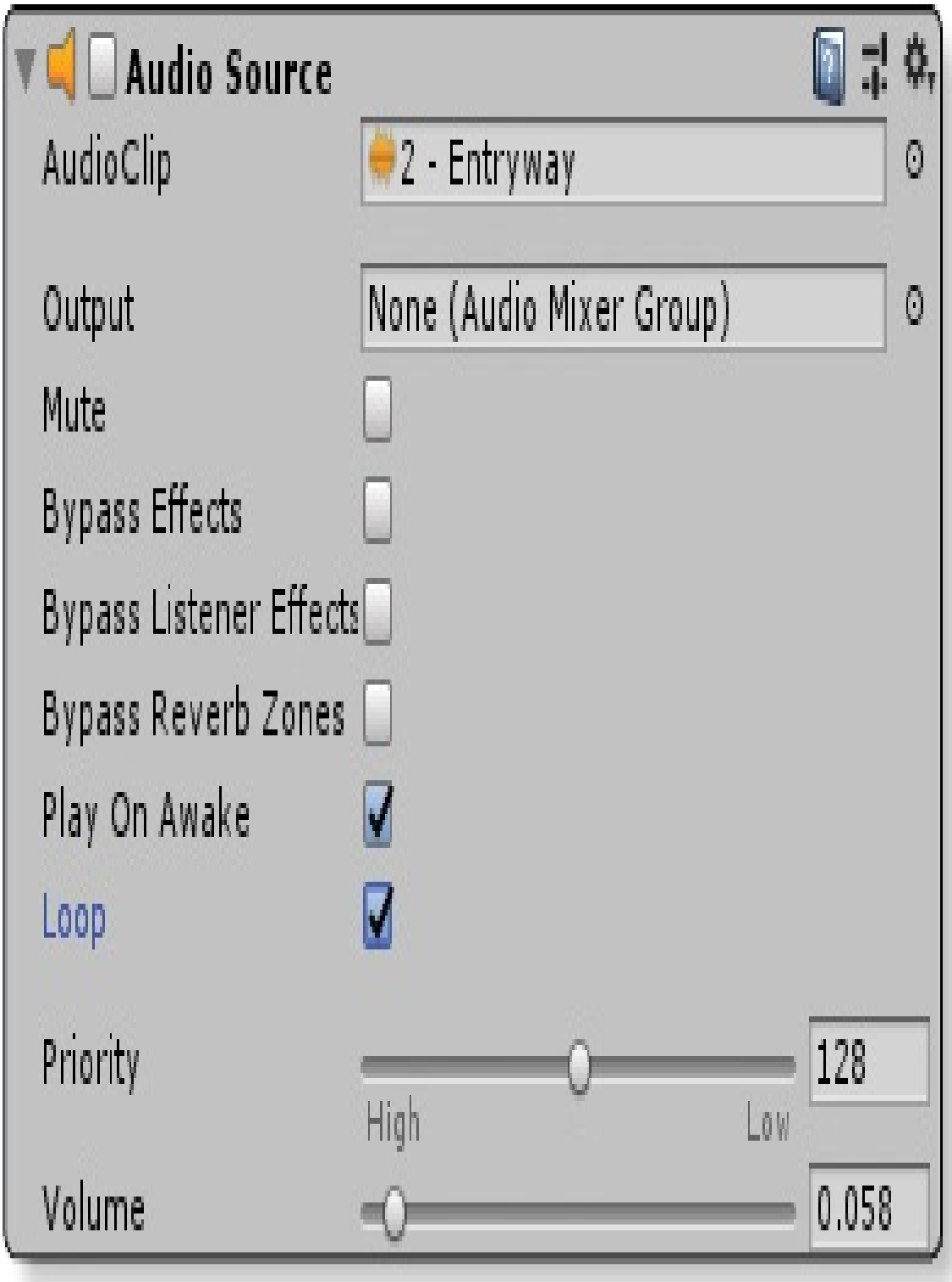
Componente AudioSource.

Previsualice el juego para escuchar la música del nivel.

Convierta el objeto “AudioManager” en Prefab.

Abra la escena de inicio (la que tiene los botones de Start y Quit).

Añada a “Hierarchy” el Prefab “AudioManager” creado y cambie el clip de audio como se indica.


Audio Source
AudioClip
2 - Entryway
Output
None (Audio Mixer Group)
Mute
Bypass Effects
Bypass Listener Effects
Bypass Reverb Zones
Play On Awake
Loop
Priority
128
High
Low
Volume
0.058

Asignación de clip de audio.

Previsualice el juego para escuchar la música de la pantalla inicial.

Programación de la funcionalidad de Pausa

En este apartado se programará la funcionalidad de pausa conociendo comandos de la librería standard para reproducir y detener el juego.

También se abarcará la forma en la que podemos manipular la escala de tiempo de nuestro videojuego, pausaremos la música y añadiremos un clip de audio a la funcionalidad de pausa.

Vaya a “Hierarchy”, haga un nuevo “EmptyObject” y renómbrelo “Pause”.

Haga un nuevo C#Script, nómbrelo “PauseScript” y asígnelo al objeto “Pause”.

Abra el script creado y escriba el siguiente código para pausar el juego al oprimir la tecla “P” del teclado.

```
public class PauseScript : MonoBehaviour {

    public static bool GameIsPaused = false;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

        if (Input.GetKeyDown (KeyCode.P)) {

            if (GameIsPaused) {
                Resume ();
            } else {
                Paused ();
            }
        }
    }

    void Resume () {

        Time.timeScale = 1f;
        GameIsPaused = false;
    }

    void Paused () {

        Time.timeScale = 0f;
        GameIsPaused = true;
    }
}
```

PauseScript.

Podemos observar que:

Los comandos “Resume” y “Paused” forman parte de la librería que estamos utilizando y que ya contienen las funcionalidades preprogramadas.

La escala de tiempo “timeScale” obedece a

1 = La reproducción normal.

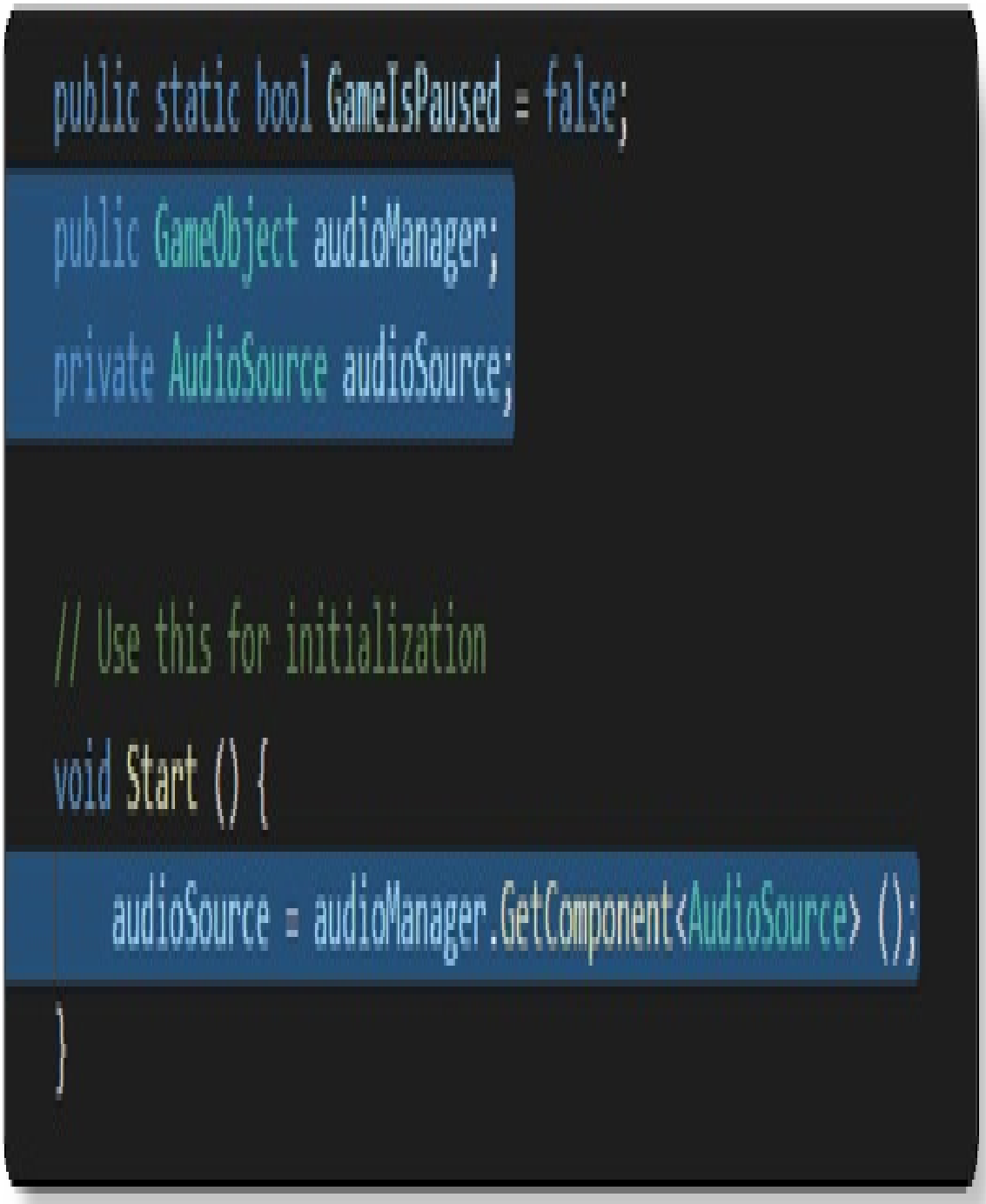
2 = Reproducción detenida

NOTA:

Si quisiéramos asignar el doble de velocidad bastaría con asignar la escala de tiempo a 2.

PROBLEMA: Al hacer pausa la música sigue corriendo y necesitamos detenerla.

SOLUCIÓN: Haga las variables indicadas e inicialícelas.


public static bool GameIsPaused = false;
public GameObject audioManager;
private AudioSource audioSource;
// Use this for initialization
void Start () {
audioSource = audioManager.GetComponent<AudioSource> ();
}

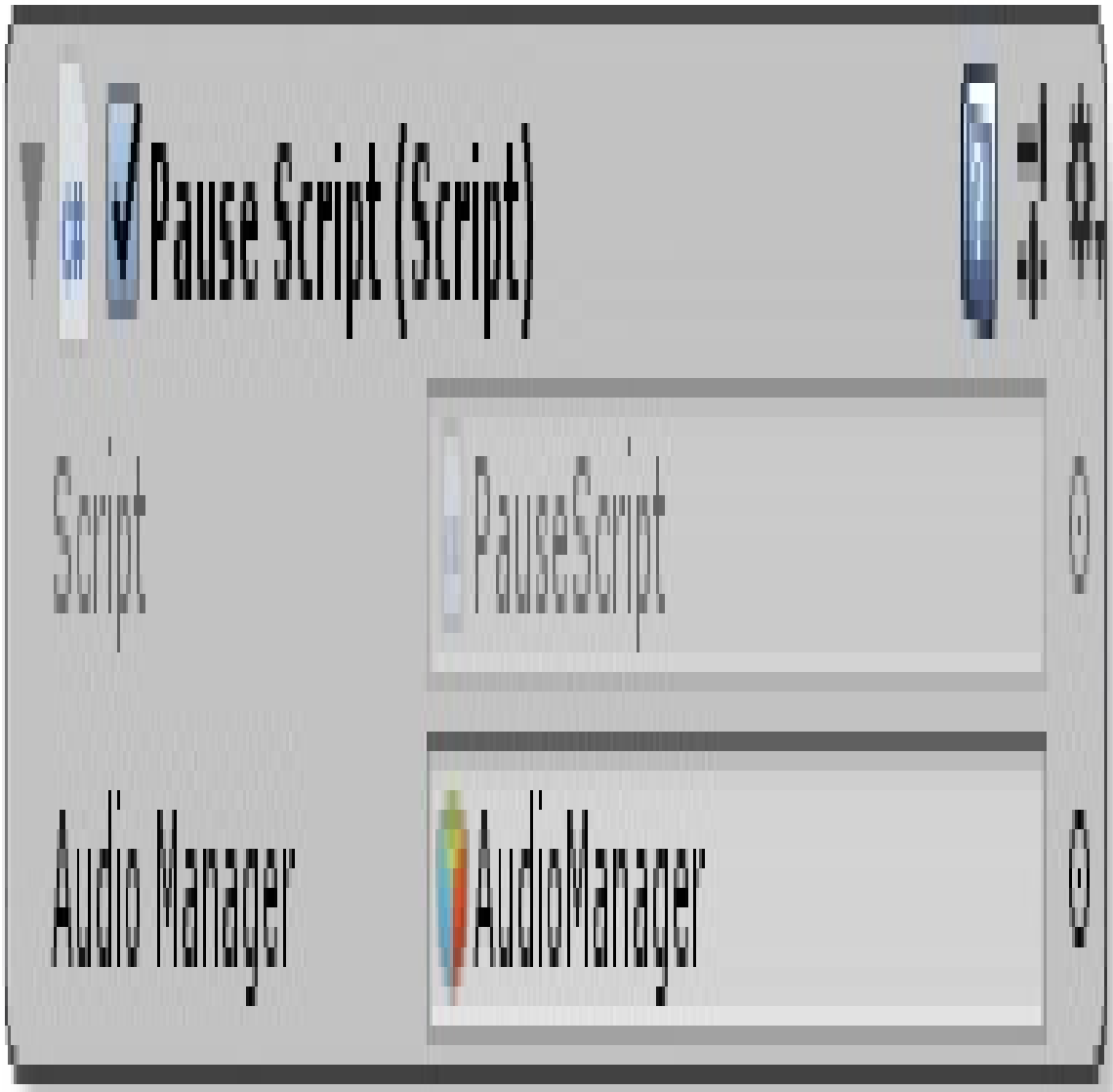
Creación e inicialización de variables.

Asigne los siguientes códigos en las funciones mostradas para pausar y reproducir la música.

```
    void Resume () {

        Time.timeScale = 1f;
        GameIsPaused = false;
        audioSource.Play ();
    }

    void Paused () {

        Time.timeScale = 0f;
        GameIsPaused = true;
        audioSource.Pause ();
    }
```

Comandos para reproducir y pausar.

Vaya a Unity y asigne el objeto “AudioManager” en el script.


Pause Script (Script)
Script
PauseScript
Audio Manager
AudioManager

Asignación de elemento AudioManager.

Asigne un componente “AudioSource” al objeto “Pause.

Previsualice el juego para escuchar la pausa en el audio también.

A continuación, necesitamos añadir un audio a la pausa.

Vaya al script “PauseScript”, haga e inicialice las siguientes variables.

```
private AudioSource audioSource;


private AudioSource pauseAudio;
public AudioClip pause;


// Use this for initialization
void Start () {
    audioSource = audioManager.GetComponent<AudioSource> ();
    pauseAudio = GetComponent<AudioSource> ();
}
```

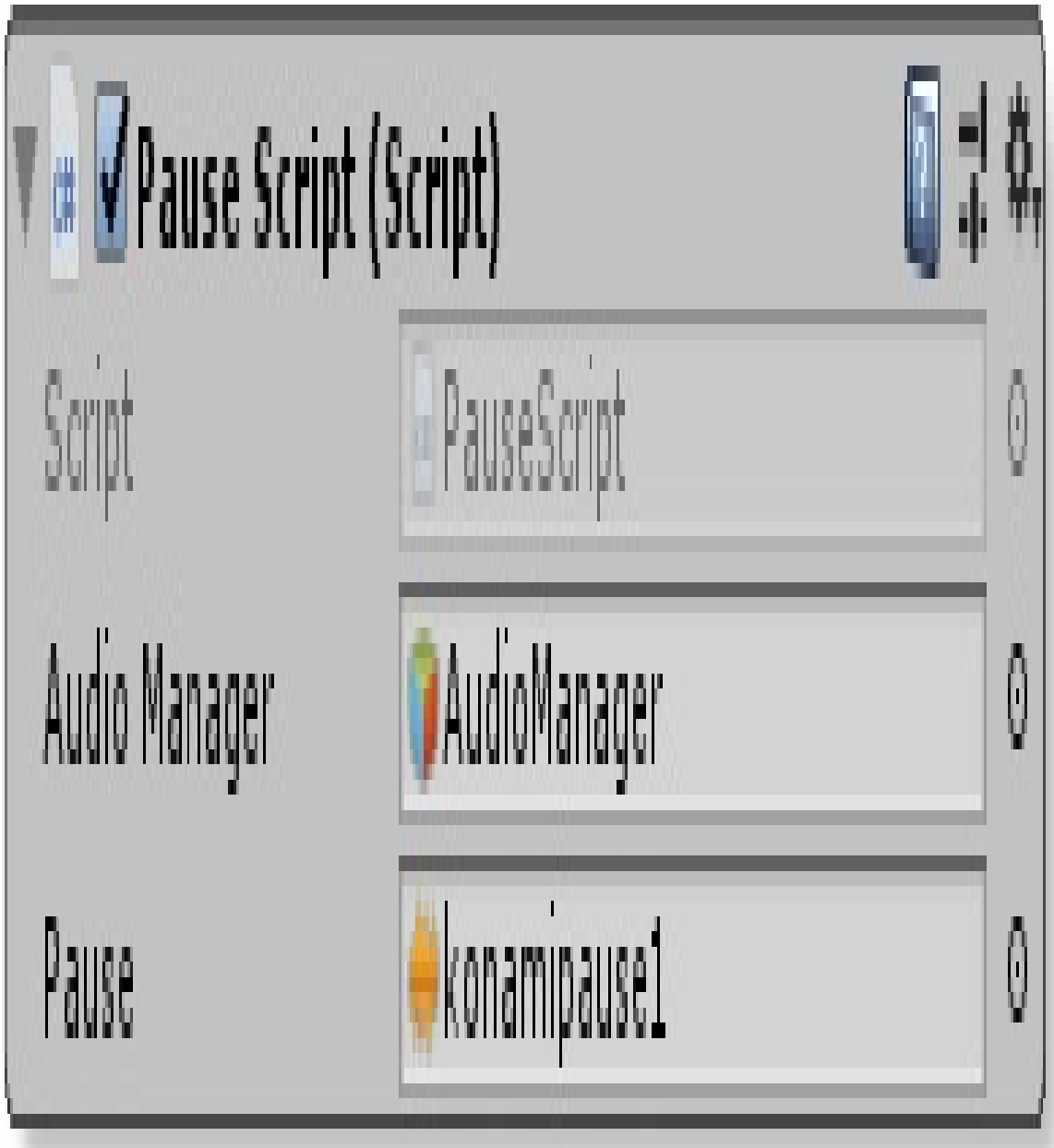
Creación e inicialización de variables.

Escriba los siguientes códigos en las funciones que se indican.

```
void Resume () {
    Time.timeScale = 1f;
    GameIsPaused = false;
    audioSource.Play ();
    pauseAudio.PlayOneShot (pause);
}

void Paused () {
    Time.timeScale = 0f;
    GameIsPaused = true;
    audioSource.Pause ();
    pauseAudio.PlayOneShot (pause);
}
```

Código para llamar el audio de la pausa.

Vaya a Unity y asigne el audio que se muestra en el script.


Pause Script (Script)
Script
PauseScript
Audio Manager
AudioManager
Pause
konamipause1

Asignación de clip de audio.

Previsualice el juego para escuchar el sonido de la pausa.

8

BATALLA CON EL JEFE FINAL

INTRODUCCIÓN

En este capítulo programaremos la funcionalidad completa de la batalla con el jefe final ocupándonos de los siguientes aspectos.

Importación y preparación del jefe

Programación de un “BossCheckPoint

Cambio de cámara para la batalla final

Crear y programar las animaciones del jefe

Comportamientos del “Player”

Asignación de tres diferentes tipos de ataques del jefe y programarlos aleatoriamente

Creación de un sistema de salud para el jefe y su barra de vida GUI

Trabajo con partículas para crear un ataque de rocas cayendo

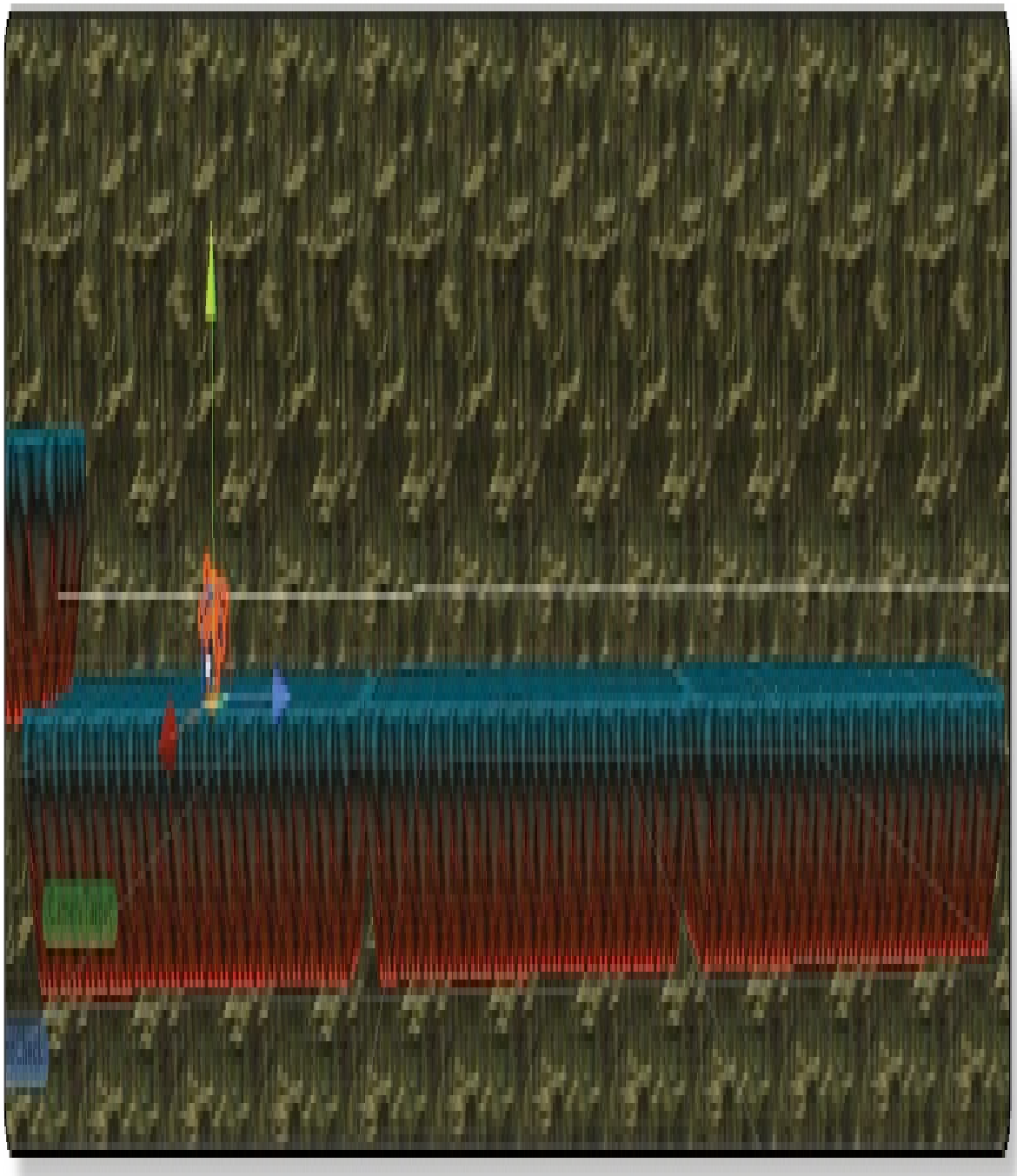
Programación de funcionalidad de daño y muerte del jefe

Detalles finales para completar un trabajo profesional con la batalla final

Preparación del Jefe Final

En este apartado importaremos a nuestro jefe final, así como preparar sus componentes funcionales para poder comenzar a programar sus ataques y comportamientos.

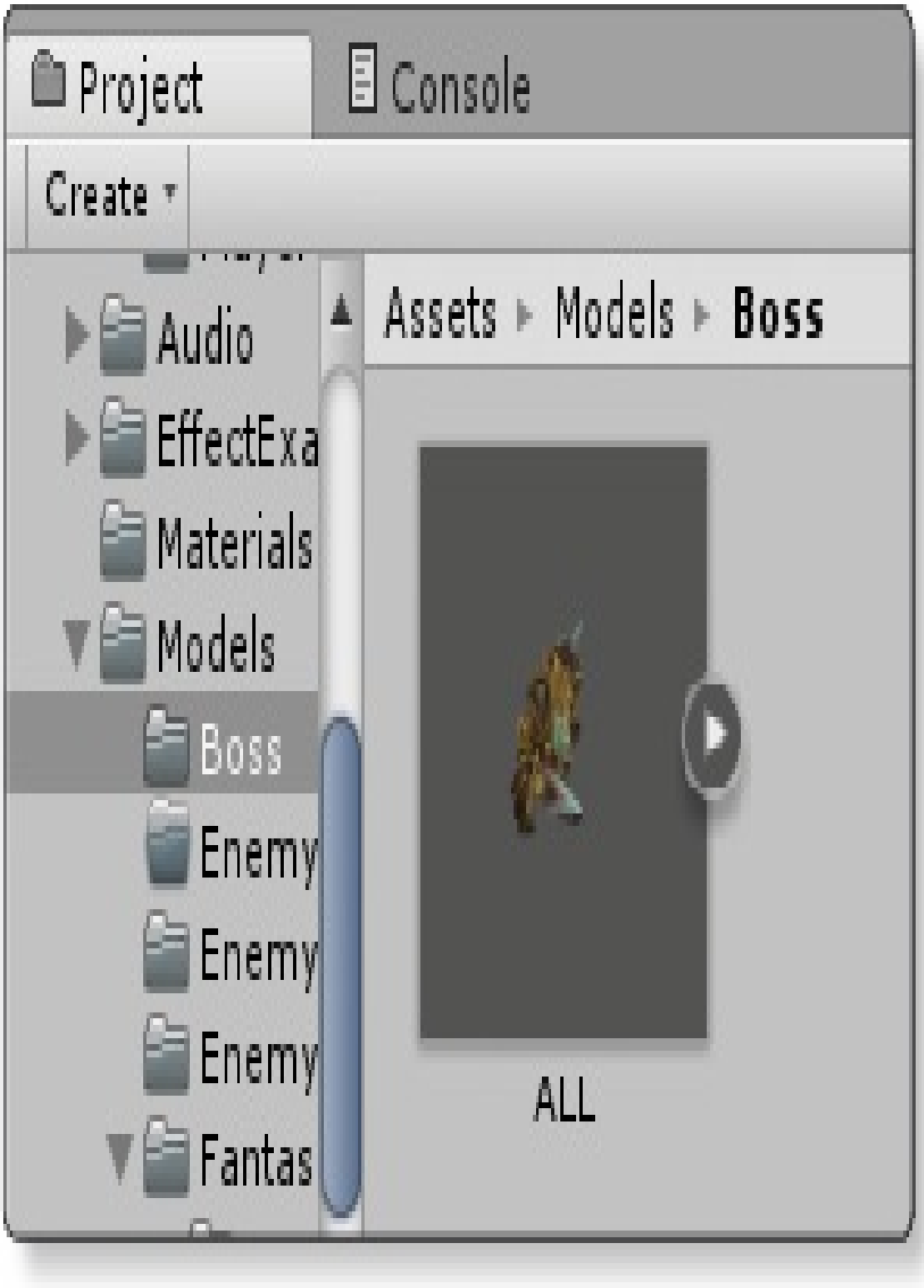
Expanda el nivel, colocando tres plataformas y coloque al “Player” al inicio de ellas.



Expansión de nivel

En la ventana “Project” vaya al folder “Models”, haga una nueva carpeta y nómbrela “Boss”.

De la carpeta externa de “resources” localice el folder “skeleton_king / animations” y con los conceptos aprendidos importe, texturice, escale y coloque el modelo “ALL.fbx” en la escena.


Project
Console
Create
Audio
EffectExa
Materials
Models
Boss
Enemy
Enemy
Enemy
Fantas
Assets ▸ Models ▸ Boss
ALL

Importación de modelo

KnifeSpawn

Escala y colocación del modelo

Inspector
Skeleton_King
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y -105.52
Z 0
Scale
X 3
Y 3
Z 3

Renombrar y escalar modelo

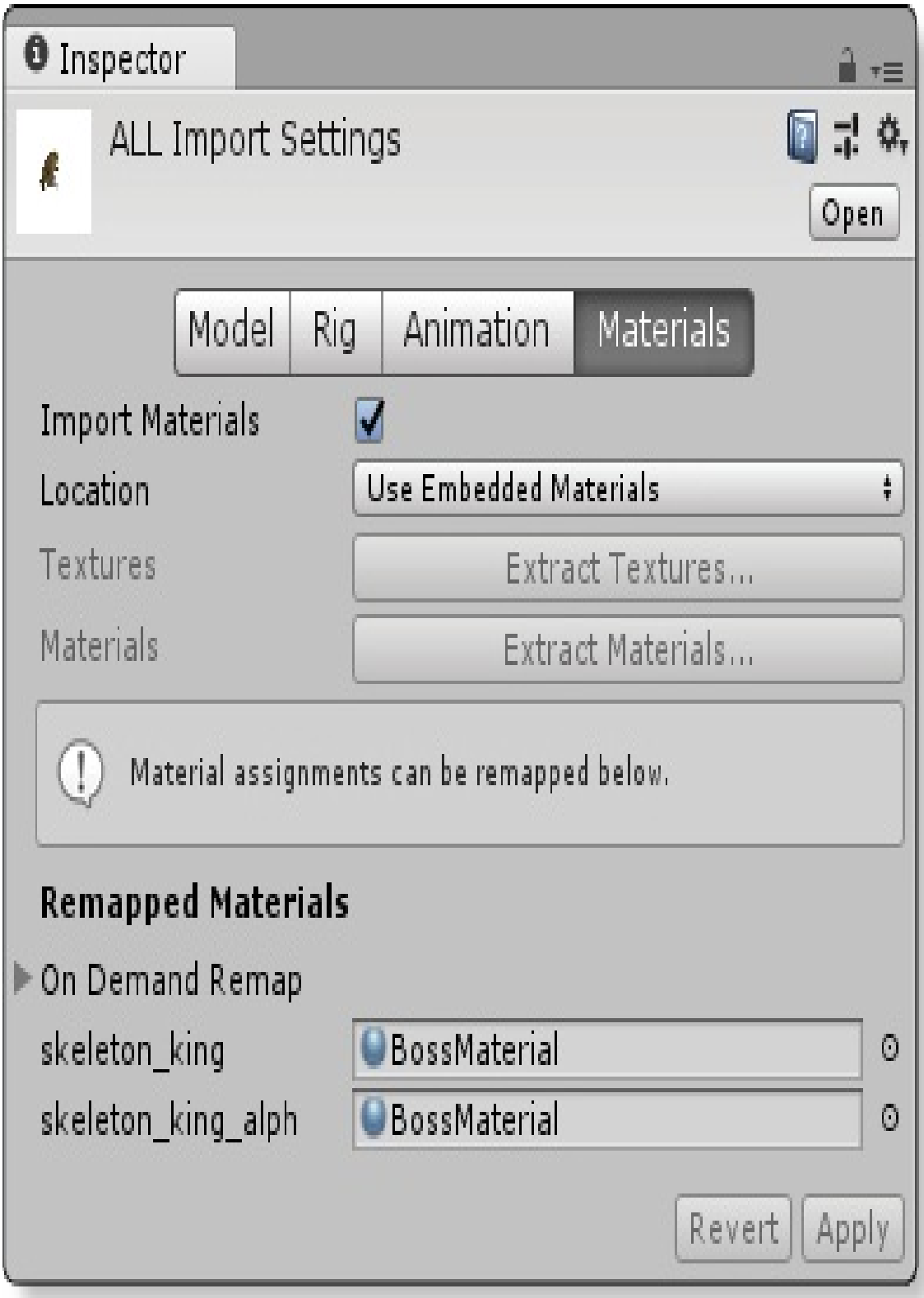
Renombre el objeto en Inspector como “Skeleton_King.

Podemos observar que el archivo “All.fbx” de la carpeta “Animations” de resources contiene también el modelo junto con la animación de todos los clips a separar juntos.

Debemos de crear su material para texturizar y asignarlo directamente en Inspector.

Project
Console
Create
Assets
Animatio
BossAr
Enemy
Enemy
Enemy
Enemy
Player
Audio
EffectExa
Materials
Assets ▸ Materials
BossMaterial

Creación de material para el modelo


Inspector
ALL Import Settings
Open
Model
Rig
Animation
Materials
Import Materials
Location
Use Embedded Materials
Textures
Extract Textures...
Materials
Extract Materials...
Material assignments can be remapped below.
Remapped Materials
On Demand Remap
skeleton_king
BossMaterial
skeleton_king_alph
BossMaterial
Revert
Apply

Asignación de material para el modelo

NOTA:

El modelo se selecciona directamente desde la ventana “Project”, no desde “Hierarchy” para que aparezcan las opciones mostradas.

Este modelo contiene dos nodos; el difuso y el alpha, en el alpha podemos colocar archivos PNG con transparencia para que algunas partes de nuestro modelo se vean transparentes, pero en este caso asignaremos el mismo material para ambos nodos.

Haga un nuevo “EmptyObject”, renómbrelo “Boss” y hágalo padre del objeto “Skeleton_King”.

Haga un nuevo “Tag”, nómbrelo “Boss” y asigne el tag al objeto “Boss”.

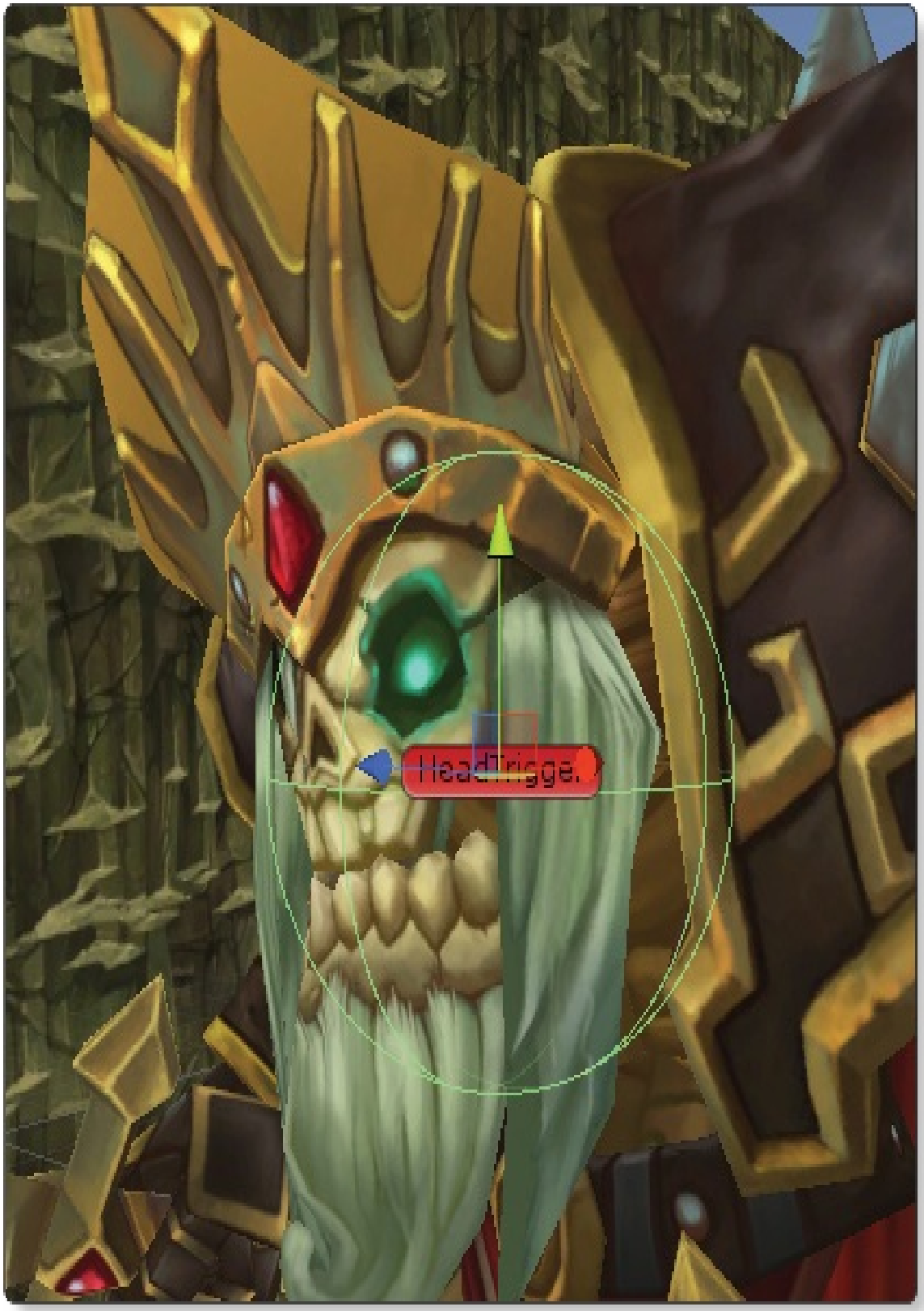
A continuación, debemos de crear un nuevo objeto que nos sirva como “Trigger” en la cabeza del jefe para poderle hacer daño cuando los proyectiles del “Player” colisionen con su cabeza.

Haga un nuevo “EmptyObject”, renómbrelo como “HeadTrigger”, asigne un “label” color rojo y emparéjelo con el objeto “Boss”.

Boss
Skeleton_King
HeadTrigger

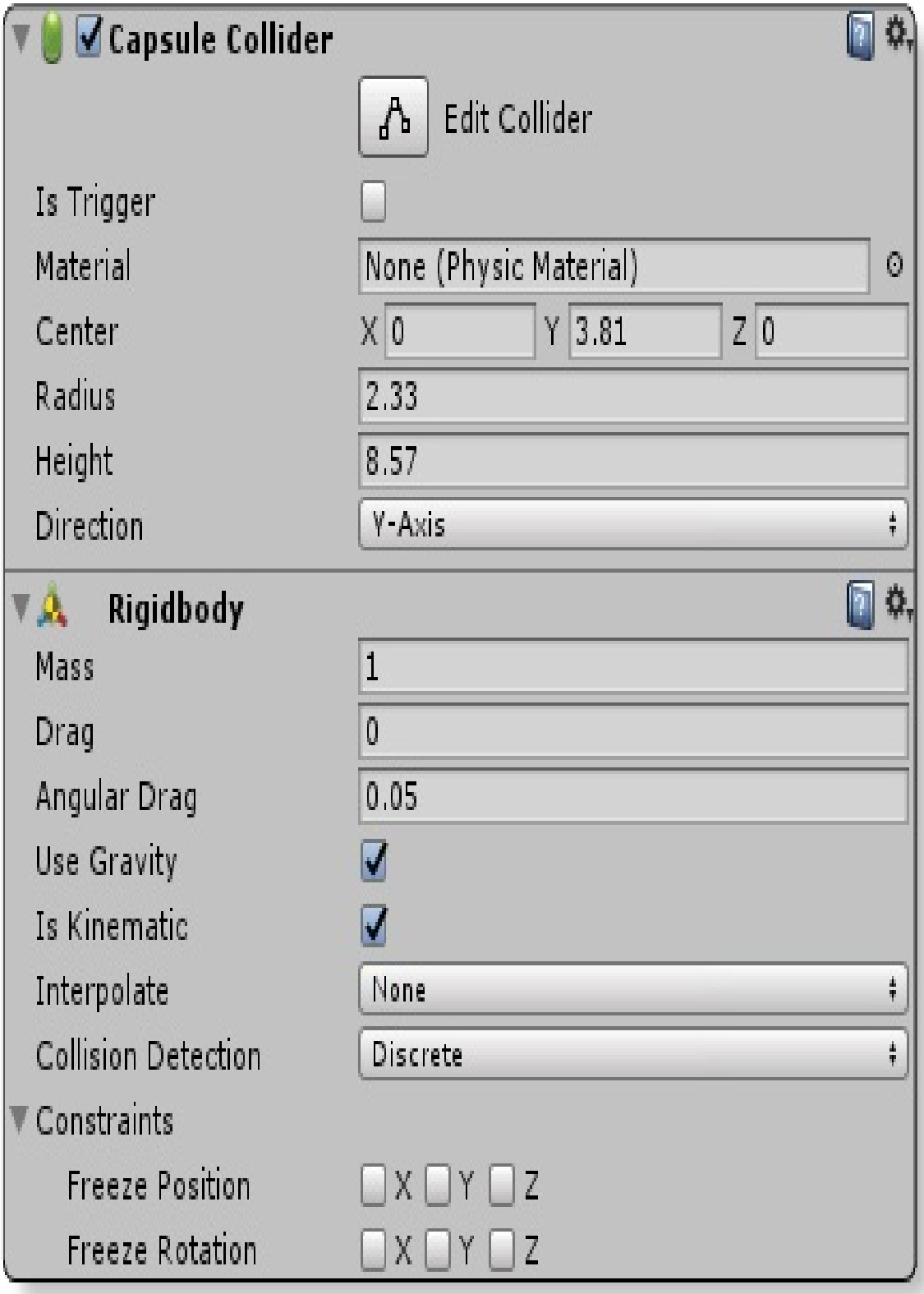
Emparentamiento de “HeadTrigger”

Asigne al objeto “HeadTrigger” un “SphereCollider”, asigne los valores necesarios para que se ajuste al tamaño de la cabeza y active su casilla “IsTrigger”.


HeadTrigge

Colocación del HeadTrigger

Seleccione el objeto “Boss” y asigne sus componentes “CapsuleCollider” y RigidBody” con los valores mostrados.


Capsule Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 3.81
Z 0
Radius
2.33
Height
8.57
Direction
Y-Axis
Rigidbody
Mass
1
Drag
0
Angular Drag
0.05
Use Gravity
Is Kinematic
Interpolate
None
Collision Detection
Discrete
Constraints
Freeze Position
X Y Z
Freeze Rotation
X Y Z

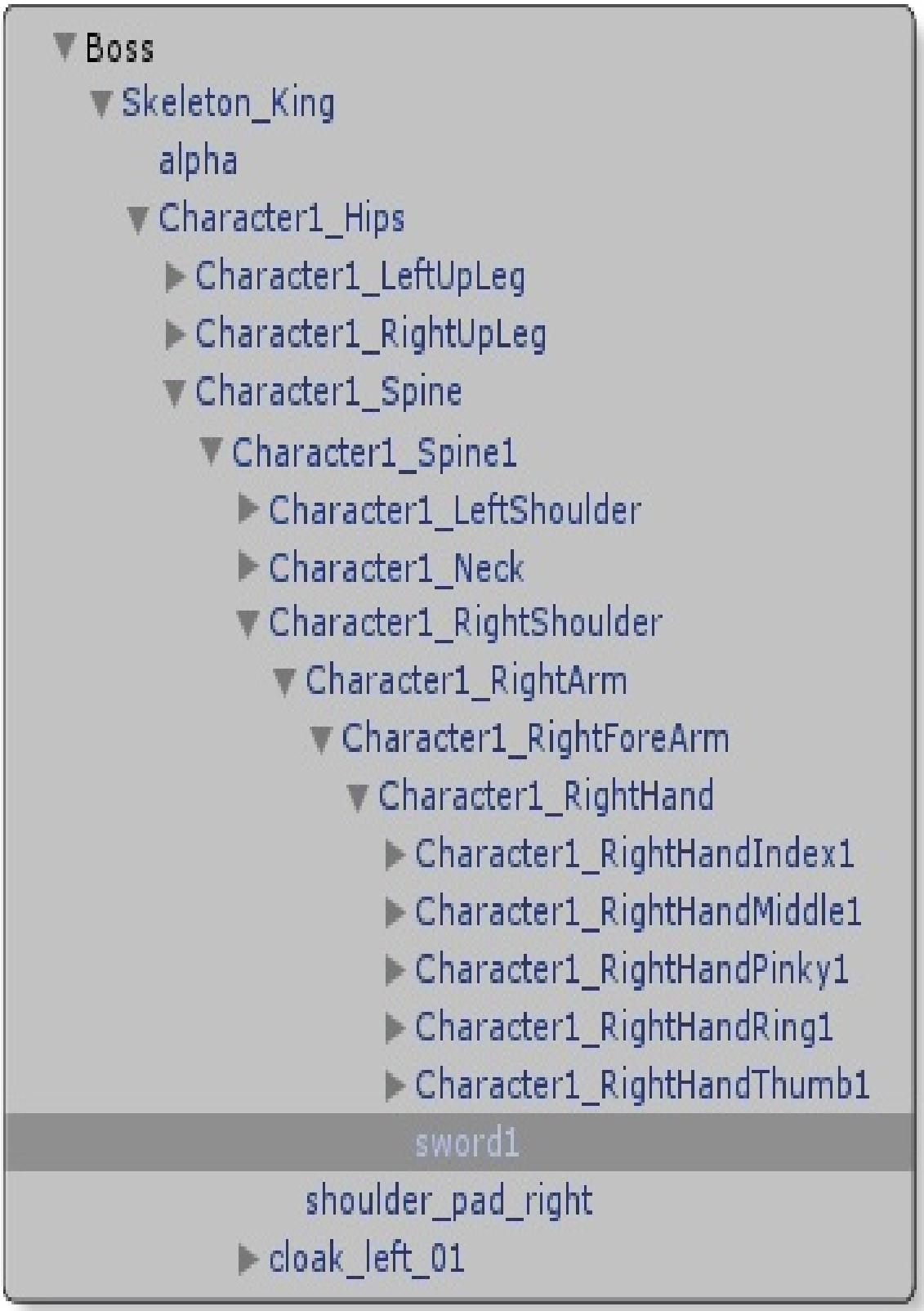
Componentes para Boss

HeadTrigger

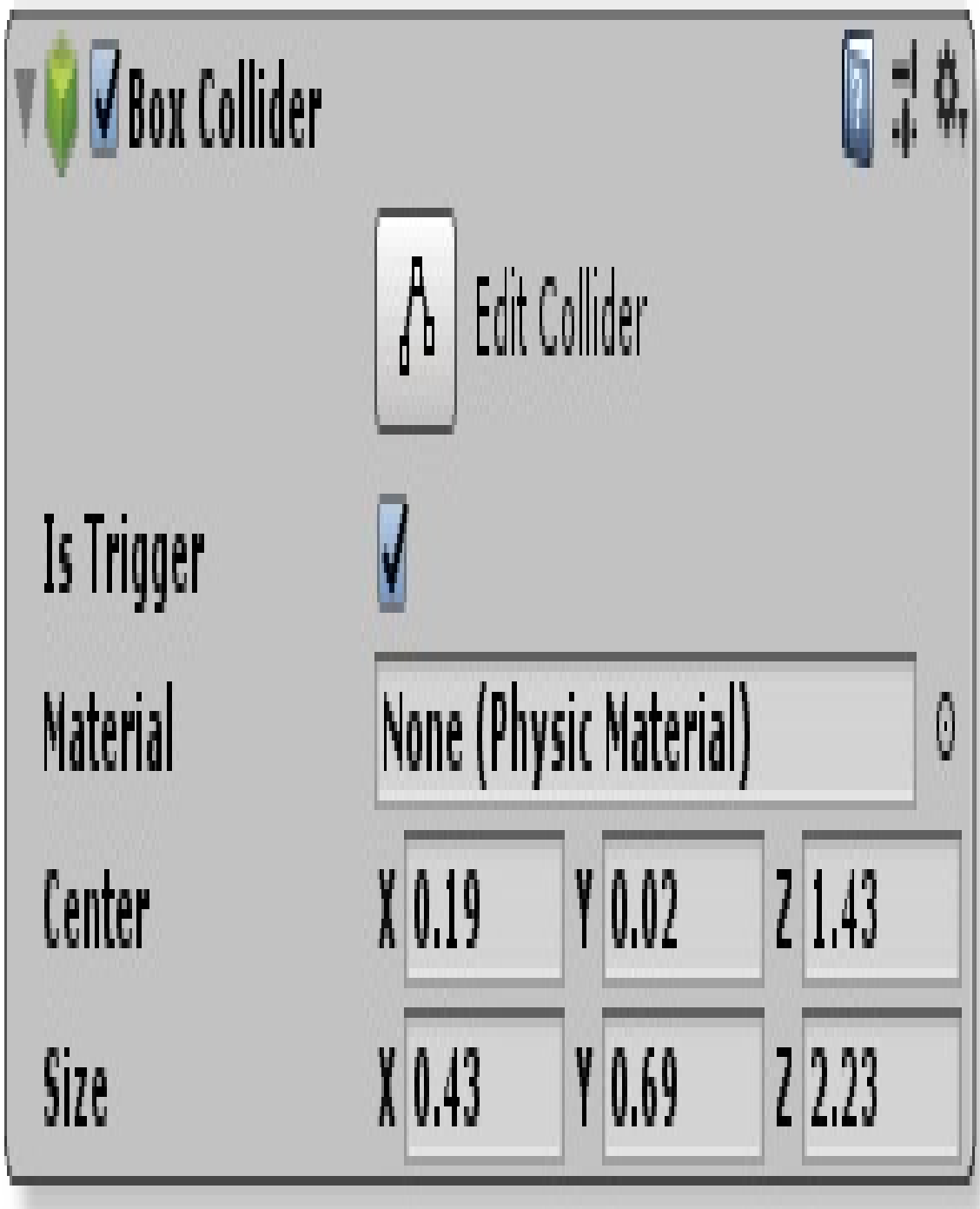
CapsuleCollider de Boss

A continuación, debemos de asignar un “BoxCollider” al arma de nuestro jefe.

Expanda la jerarquía del modelo hasta encontrar el elemento “Sword” y asigne un “BoxCollider” con los valores mostrados.


Boss
Skeleton_King
alpha
Character1_Hips
Character1_LeftUpLeg
Character1_RightUpLeg
Character1_Spine
Character1_Spine1
Character1_LeftShoulder
Character1_Neck
Character1_RightShoulder
Character1_RightArm
Character1_RightForeArm
Character1_RightHand
Character1_RightHandIndex1
Character1_RightHandMiddle1
Character1_RightHandPinky1
Character1_RightHandRing1
Character1_RightHandThumb1
sword1
shoulder_pad_right
cloak_left_01

Localización de joint “Sword”


Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0.19
Y 0.02
Z 1.43
Size
X 0.43
Y 0.69
Z 2.23

Valores de BoxCollider de Sword

HeadTrigger

BoxCollider de Sword correcto

Elimine el componente “Animator” del objeto “Skeleton_King.”

Añada un componente “Animator” al objeto “Boss”.

Con los conceptos aprendidos:

Haga el “AnimatorControl” para el Boss y nómbrelo “BossAC”

Haga la animación de “Idle” para el Boss y nómbrela “BossIdle”

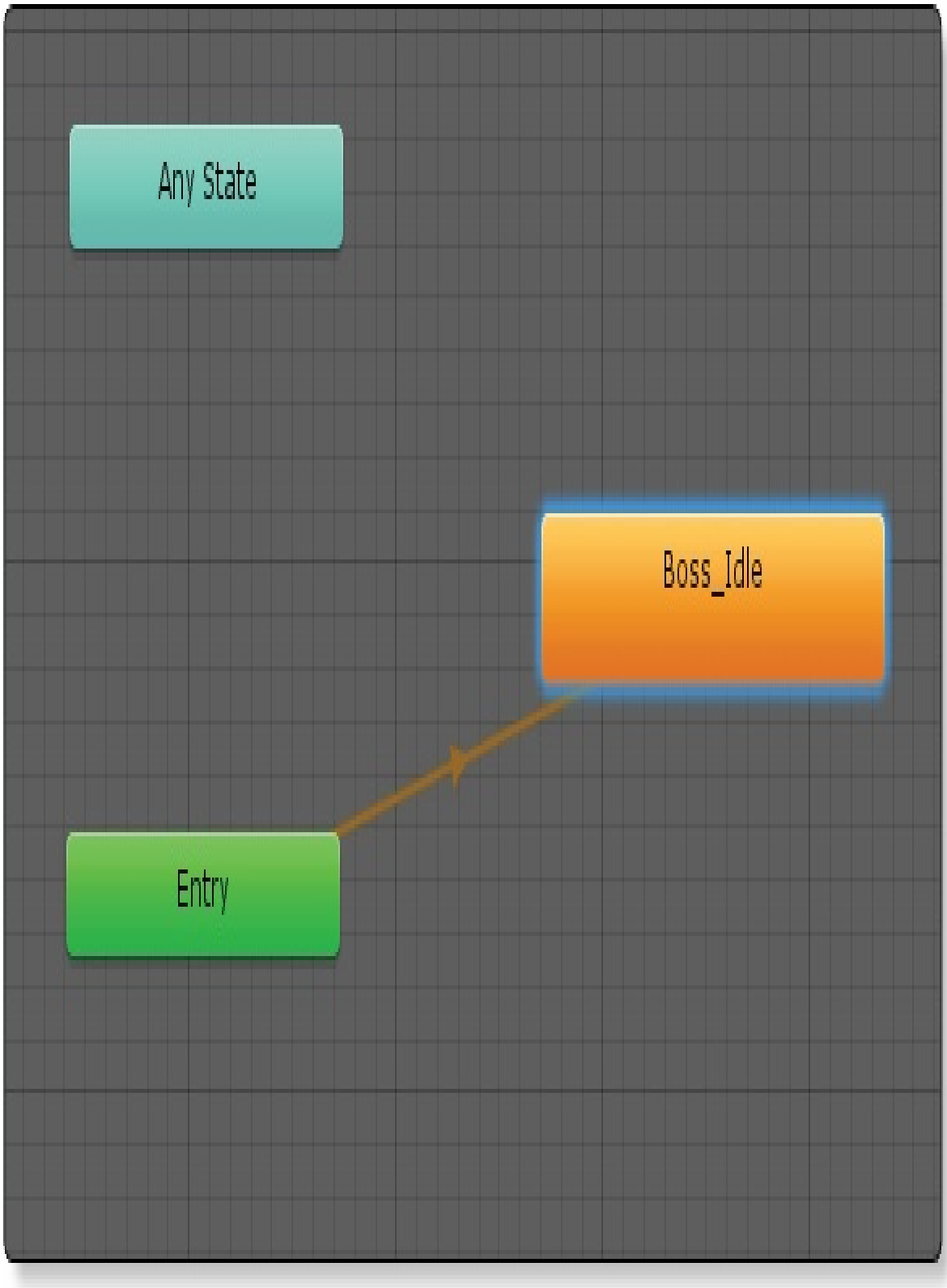
frame: 0060-0119: idle (loop)

Asigne los elementos que se muestran en “Animator”

Animator
Controller
BossAC
Avatar
ALLAvatar
Apply Root Motion
Update Mode
Normal
Culling Mode
Always Animate

Componente Animator de Boss

En la ventana “Animator” asigne la animación “BossIdle” como estado default


Any State
Boss_Idle
Entry

Asignación de animación Boss_Idle

Previsualice el juego para observar la animación de Idle del jefe.

Programación de un BossCheckpoint

En este apartado crearemos un “CheckPoint” relativo a la entrada con la pelea con el jefe final y lo programaremos para que cuando nuestro Player pase a través de él, ya no pueda regresar.

Haga un nuevo “EmptyObject” con los siguientes elementos:

Renómbrelo “BossCheckPoint”

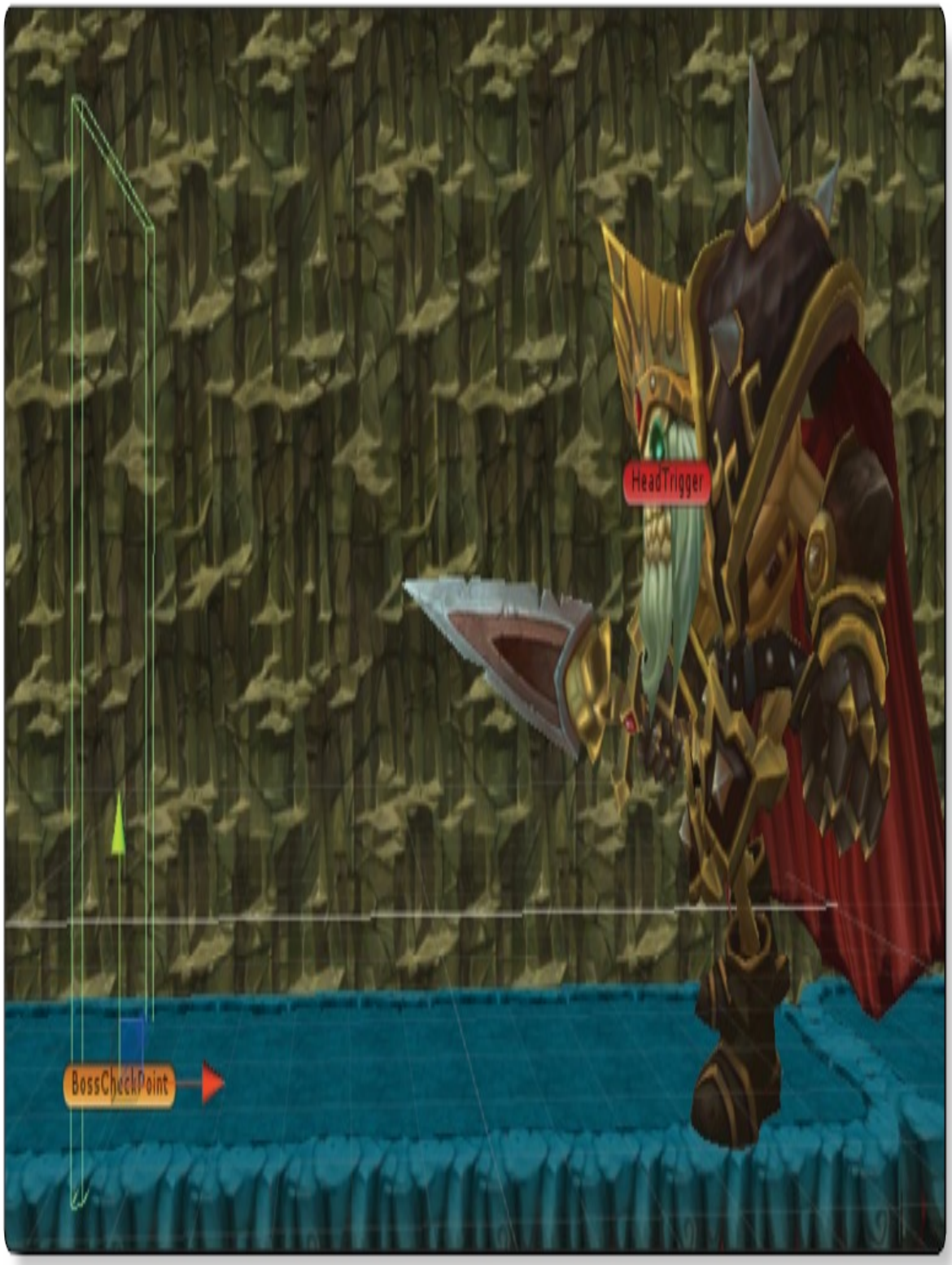
Haga un nuevo “Tag” y nómbrelo “BossCheckPoint”

Asigne el tag “BossCheckPoint” al objeto “BossCheckPoint”

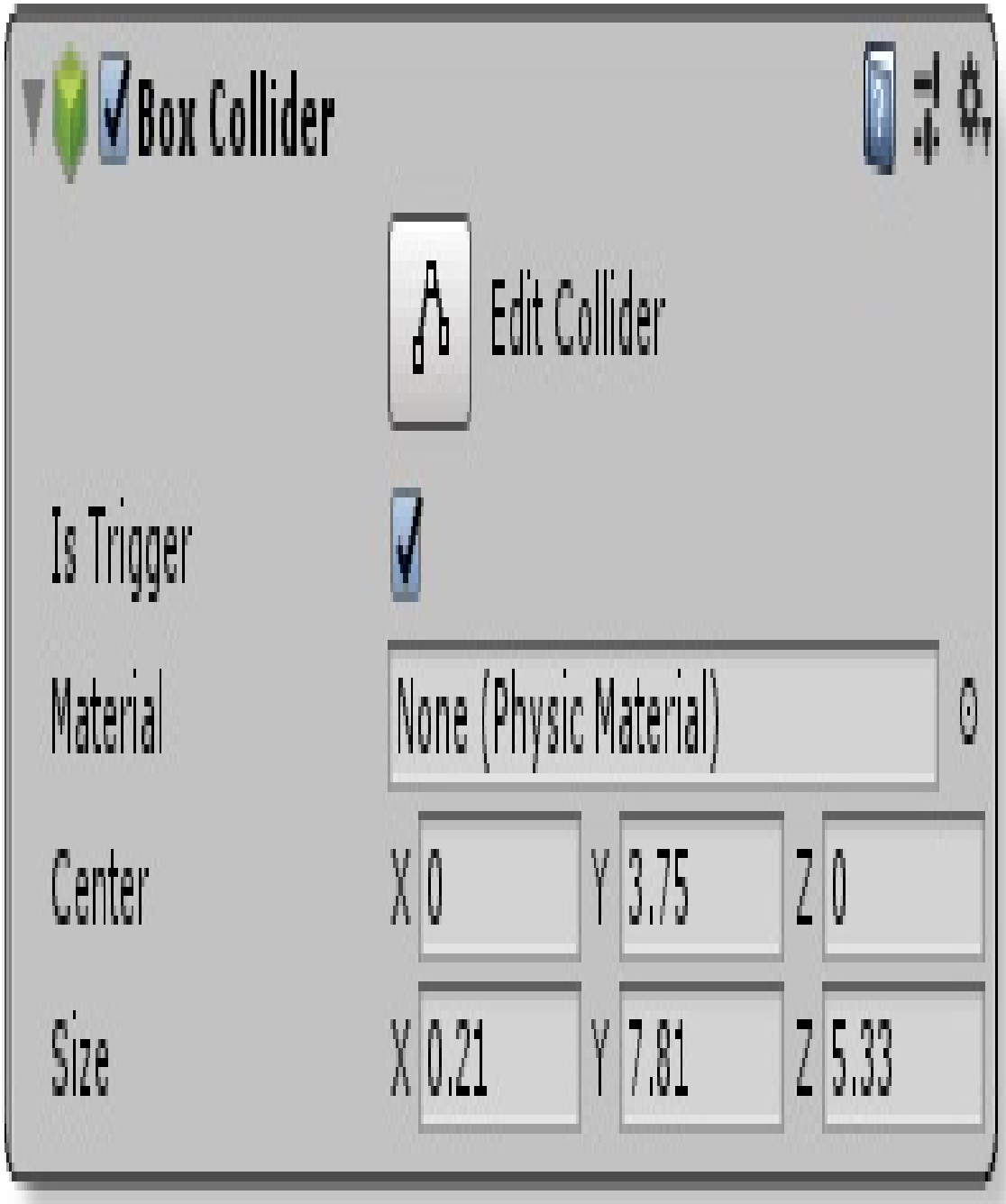
Asigne un label color naranja

Asigne un componente “BoxCollider” con los valores que se muestran

Colóquelo donde se muestra


HeadTrigger
BossCheckPoint

Creación del BossCheckPoint

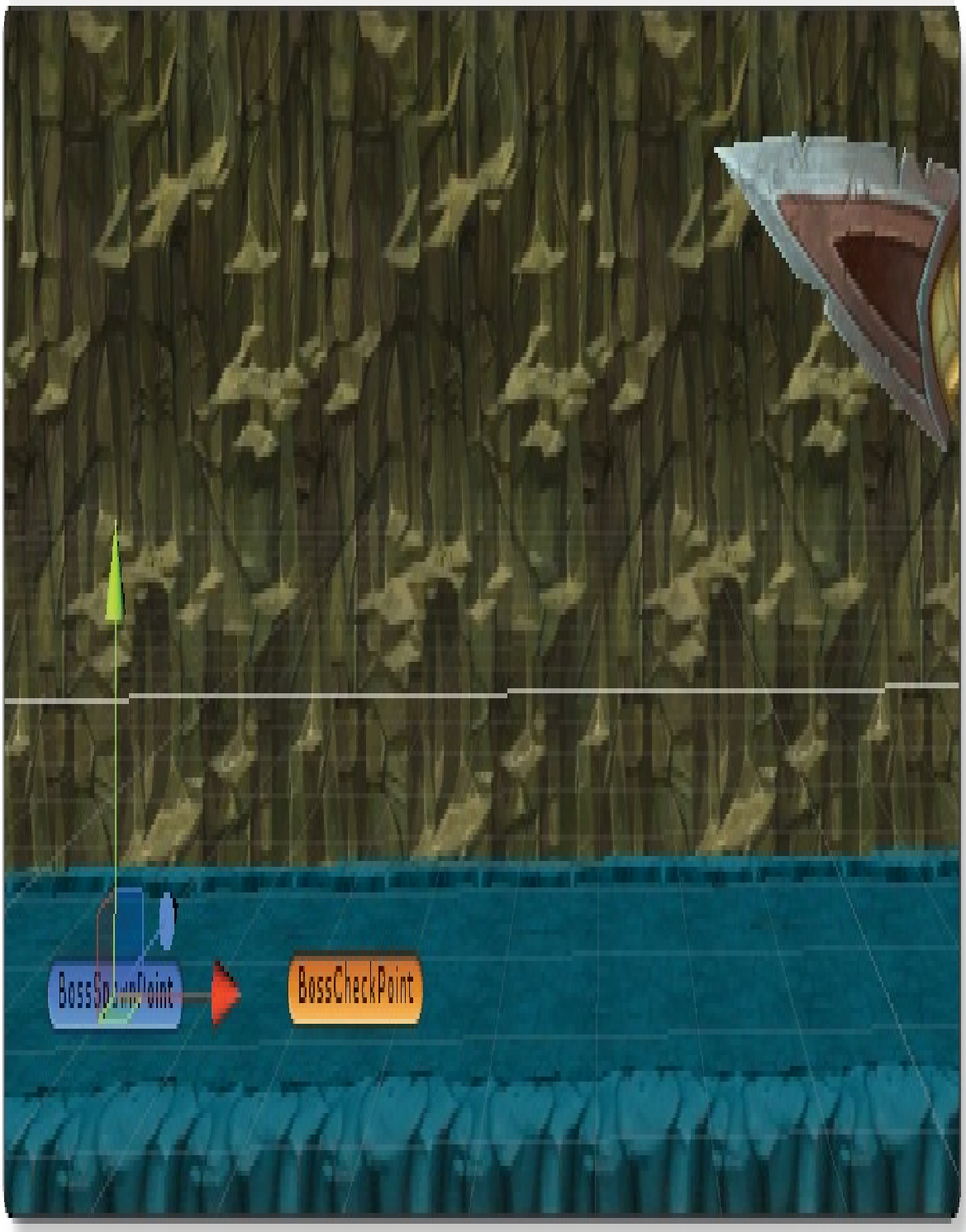

Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0
Y 3.75
Z 0
Size
X 0.21
Y 7.81
Z 5.33

Valores del BoxCollider

Asigne el tag “Player” al objeto “Player” como se muestra en la siguiente figura

Inspector
Player
Tag
Player
Laye
Prefab
Select
Revert

Haga un nuevo “EmptyObject”, renómbrelo “BossSpawnPoint”, asigne un label color azul y colóquelo detrás del “BossCheckPoint”.


BossSpawnPoint
BossCheckPoint

Creación de un “BossSpawnPoint”

Haga un nuevo C#Script, y nómbrelo “BossCheckPoint”.

Asigne el script “BossCheckPoint” al objeto “BossCheckpoint”.

Abra el script y escriba el siguiente código para convertir el “BossCheckPoint” en “No Trigger” cuando pase por él nuestro “Player”.

```
public class BossCheckPoint : MonoBehaviour {

    public BoxCollider collider;

    // Use this for initialization
    void Start () {
        collider = GetComponent<BoxCollider>();
    }

    // Update is called once per frame
    void Update () {

    }

    void OnTriggerExit (Collider other) {
        if (other.tag == "Player") {
            collider.isTrigger = false;
        }
    }
}
```

Script BossCheckPoint

NOTA:

Al desactivar su función “IsTrigger”, automáticamente se convierte en “Collider” y no permitirá que el “Player” vuelva a pasar por él.

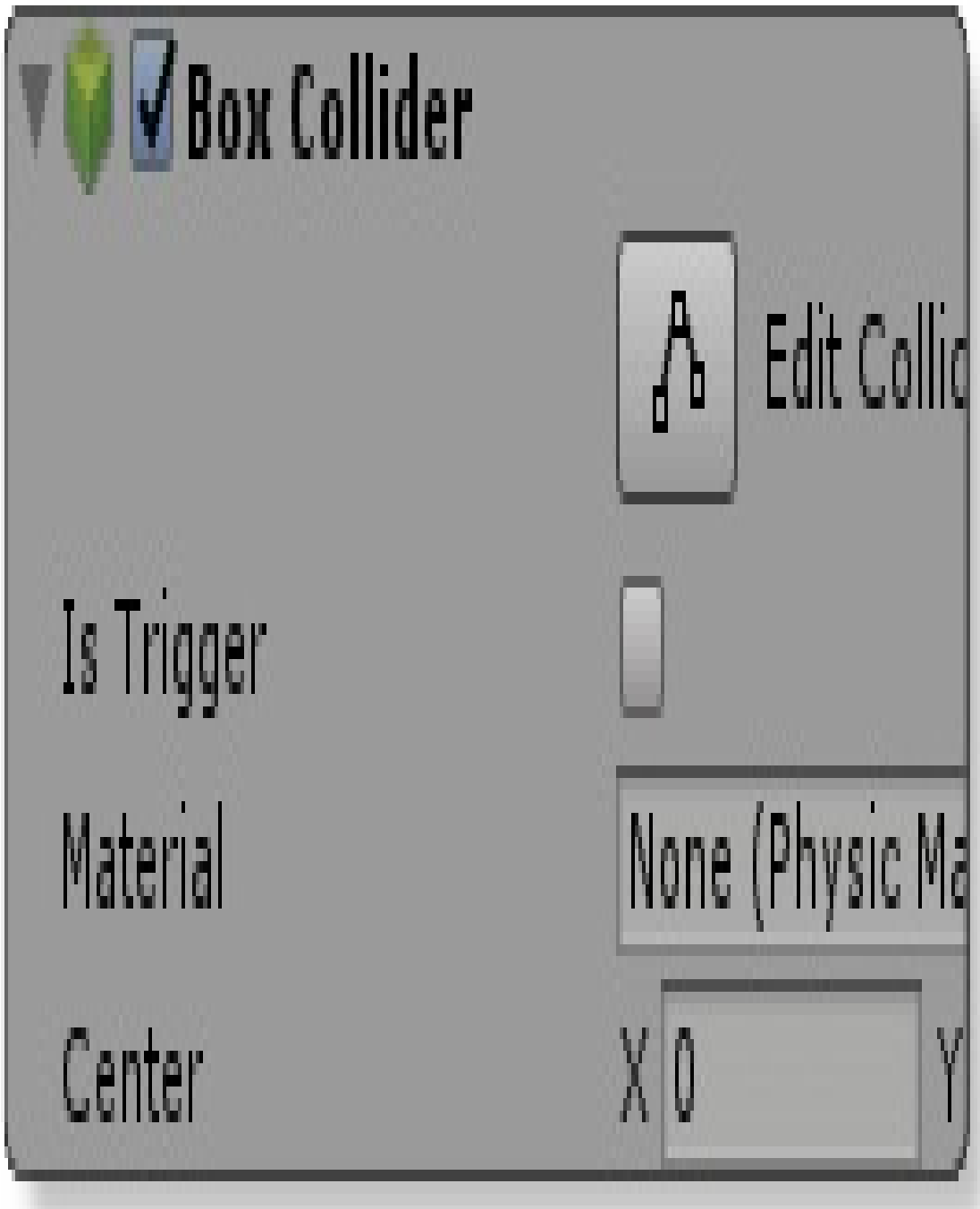
Previsualice el juego para observar lo siguiente:

La casilla de “IsTrigger” del BoxCollider de “BossCheckPoint” se desactiva cuando el Player pasa a través de ella.

El Player no puede pasar de regreso por el “BossCheckPoint” una vez que ya lo haya pasado.

HeadTrigger
KnifeSpawn
BossCheckPoint

Previsualización de funcionalidad de BossCheckPoint


Box Collider
Edit Collic
Is Trigger
Material
None (Physic Ma
Center
X 0
Y

Previsualización de desactivación de IsTrigger

Despertar al Jefe I

En este apartado crearemos la funcionalidad “BossAwake” para asignar diversos comportamientos al momento en que nuestro “Player” cruza el “BossCheckPoint”.

Haga un nuevo C#Script y nómbrelo “BossController”.

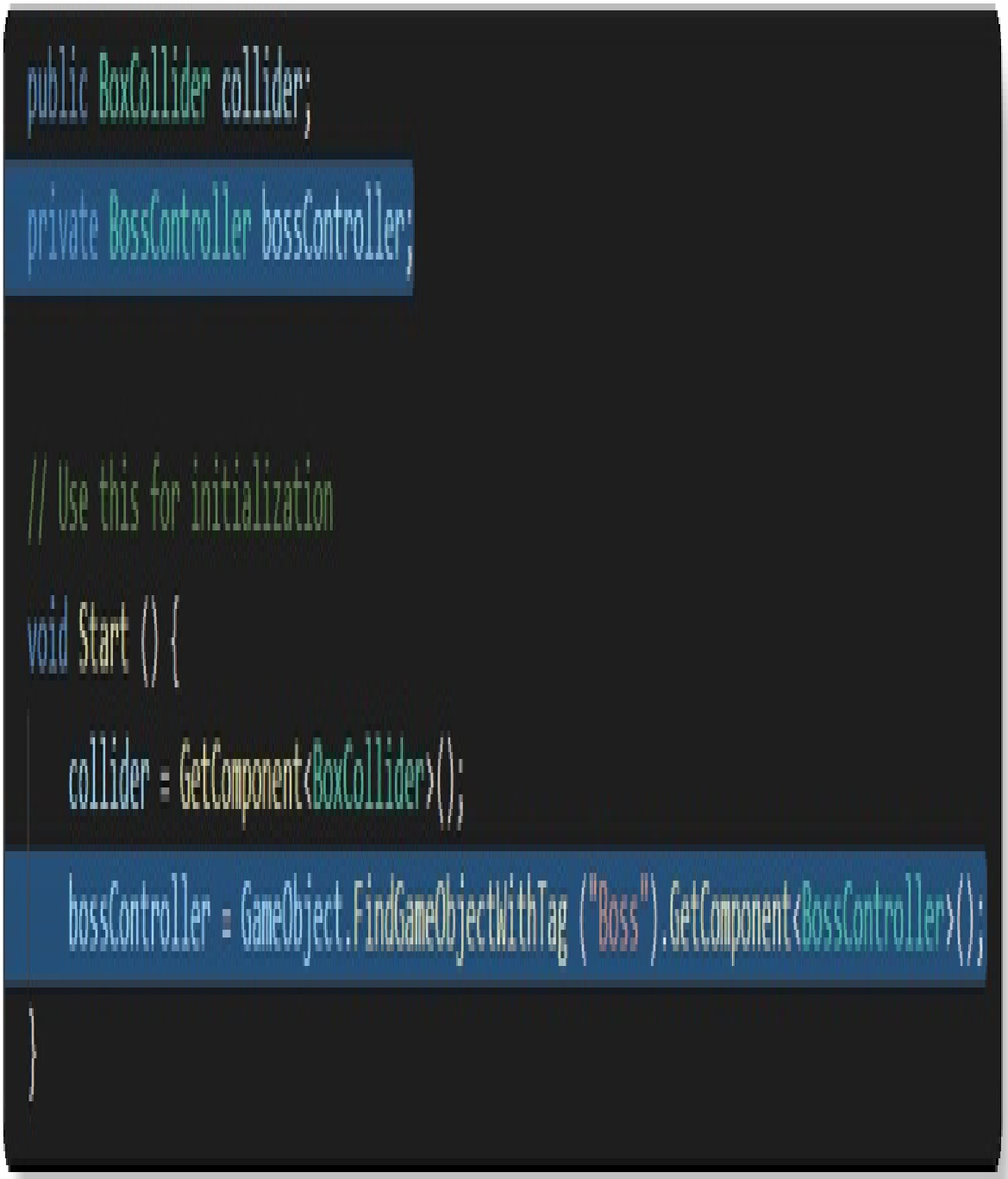
Asigne el nuevo script al objeto “Boss”.

Abra el script y escriba el siguiente código para comprobar que el Jefe despierta.

```
public class BossController : MonoBehaviour {

    public bool bossAwake = false;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

        if (bossAwake){
            print ("Boss is awake!");
        }
    }
}
```

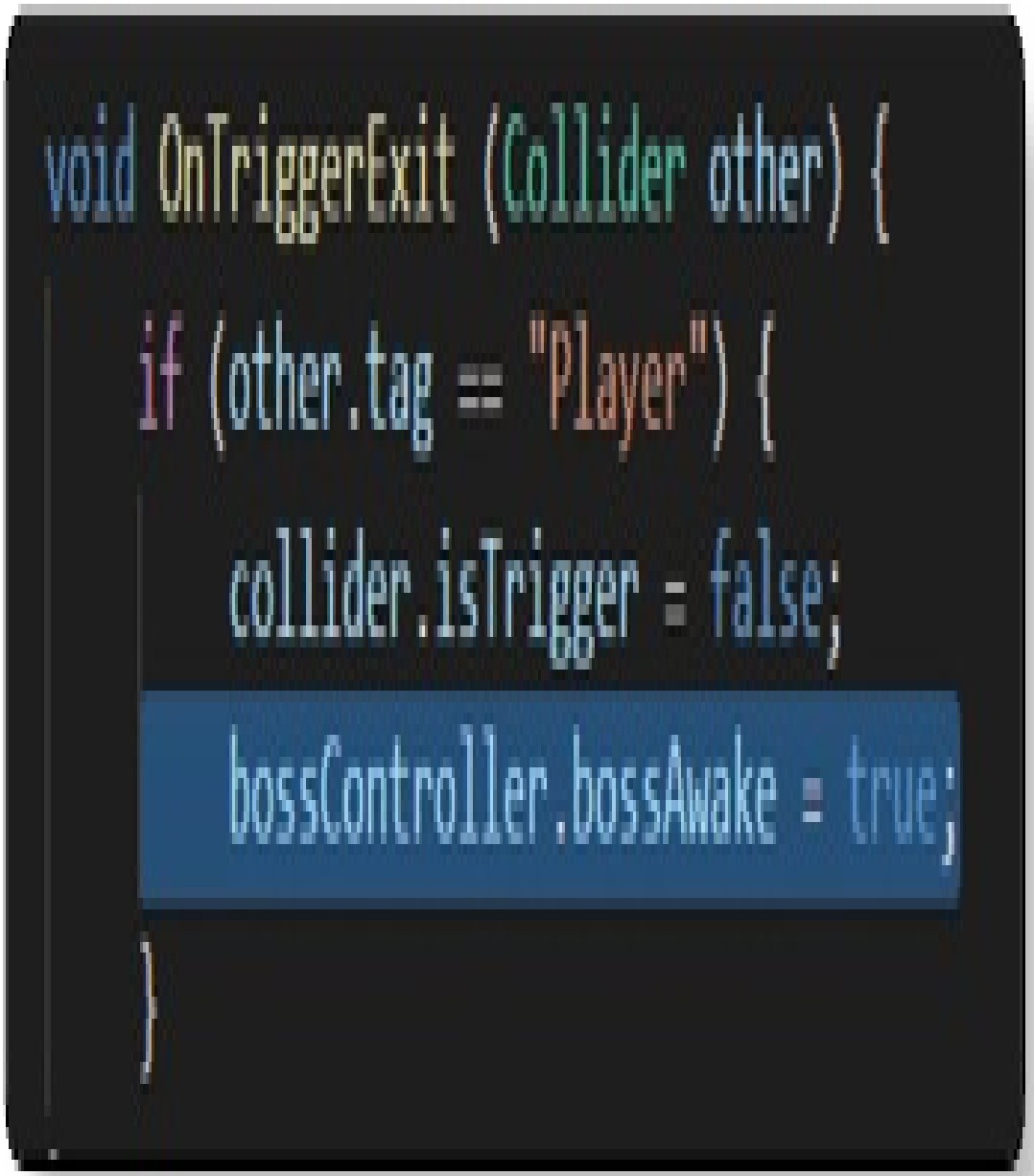
Script BossController

Vaya al script “BossCheckPoint”, haga e inicialice la siguiente variable para mandar llamar elementos del script “BossController”.


public BoxCollider collider;
private BossController bossController;
// Use this for initialization
void Start () {
collider = GetComponent<BoxCollider>();
bossController = GameObject.FindGameObjectWithTag ("Boss").GetComponent<BossController>();
}

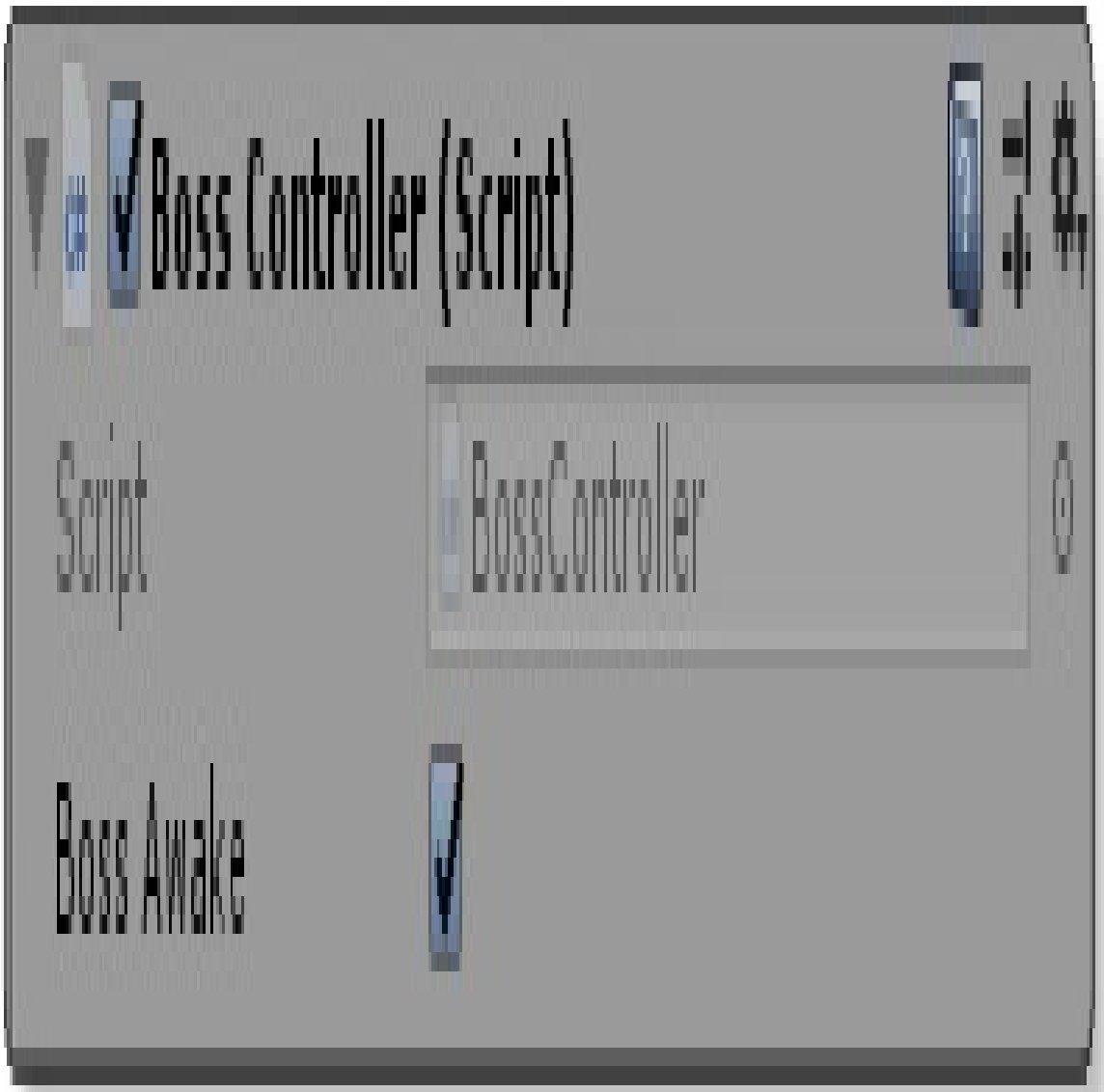
Variable para mandar llamar elementos de otro script

Vaya a la función “OnTriggerExit” y añada el siguiente código para activar “bossAwake” cuando nuestro Player pase por él.

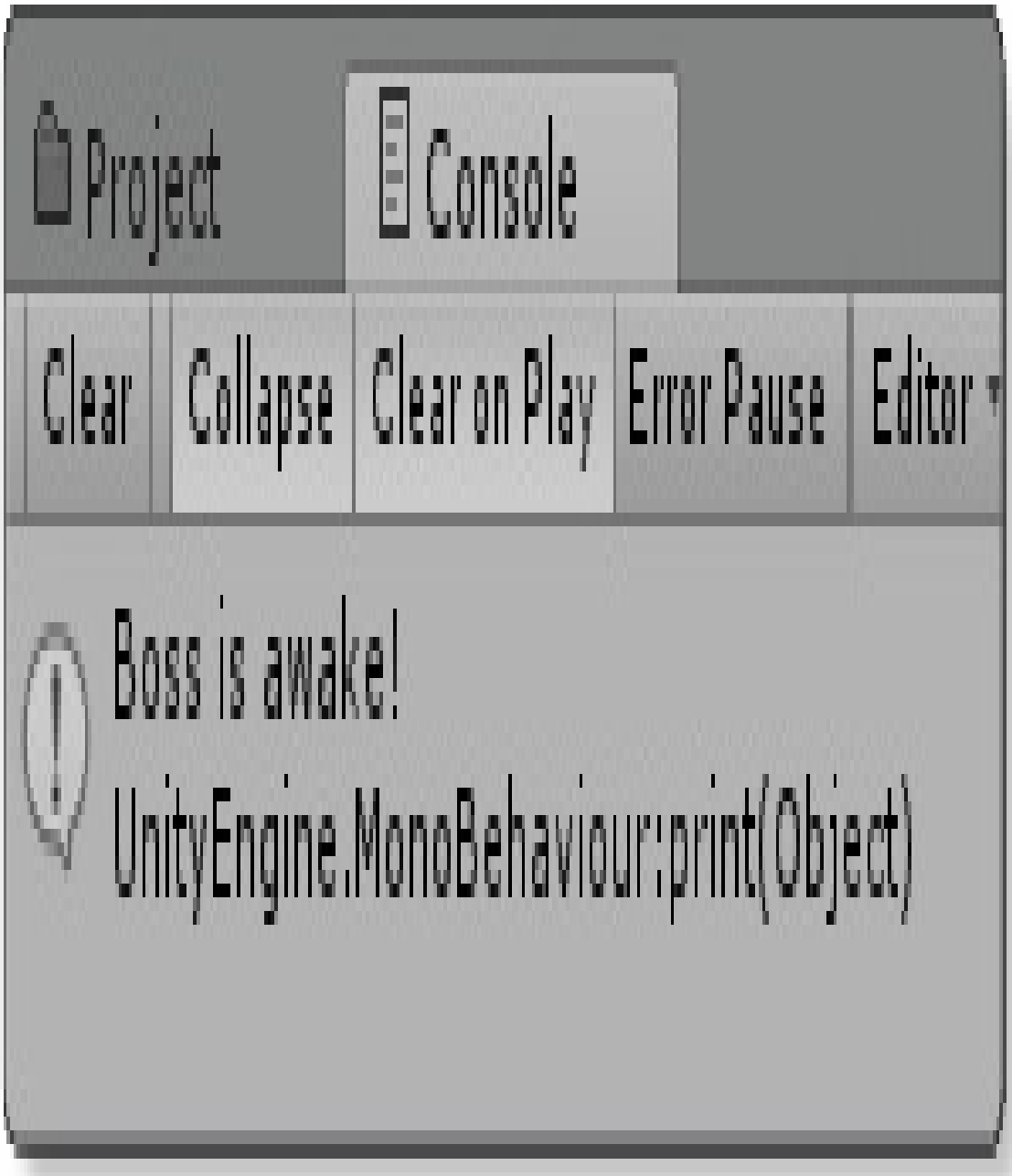

void OnTriggerExit (Collider other) {
if (other.tag == "Player") {
collider.isTrigger = false;
bossController.bossAwake = true;
}

Cambio de polaridad de variable booleana

Previsualice el juego y compruebe que se activa la casilla “bossAwake” cuando el “Player” pasa por “BossCheckPoint”.


Boss Controller (Script)
Script
BossController
Boss Awake

BossAwake activo


Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
Boss is awake!
UnityEngine.MonoBehaviour:print(Object)

Comprobación en Consola

A continuación, debemos de deshabilitar el movimiento del personaje por unos momentos para que corra la animación inicial del jefe.

Vaya al script “BossCheckPoint”, añada y edite las siguientes líneas de código para llamar elementos del script “CharacterMovement” del Player y deshabilitar los movimientos.

```
public BoxCollider collider;
private BossController bossController;
private CharacterMovement characterMovement;

// Use this for initialization
void Start () {
    collider = GetComponent<BoxCollider>();
    bossController = GameObject.FindGameObjectWithTag ("Boss").GetComponent<BossController>();
    characterMovement = GameObject.FindGameObjectWithTag ("Player").GetComponent<CharacterMovement>();
}

// Update is called once per frame
void Update () {

}

void OnTriggerExit (Collider other) {
    if (other.tag == "Player") {
        collider.isTrigger = false;
        bossController.bossAwake = true;
        characterMovement.enabled = false;
    }
}
```

Variable, inicialización y cambio de polaridad

Vaya a Unity, seleccione Player y previsualice el juego para observar que cuando el “Player” atraviesa el “BossCheckPoint” se desactiva su script “CharacterMovement” causando que se deshabiliten sus movimientos.

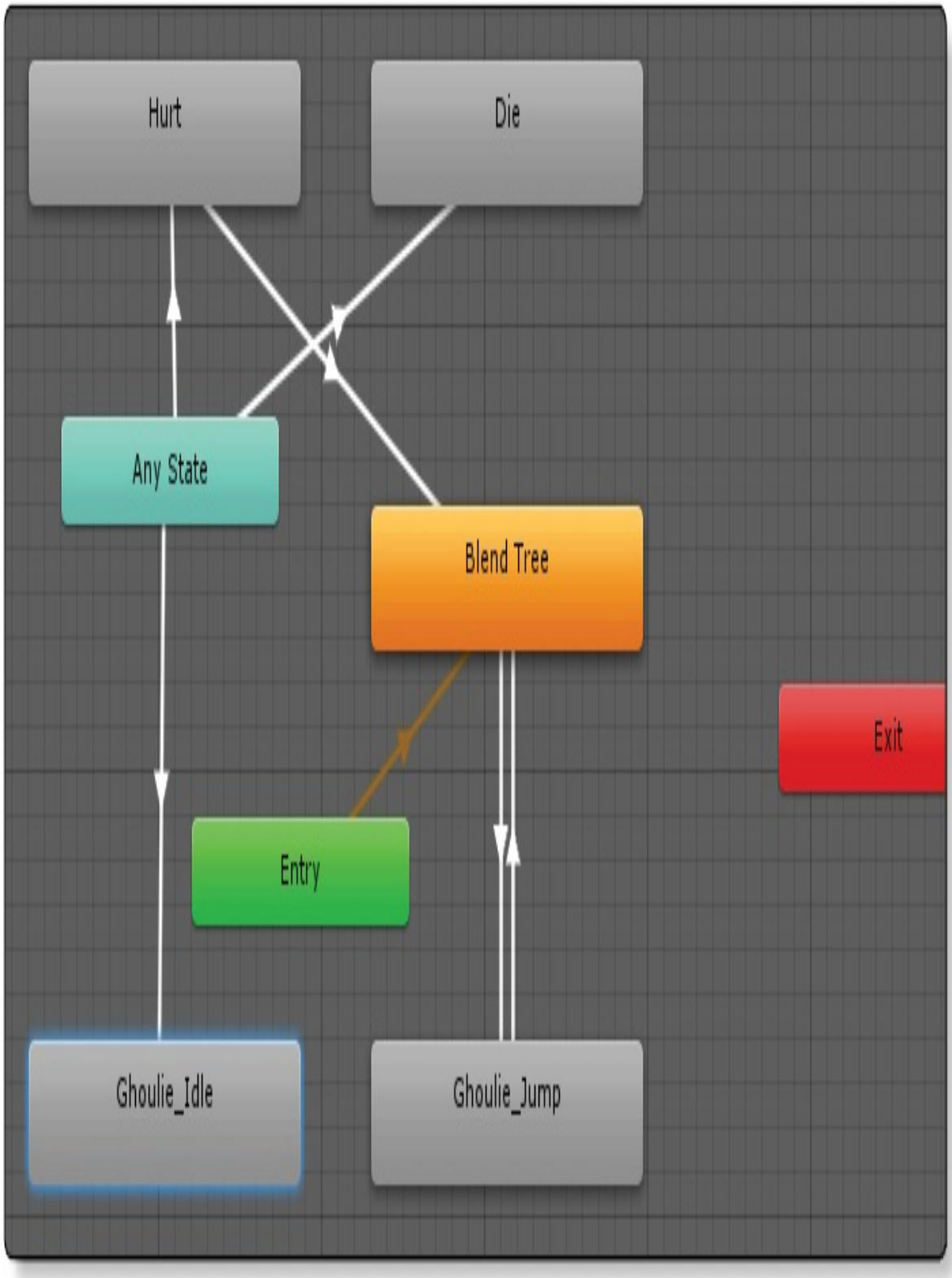
Character Movement (Script)
Script
CharacterMovement

Script inactivo

PROBLEMA: Al deshabilitarse el componente “CharacterMovement” del “Player” continúa corriendo la animación de corrido del mismo.

Player con su animación de correr

SOLUCIÓN: Seleccione “Player”, abra la ventana “Animator” y asigne su animación de “Idle” con la transición proveniente de “AnyState”.


Hurt
Die
Any State
Blend Tree
Exit
Entry
Ghoulie_Idle
Ghoulie_Jump

Asignación de animación en Animator

Vaya al script “BossCheckPoint” y añada el siguiente código para mandar llamar la animación “Ghoulie_Idle” al pasar por el “BossCheckPoint”.

```
private CharacterMovement characterMovement;
private Animator playerAnimator;


// Use this for initialization
void Start () {
    collider = GetComponent<BoxCollider>();
    bossController = GameObject.FindGameObjectWithTag ("Boss").GetComponent<BossController>(
    characterMovement = GameObject.FindGameObjectWithTag ("Player").GetComponent<CharacterMo
    playerAnimator = GameObject.FindGameObjectWithTag ("Player").GetComponent<Animator>();
}
```

Creación e inicialización de variable

```
void OnTriggerExit (Collider other) {
    if (other.tag == "Player") {
        collider.isTrigger = false;
        bossController.bossAwake = true;
        characterMovement.enabled = false;
        playerAnimator.Play("Ghoulie_Idle");
    }
}
```

Reproducir animación

Previsualice el juego para observar que al pasar nuestro “Player” por el “BossCheckPoint” corre su animación de “Idle” mientras su componente “CharacterMovement” se encuentra inactivo.

Trabajo con cámara para batalla final

En este apartado asignaremos la funcionalidad para que cuando se inicie la batalla final se cambie automáticamente el objetivo de la cámara para visualizar únicamente el entorno de la batalla.

Seleccione el objeto “CameraTarget” que se encuentra como hijo de “Player” y haga un duplicado.

Desemparente de la jerarquía el duplicado creado y renómbrelo “BossCameraTarget”.

Asigne un label color azul y colóquelo donde se muestra.

HeadTrigger
BossCameraTarget
BossCheckPoint

Posicionamiento de BossCameraTarget

Haga un nuevo “Tag”, nómbrelo “BossCameraTarget” y asígnelo al objeto “BossCameraTarget”.

Vaya al script “SmoothFollow”, haga las siguientes variables e inicialícelas.

```
public Transform cameraTarget;


public Transform bossCameraTarget;
public bool bossCameraActive = false;
public float cameraSpeed = 30.0f;


void Awake (){
    cameraTarget = GameObject.FindGameObjectWithTag ("CameraTarget").transform;
    bossCameraTarget = GameObject.FindGameObjectWithTag ("BossCameraTarget").transform;
}
```

Creación e inicialización de variables

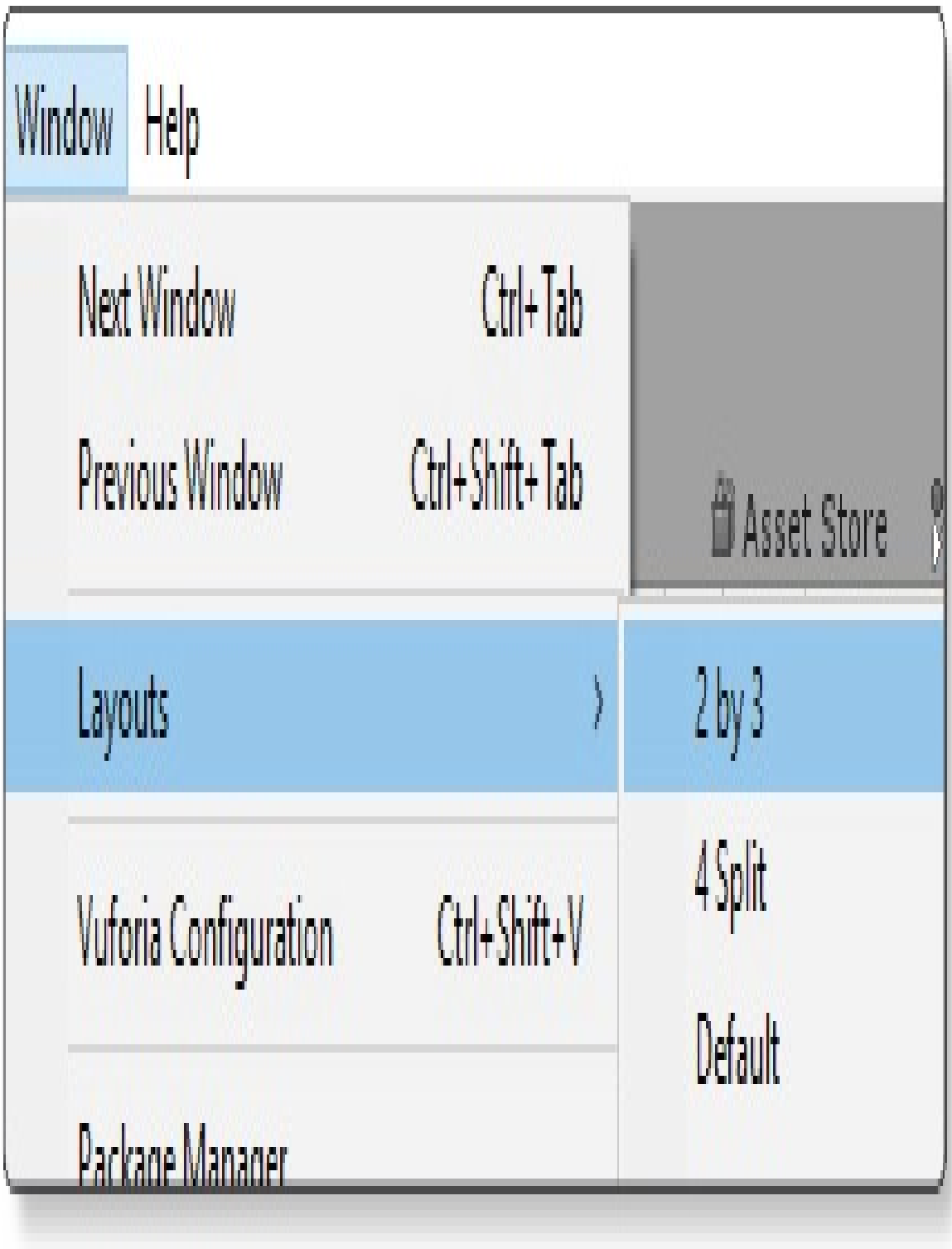
Edite el siguiente código en la función “FixedUpdate” para hacer la transición entre los “CameraTargets”.

```
targetX = Mathf.Clamp (targetX, minXandY.x, maxXandY.x);
targetY = Mathf.Clamp (targetY, minXandY.y, maxXandY.y);


if (bossCameraActive) {
    transform.position = new Vector3
    (Mathf.Lerp (transform.position.x, bossCameraTarget.position.x, 1.0f /cameraSpeed),
    Mathf.Lerp (transform.position.y, bossCameraTarget.position.y, 1.0f / cameraSpeed),
    Mathf.Lerp (transform.position.z, bossCameraTarget.position.z, 1.0f / cameraSpeed));
} else {
    transform.position = new Vector3 (targetX, targetY, transform.position.z);
}
```

Edición de código

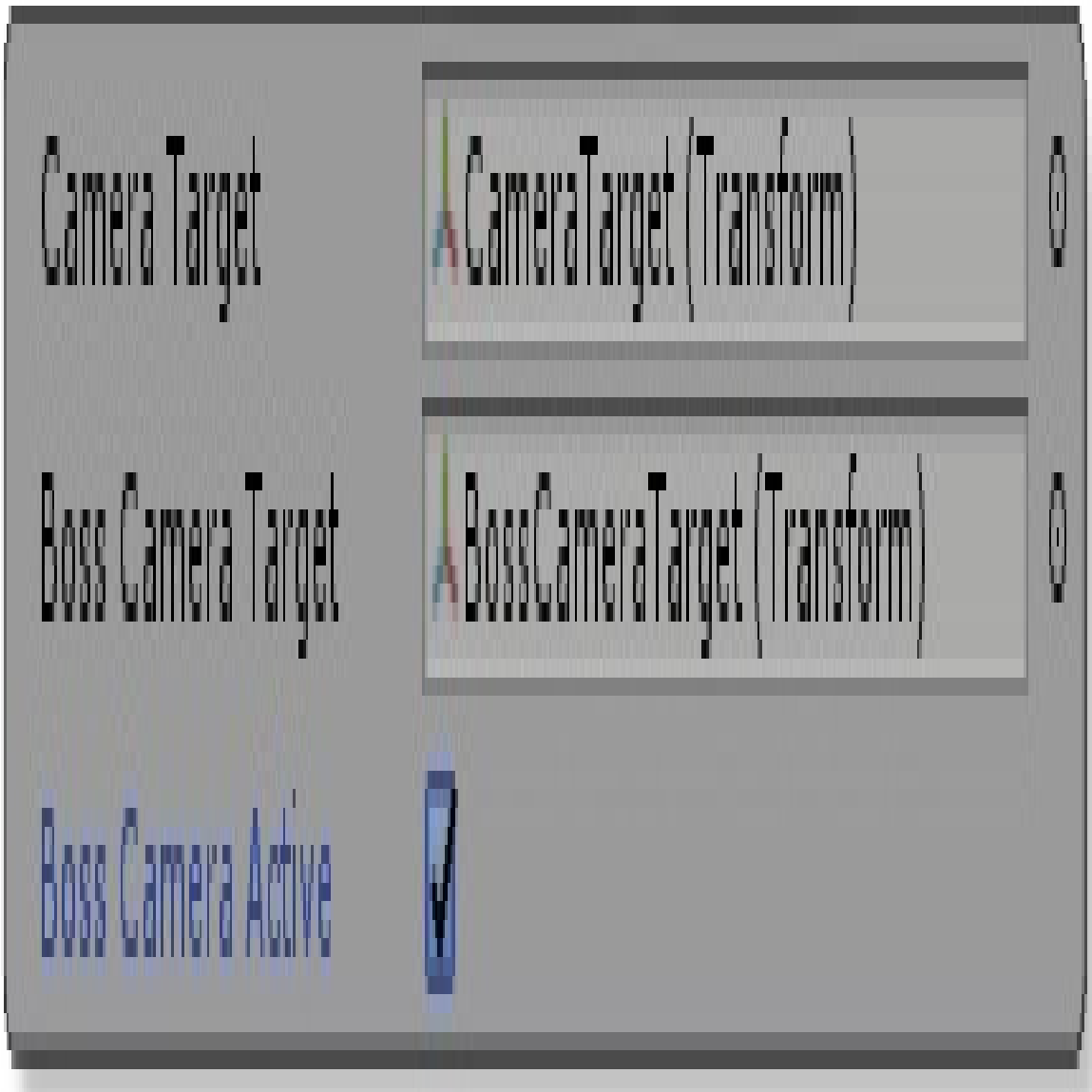
A continuación, reposicionará manualmente el “BossCameraTarget” a la posición ideal para la batalla.

Active el “Layout: 2 by 3” en el menú general para poder observar la vista de “Game” y de “Scene” simultáneamente.


Window
Help
Next Window
Ctrl+Tab
Previous Window
Ctrl+Shift+Tab
Asset Store
Layouts
2 by 3
4 Split
Default
Vuforia Configuration
Ctrl+Shift+V
Package Manager

Cambio de Layout

Previsualice el juego, seleccione el objeto “MainCamera” y active su casilla de “BossCameraActive”.


Camera Target
CameraTarget (Transform)
Boss Camera Target
BossCameraTarget (Transform)
Boss Camera Active

Activación manual de BossCameraActive

Mueva el objeto “CameraTarget” en la vista “Game” para que se coloque la cámara de la batalla.

Scene
Shaded
2D
Gizmos
Persp
BossCameraTarget
KnifeSpawn
BossSpawnPoint
Camera Preview
Game
Display 1
Free Aspect
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
X 3

Posicionamiento manual de BossCameraTarget

Asigne el tag “MainCamera” al objeto MainCamera como se muestra en la siguiente figura

Inspector
Main Camera
Tag
MainCamera
Laye

NOTA:

Tome el registro de la nueva posición del “BossCameraTarget” ya que cuando detenga la previsualización regresará a su posición anterior y deberá de asignar nuevamente y manualmente la posición registrada al momento de haber corrido el juego.

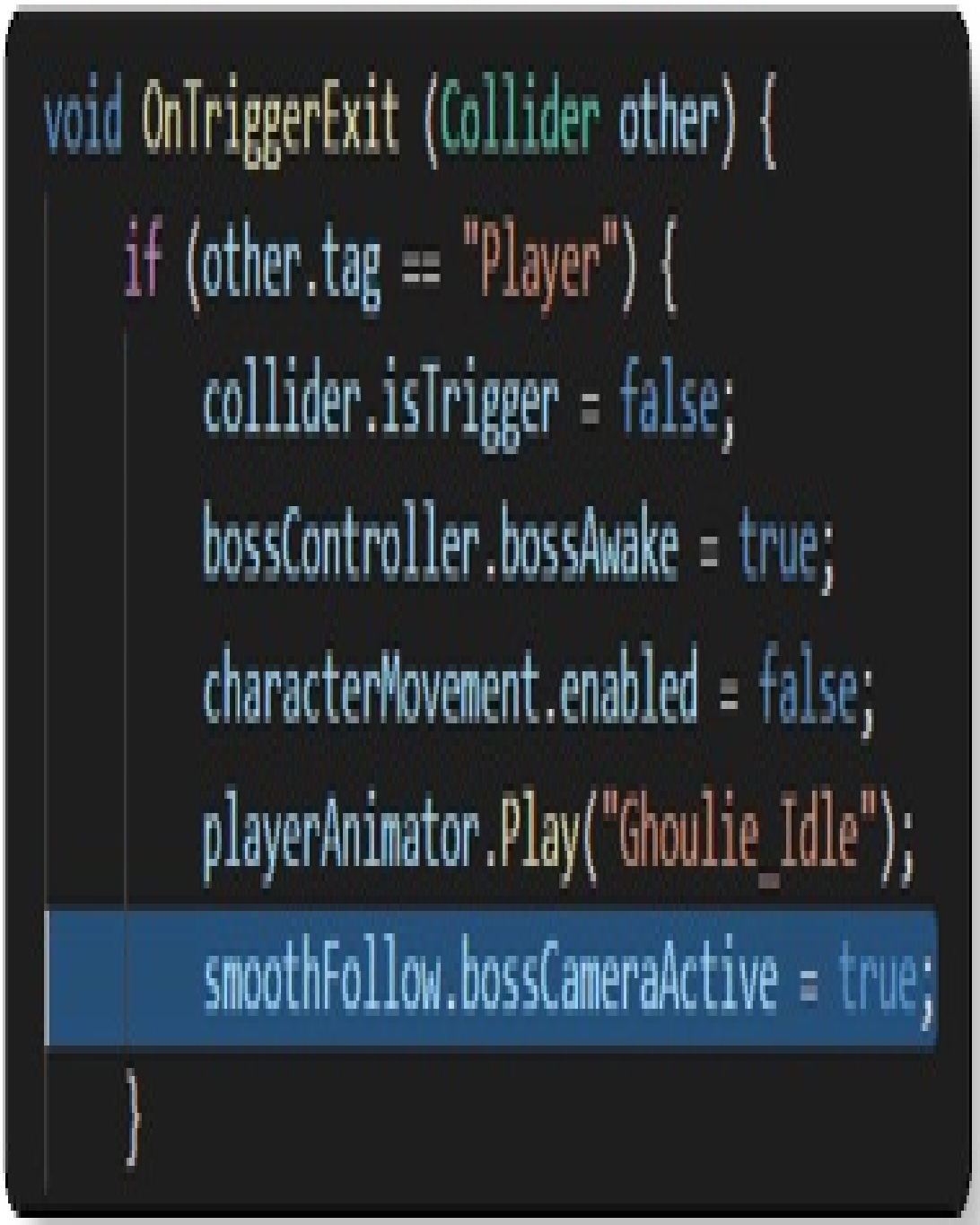
A continuación, programaremos la funcionalidad para que se active “BossCameraTarget” cuando el “Player” atraviese el “BossCheckPoint”.

Vaya al script “BossCheckPoint”, escriba e inicialice las variables que se muestran.

```
private Animator playerAnimator;
private SmoothFollow smoothFollow;


// Use this for initialization
void Start () {
    collider = GetComponent<BoxCollider>();
    bossController = GameObject.FindGameObjectWithTag ("Boss").GetComponent<BossController>();
    characterMovement = GameObject.FindGameObjectWithTag ("Player").GetComponent<CharacterMovement>
    playerAnimator = GameObject.FindGameObjectWithTag ("Player").GetComponent<Animator>();
    smoothFollow = GameObject.FindGameObjectWithTag ("MainCamera").GetComponent<SmoothFollow>();
}
```

Creación e inicialización de variables

Escriba el siguiente código en la función “OnTriggerExit” para activar el “BossCameraTarget”.


void OnTriggerExit (Collider other) {
if (other.tag == "Player") {
collider.isTrigger = false;
bossController.bossAwake = true;
characterMovement.enabled = false;
playerAnimator.Play("Ghoulie_Idle");
smoothFollow.bossCameraActive = true;
}

Activación de BossCameraActive por código

Previsualice el juego para observar que la cámara cambia cuando nuestro “Player” pasa a través de “BossCheckPoint”.

Creación de las animaciones del Jefe Final

En este apartado generaremos las animaciones necesarias para las condiciones y los ataques de nuestro jefe y programaremos sus animaciones iniciales conforme a su condición de “Awake”.

Con los conceptos aprendidos haga los clips de animación del jefe conforme a sus tiempos y asignando los nombres indicados.

frame 0236-0265: Boss_Hurt

frame 0613-0647: Boss_Jump

frame 0647-0699: Boss_Attack01

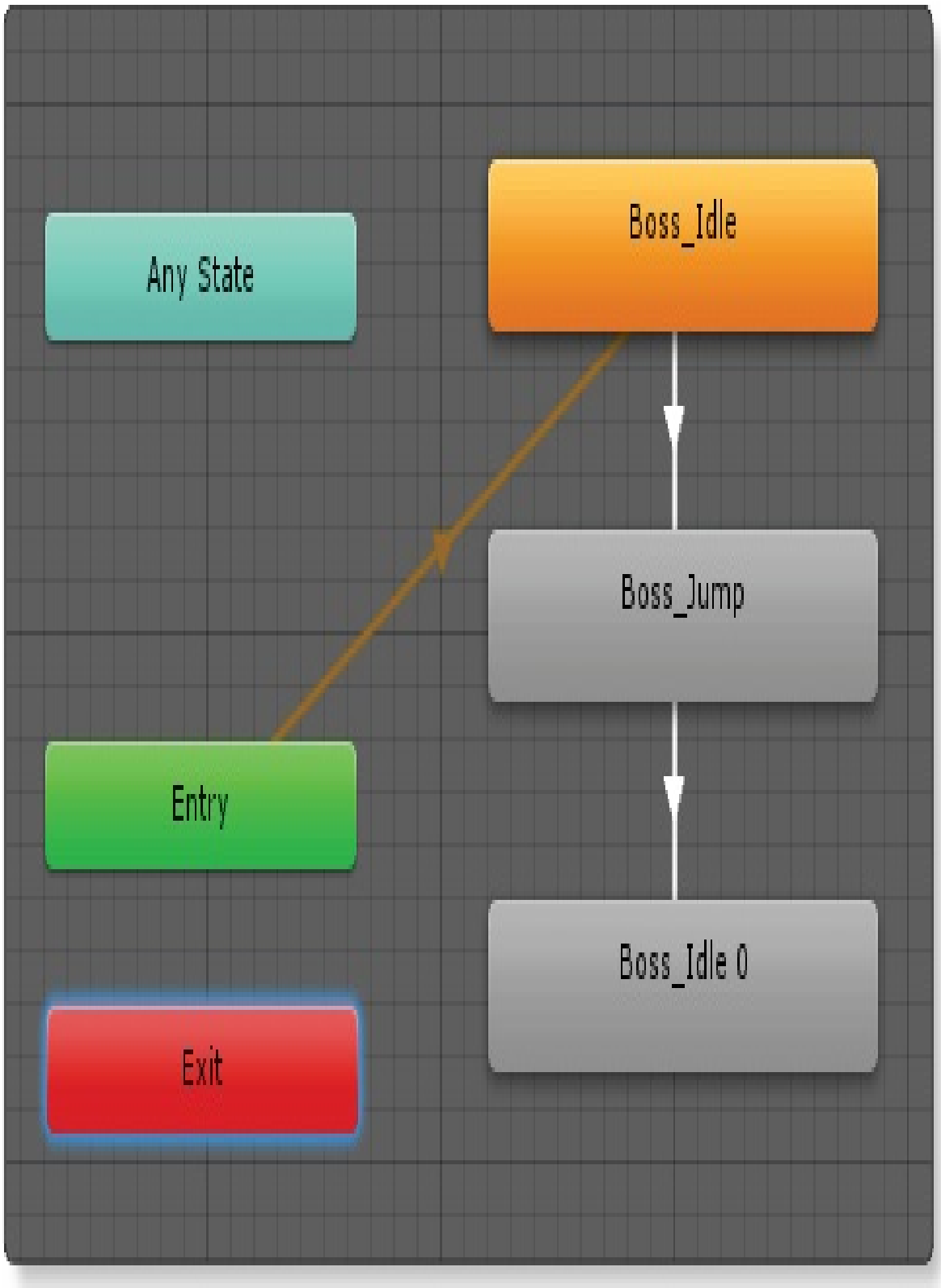
frame 0699-0752: Boss_Attack 2

frame: 0752-0934: Boss_Attack 3

frame: 1254-1303: Boss_Spawn

frame: 1303-1377: Boss_Death

Seleccione “Boss”, vaya a la ventana Animator y asigne las animaciones como se muestran con sus transiciones.

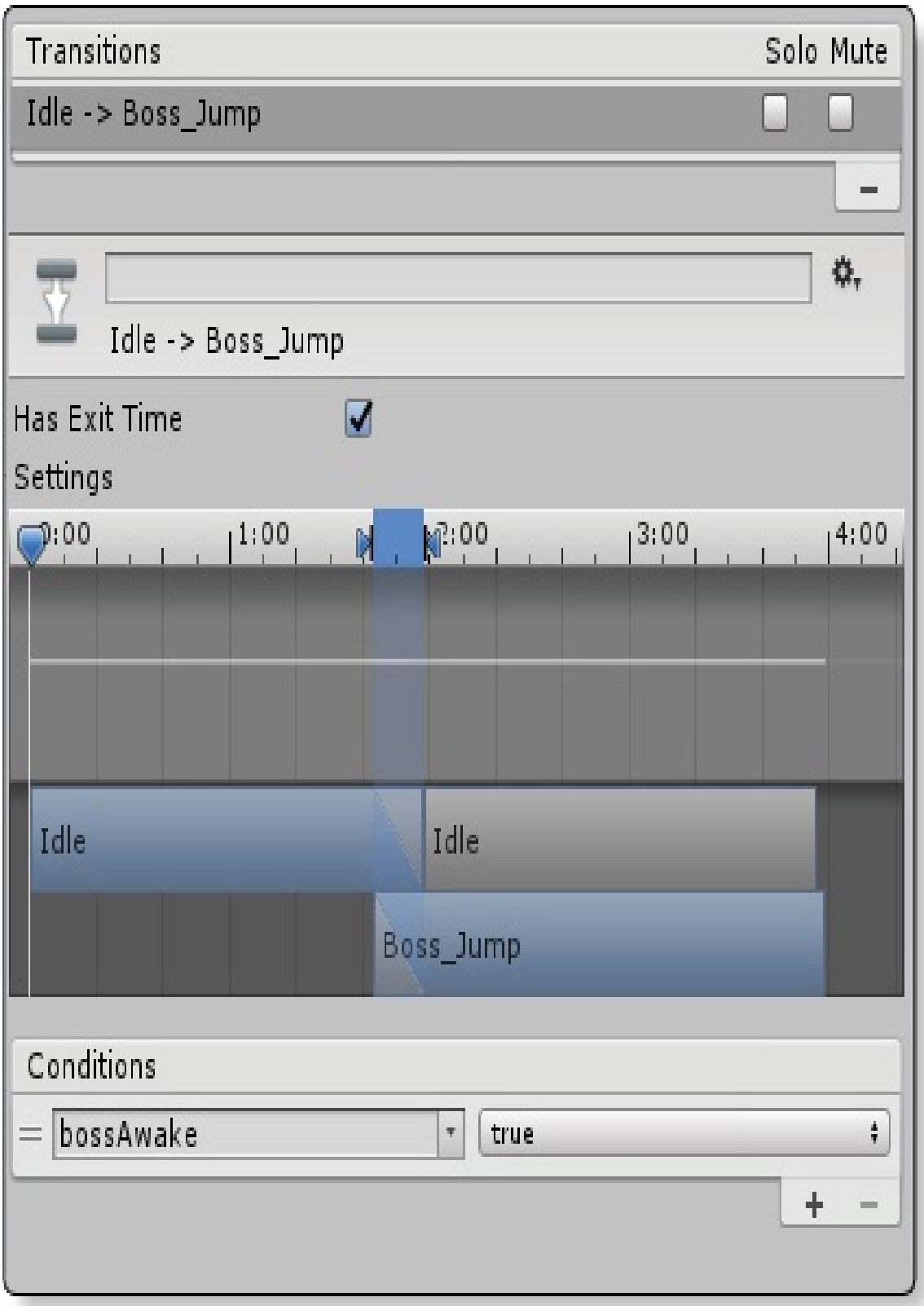

Any State
Boss_Idle
Boss_Jump
Entry
Boss_Idle 0
Exit

Asignación de clips de animación en Animator

Haga el siguiente parámetro booleano y asígnelo a la transición de Idle a Jump.

Layers
Parameters
ayName
bossAwake

Creación de parámetro booleano en Animator


Transitions
Solo Mute
Idle -> Boss_Jump
Idle -> Boss_Jump
Has Exit Time
Settings
1:00
3:00
4:00
Idle
Idle
Boss_Jump
Conditions
bossAwake
true

Asignación de parámetro

Vaya al script de BossController y escriba el siguiente código para que cuando se despierte el jefe, active el parámetro de “bossAwake”.

```
public bool bossAwake = false;
private Animator anim;

// Use this for initialization
void Start () {
    anim = GetComponent<Animator>();
}

// Update is called once per frame
void Update () {

    if (bossAwake){
        print ("Boss is awake!");
        anim.SetBool("bossAwake", true);
    }
}
```

Código para activar el parámetro de Animation

En Animator seleccione la animación del salto y baje su “Speed” a 0.5.

Previsualice el juego para observar que el jefe da un salto cuando despierta e inmediatamente regresa a su animación de Idle.

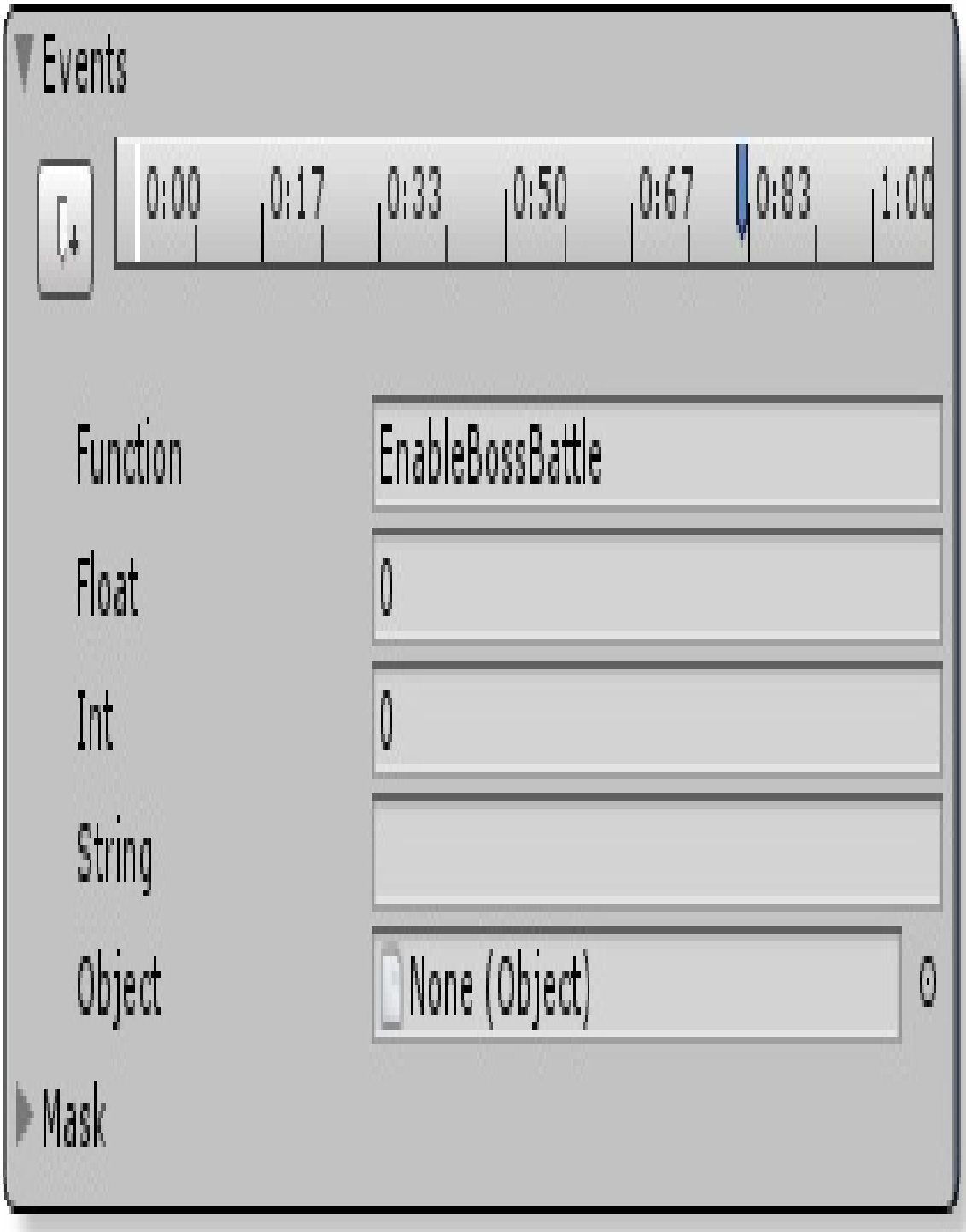
Previsualización de animación de salto

Reactivación de Player con Animation Event

En este apartado crearemos un evento de animación en el clip de salto del jefe que permitirá reactivar la movilidad de nuestro “Player” cuando este por terminar el salto.

En la ventana “Project” seleccione el clip de animación de “Jump” y haga un evento colocando su indicador azul en la parte final del clip.

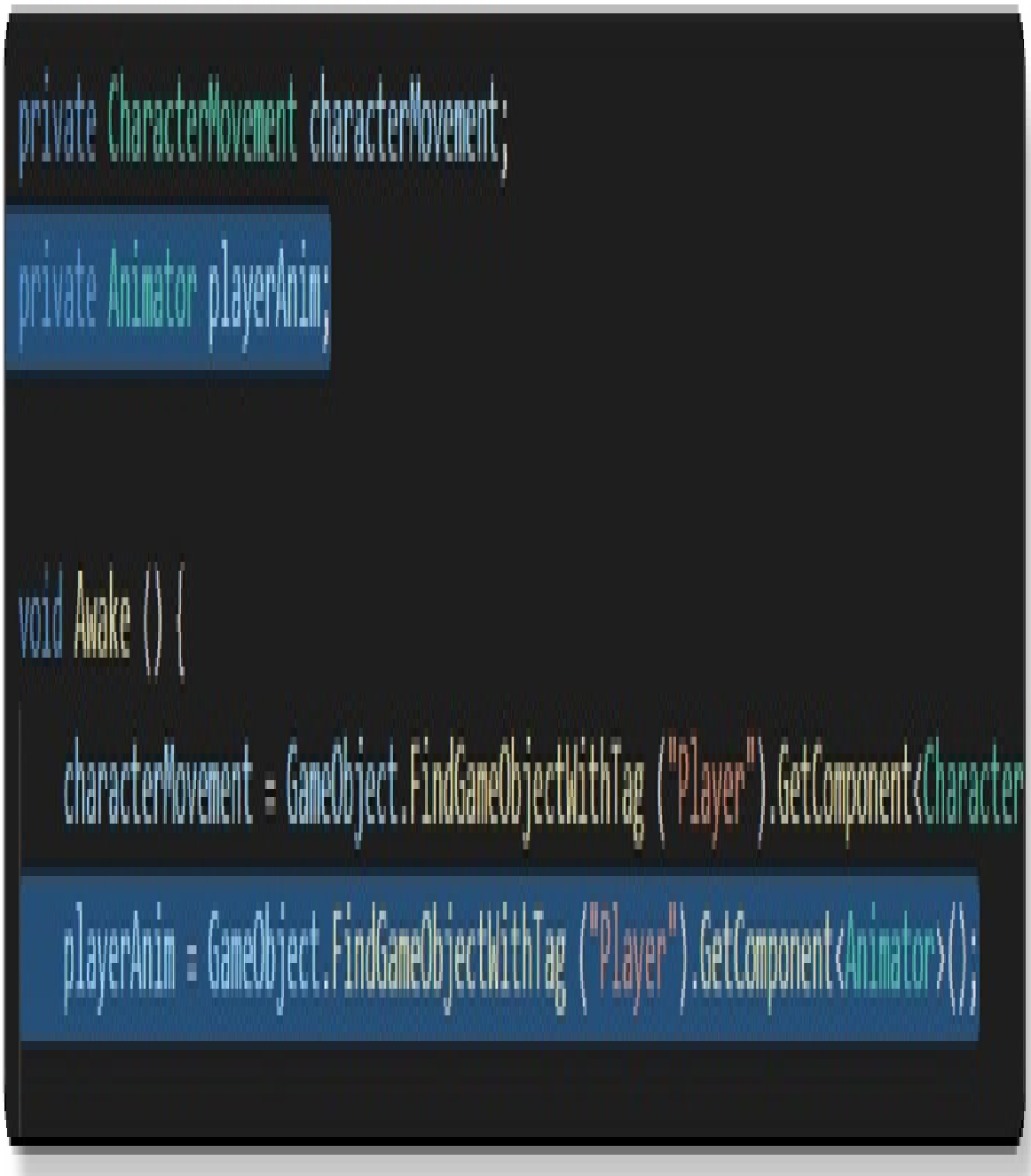
Haga un nuevo C#Script y nómbrelo “EventHandler”


Events
0:00
0:17
0:33
0:50
0:67
0:83
1:00
Function
EnableBossBattle
Float
0
Int
0
String
Object
None (Object)
Mask

Creación de AnimationEvent

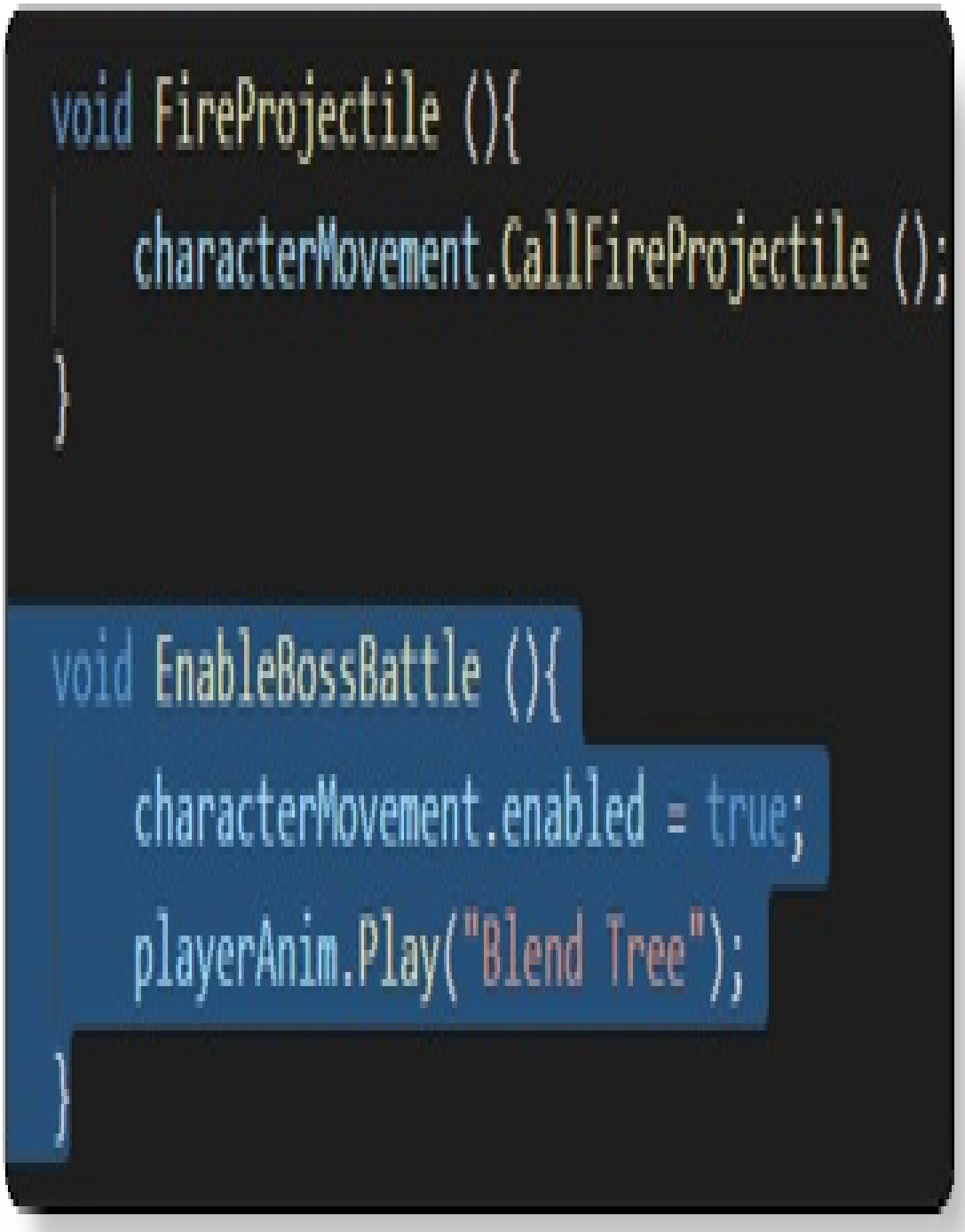
Asigne el script “EventHandler” al objeto “Boss”.

Vaya al script “EventHandler”, escriba e inicialice la variable que se muestra.


private CharacterMovement characterMovement;
private Animator playerAnim;
void Awake () {
characterMovement = GameObject.FindGameObjectWithTag ("Player").GetComponent<Character
playerAnim = GameObject.FindGameObjectWithTag ("Player").GetComponent<Animator>();

Creación e inicialización de variables

En el mismo script haga la función que se muestra.


void FireProjectile (){
characterMovement.CallFireProjectile ();
}
void EnableBossBattle (){
characterMovement.enabled = true;
playerAnim.Play("Blend Tree");
}

Creación de función “EnableBossBattle”

Previsualice el juego para observar que al terminar el salto del jefe se reactiva el componente “CharacterMovement” de nuestro Player y recupera la movilidad.

Preparación de la condición de ataque del Jefe

En este apartado programaremos la funcionalidad y la mecánica de ataque de nuestro Boss conforme a un sistema por tiempos.

También crearemos y programaremos un sistema de salud para nuestro jefe y un sistema de ataque por tiempos.

Vaya al script “BossController” y escriba las siguientes variables.

```
private Animator anim;


    public bool inBattle = false;
    public bool attacking = false;
    public float idleTimer = 0.0f;
    public float idleWaitTime = 10.0f;
```

Creación de variables

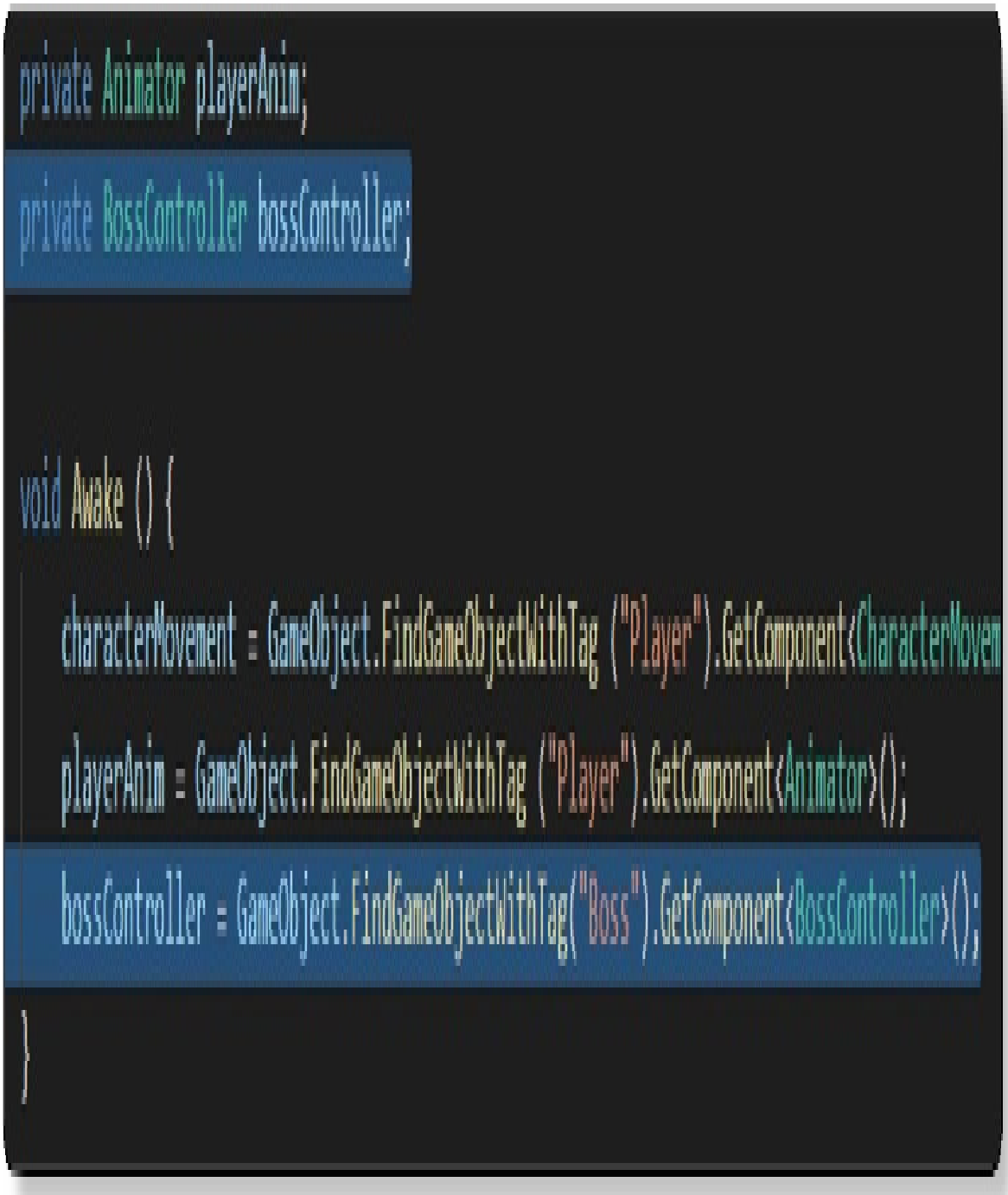
Vaya a la función “Update” y escriba el siguiente código para crear el estado de ataque cuando el jefe despierte.

```
if (bossAwake){
    print ("Boss is awake!");
    anim.SetBool("bossAwake", true);

    if(inBattle){
        if(!attacking){
            idleTimer += Time.deltaTime;
        }
        if(idleTimer >= idleWaitTime){
            print ("Boss is attacking!");
            attacking = true;
            idleTimer = 0.0f;
        }
    } else {
        idleTimer = 0.0f;
    }
}
```

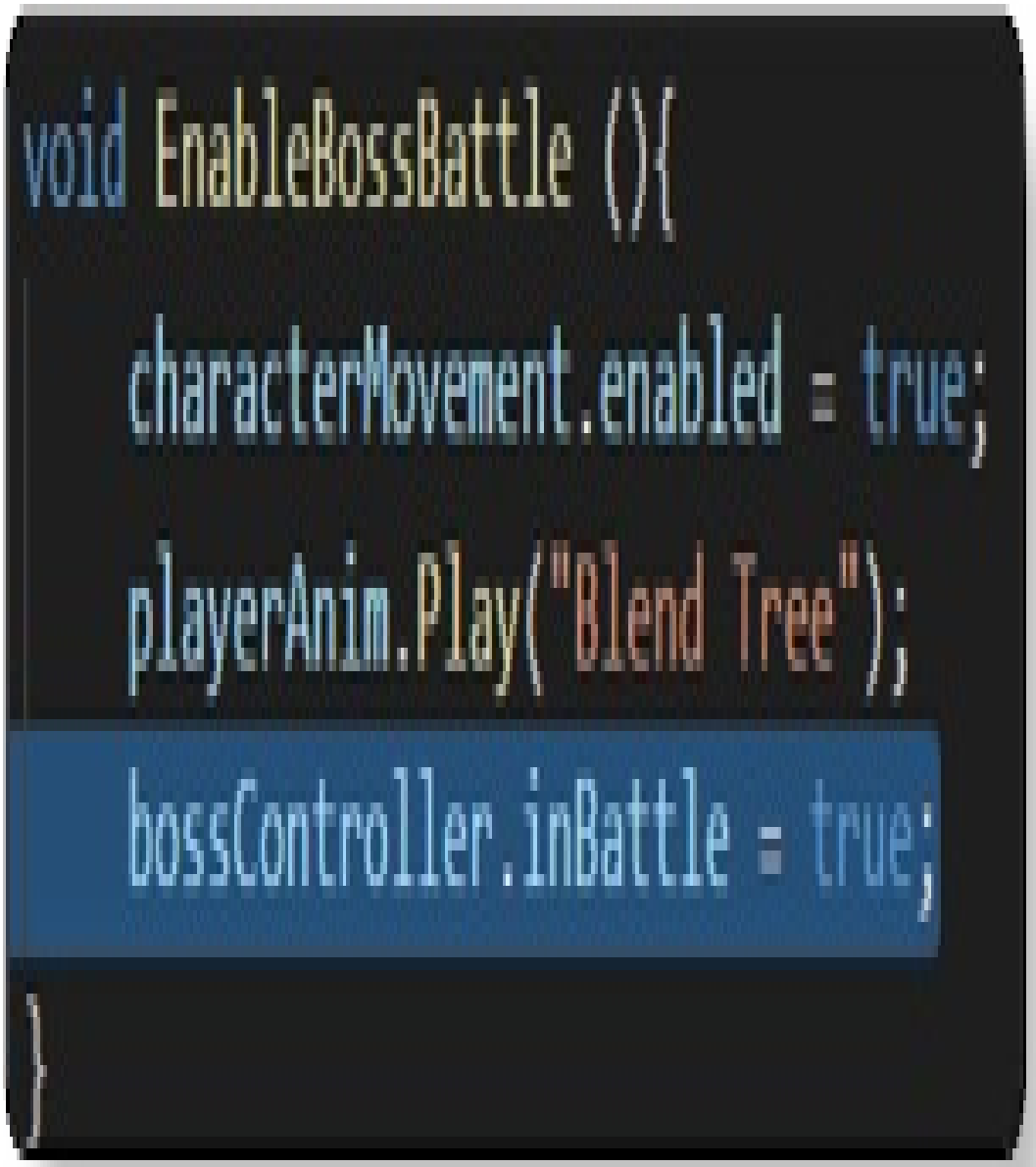
Funcionalidad para el estado de ataque

Para que esta funcionalidad arranque, es necesario asignar un valor verdadero a la variable “inBattle”, lo cual haremos en la función “EnableBossBattle” del script “EventHandler”.

Vaya al script “EventHandler”, escriba e inicialice las variables que se indican y añada el código que se muestra en la función “EnableBossBattle”.


private Animator playerAnim;
private BossController bossController;
void Awake () {
characterMovement = GameObject.FindGameObjectWithTag ("Player").GetComponent<CharacterMovem
playerAnim = GameObject.FindGameObjectWithTag ("Player").GetComponent<Animator>();
bossController = GameObject.FindGameObjectWithTag("Boss").GetComponent<BossController>();
}

Creación e inicialización de variables


void EnableBossBattle (){
characterMovement.enabled = true;
playerAnim.Play("Blend Tree");
bossController.inBattle = true;
}

Asignación de valor para varialbe inBattle

Vaya a Unity, seleccione el objeto “Boss” y previsualice el juego para observar lo siguiente cuando nuestro “Player” atraviesa el “BossCheckPoint”.

Se activan las casillas de “BossAwake” y “InBattle”

Comienza a correr el tiempo en “IdleTimer” y cuando este llega a 10 se activa la casilla “Attacking”

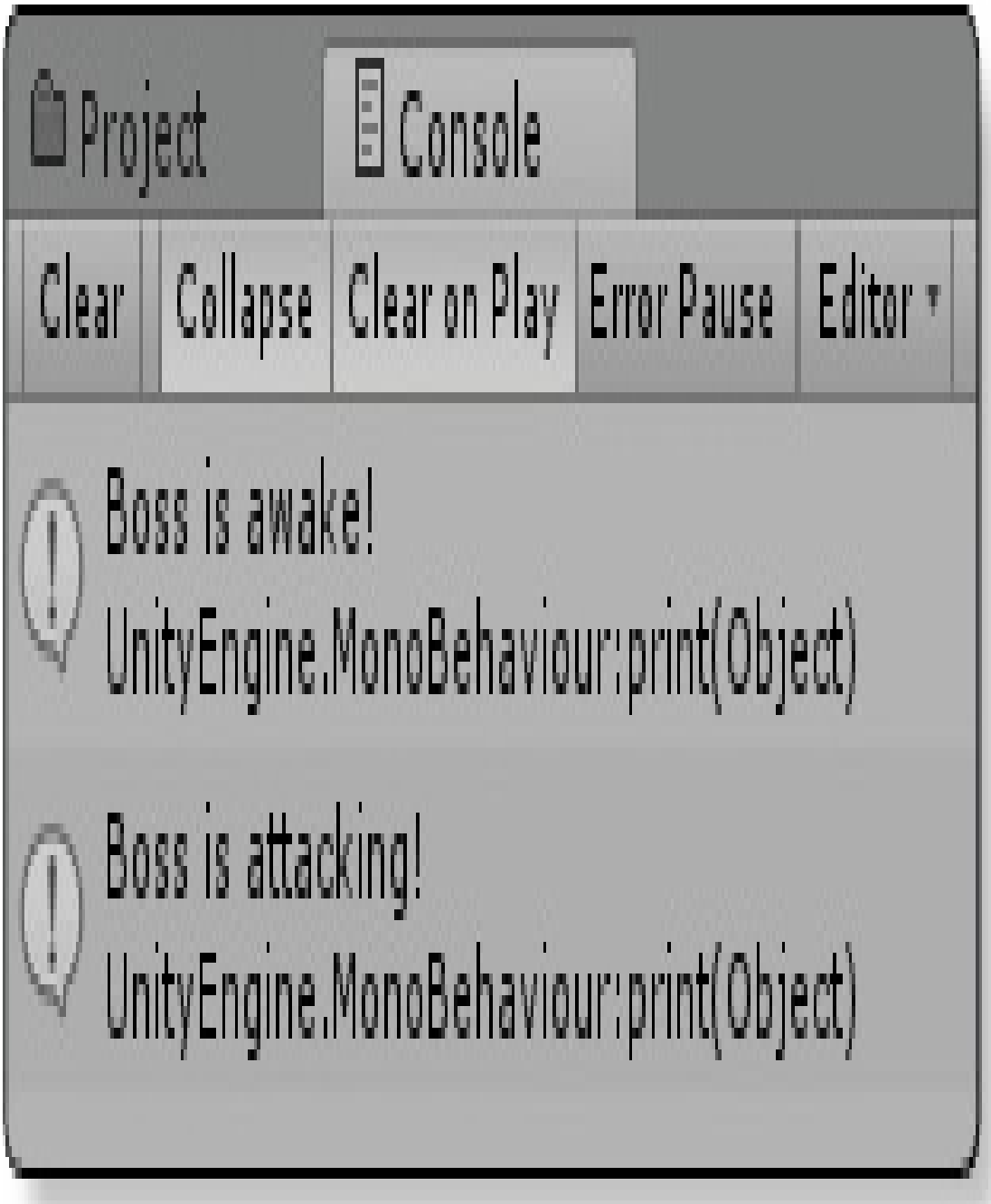
Boss Controller (Script)
Script
BossController
Boss Awake
In Battle
Attacking
Idle Timer
8.742701
Idle Wait Time
10

Previsualización de funcionalidad de ataque

Boss Controller (Script)
Script
BossController
Boss Awake
In Battle
Attacking
Idle Timer
0
Idle Wait Time
10

Fase de ataque activa

¡Se imprime en la consola “Boss is attacking!”


Project
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
Boss is awake!
UnityEngine.MonoBehaviour:print(Object)
Boss is attacking!
UnityEngine.MonoBehaviour:print(Object)

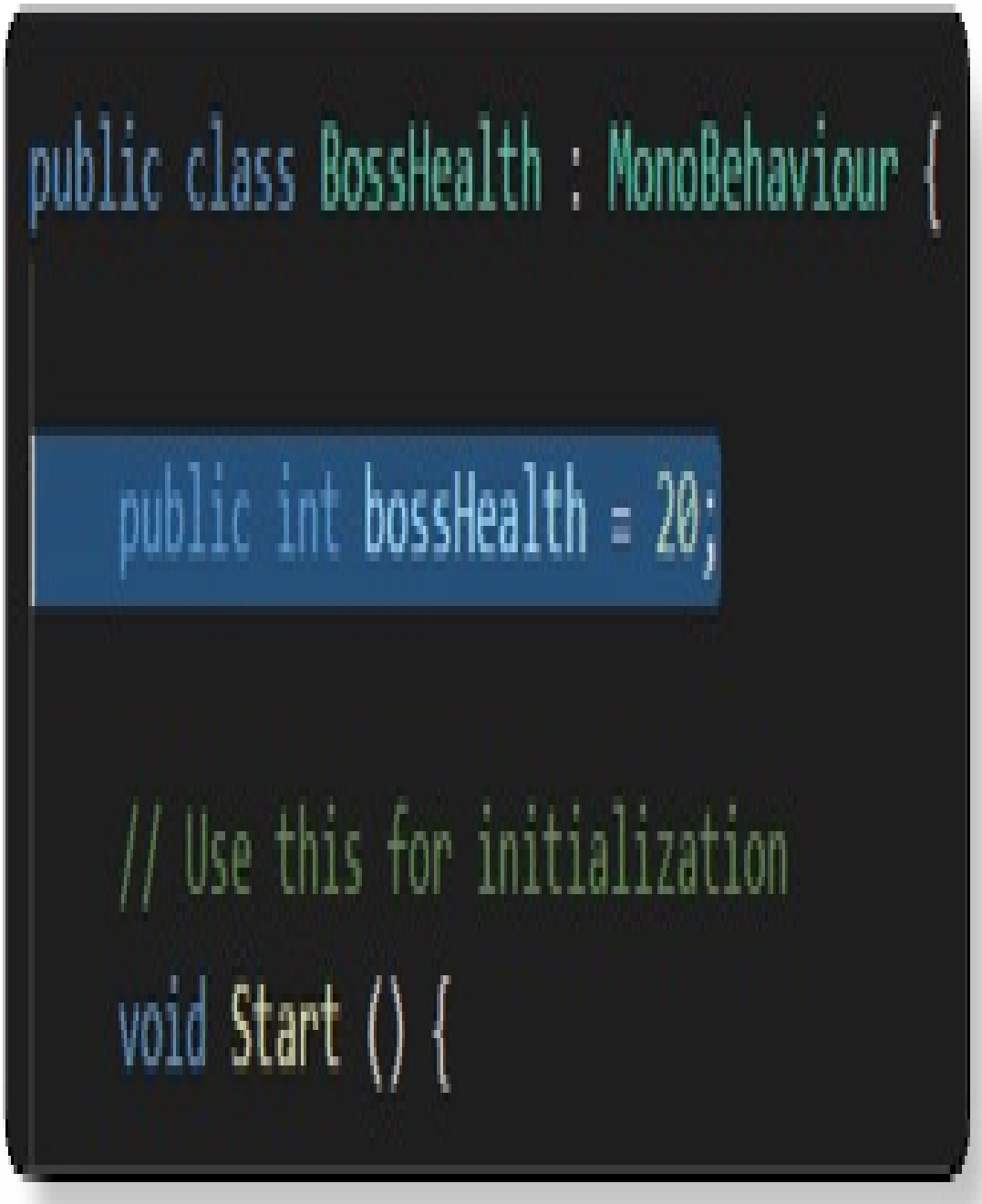
Impresión en consola de estado de ataque

A continuación, crearemos un script de salud para el jefe.

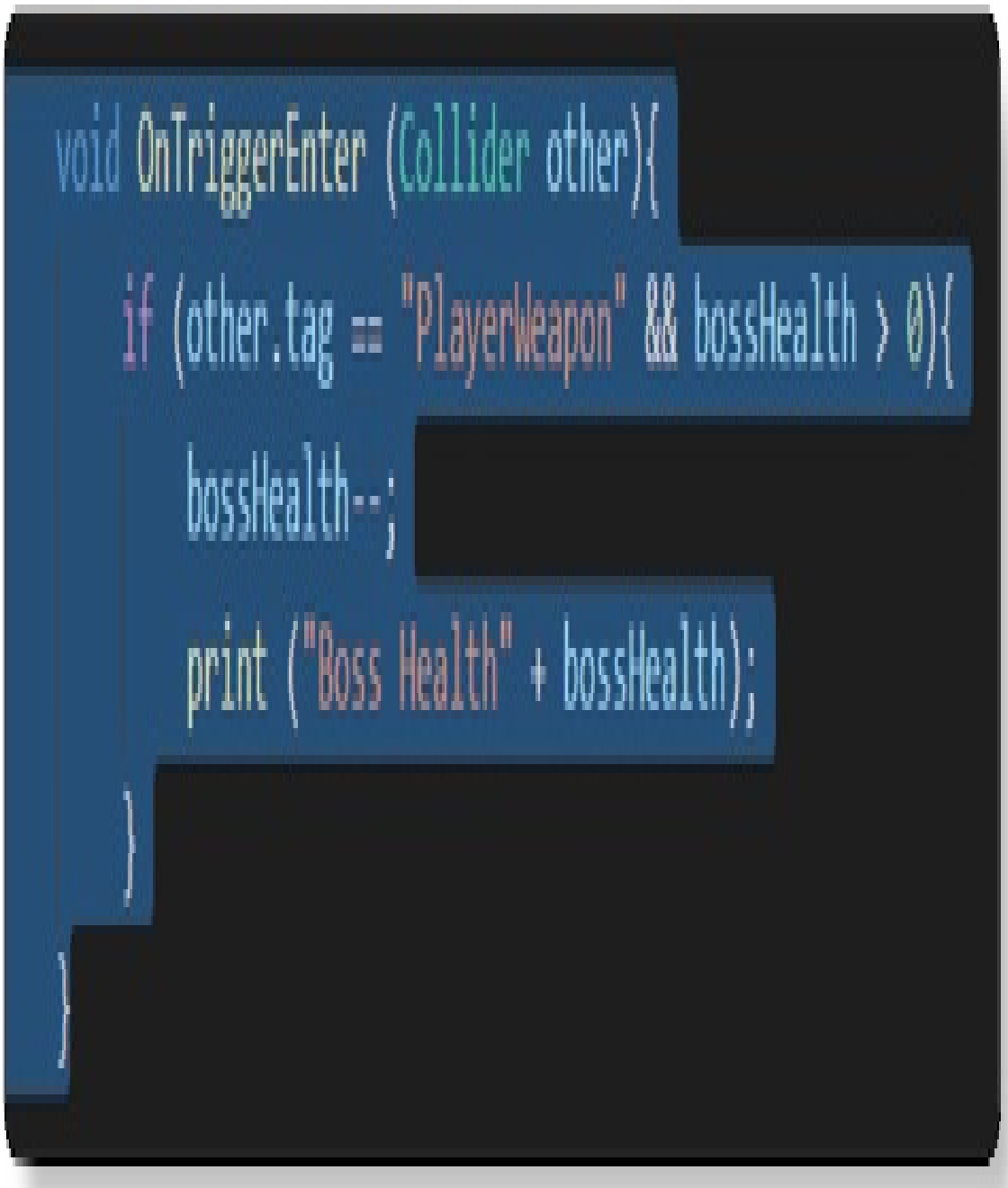
Haga un nuevo C#Script y nómbrelo “BossHealth”.

Asigne el nuevo script al objeto “HeadTrigger”.

Abra el script y escriba el siguiente código para hacer que cada vez que sea colisionado por el proyectil del “Player” baje su salud.


public class BossHealth : MonoBehaviour {
public int bossHealth = 20;
// Use this for initialization
void Start () {

Creación de variable


void OnTriggerEnter (Collider other){
if (other.tag == "PlayerWeapon" && bossHealth > 0){
bossHealth--;
print ("Boss Health" + bossHealth);
}
}

Función OnTriggerEnter

Previsualice el juego para comprobar que cada vez que el proyectil del “Player” acierta en la cabeza del jefe, es decir en su “HeadTrigger”, la salud de éste comienza a aminorar.

A continuación, debemos de crear un sistema para controlar los ataques conforme a tiempos.

Vaya al script “BossController”, escriba e inicialícelas variables que se muestran.

```
private BossHealth bossHealth;
public float attackTimer = 0.0f;
public float attackWaitTime = 2.0f;

// Use this for initialization
void Start () {
    anim = GetComponent<Animator>();
    bossHealth = GetComponent<BossHealth>();
}
```

Creación e inicialización de variables

Vaya a la función “Update” y asigne las siguientes líneas de código para activar la fase de ataque.

```
if(inBattle){
    if(!attacking){
        idleTimer += Time.deltaTime;
    } else {
        idleTimer = 0.0f;
        attackTimer += Time.deltaTime;
        if (attackTimer >= attackWaitTime){
            attacking = false;
            attackTimer = 0.0f;
            print ("Boss Smash");
        }
    }
    if(idleTimer >= idleWaitTime){
        print ("Boss is attacking!");
```

Código para activar la fase de ataque

Vaya a Unity, seleccione el objeto “Boss”, previsualice el juego para comprobar que cuando se activa la casilla de ataque:

Se activa el contador de “AttackTimer”

Boss Controller (Script)
Script
BossController
Boss Awake
In Battle
Attacking
Idle Timer
0
Idle Wait Time
5
Attack Timer
1.584062
Attack Wait Time
2

Previsualización de fase de ataque

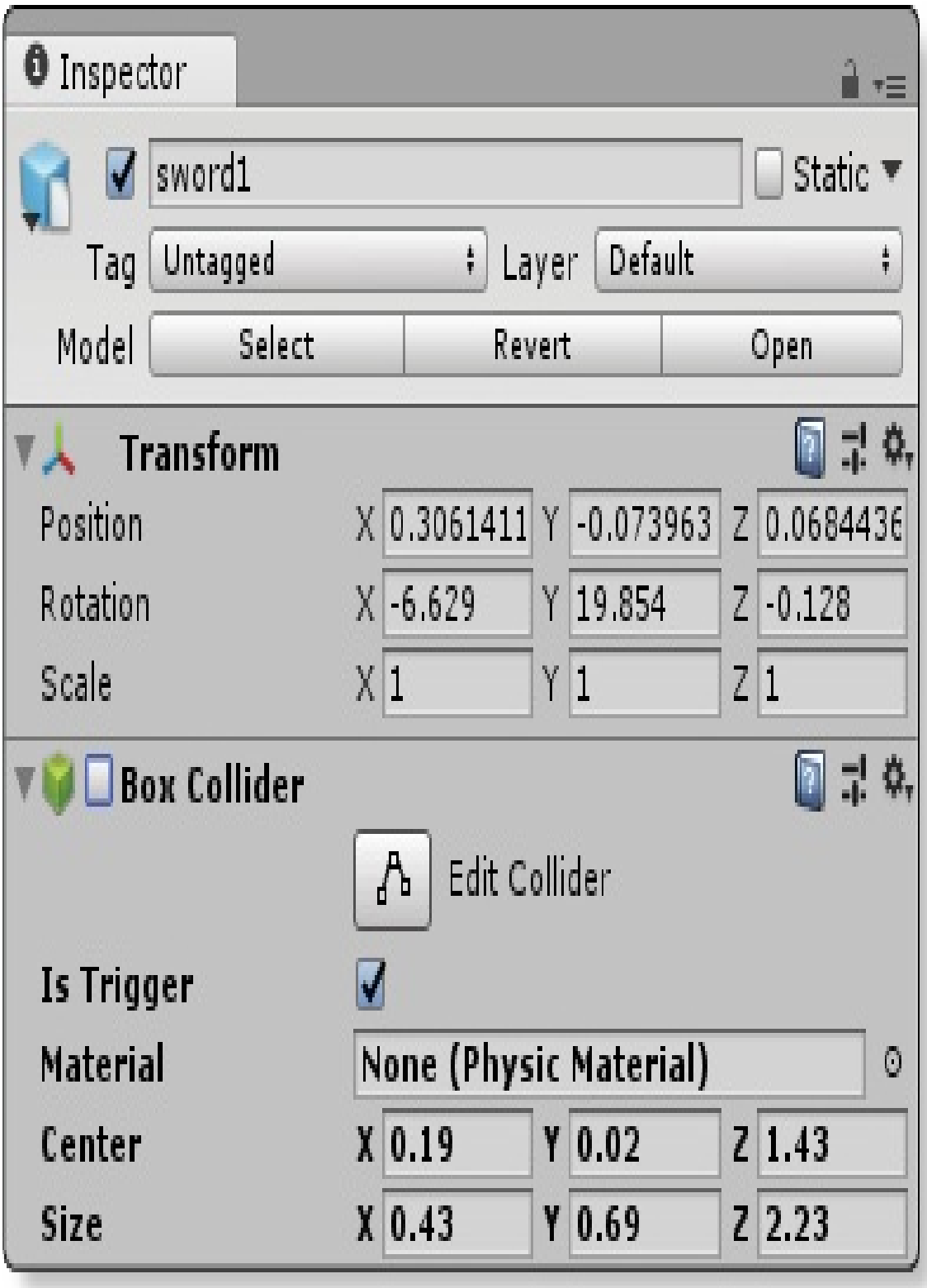
Cuando llega al tiempo relativo de “AttackWaitTime” se desactiva la casilla “Attacking” y comienza nuevamente el ciclo

Boss Controller (Script)
Script
BossController
Boss Awake
In Battle
Attacking
Idle Timer
1.709349
Idle Wait Time
5
Attack Timer
0
Attack Wait Time
2

Previsualización determino de fase de ataque

A continuación, necesitamos deshabilitar el Collider del arma del jefe y habilitarlo únicamente cuando se encuentre en su fase de ataque.

Seleccione el objeto “Sword” de la jerarquía del “Boss” y desactive su componente “BoxCollider”.


Inspector
sword1
Static
Tag
Untagged
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 0.3061411
Y -0.073963
Z 0.0684436
Rotation
X -6.629
Y 19.854
Z -0.128
Scale
X 1
Y 1
Z 1
Box Collider
Edit Collider
Is Trigger
Material
None (Physic Material)
Center
X 0.19
Y 0.02
Z 1.43
Size
X 0.43
Y 0.69
Z 2.23

Deshabilitación de BoxCollider de espada

Vaya al script “BossController”, escriba e inicialice la siguiente variable para llamar el “BoxCollider” de la espada.

```
public float attackWaitTime = 2.0f;


private BoxCollider swordTrigger;


// Use this for initialization
void Start () {
    anim = GetComponent<Animator>();
    bossHealth = GetComponent<BossHealth>();
    swordTrigger = GameObject.Find ("Boss").GetComponentInChildren<BoxCollider>();
}
```

Creación e inicialización de variables

Asigne las siguientes líneas de código para reactivar el “Collider” de la espada cuando nuestro “Boss” entre en la fase de ataque.

```
if(inBattle){
    if(!attacking){
        idleTimer += Time.deltaTime;
    } else {
        idleTimer = 0.0f;
        attackTimer += Time.deltaTime;
        if (attackTimer >= attackWaitTime){
            attacking = false;
            attackTimer = 0.0f;
            print ("Boss Smash");
            swordTrigger.enabled = true;
            print ("SwordTrigger is enabled");
        }
    }
```

Habilitación de BoxCollider de la espada

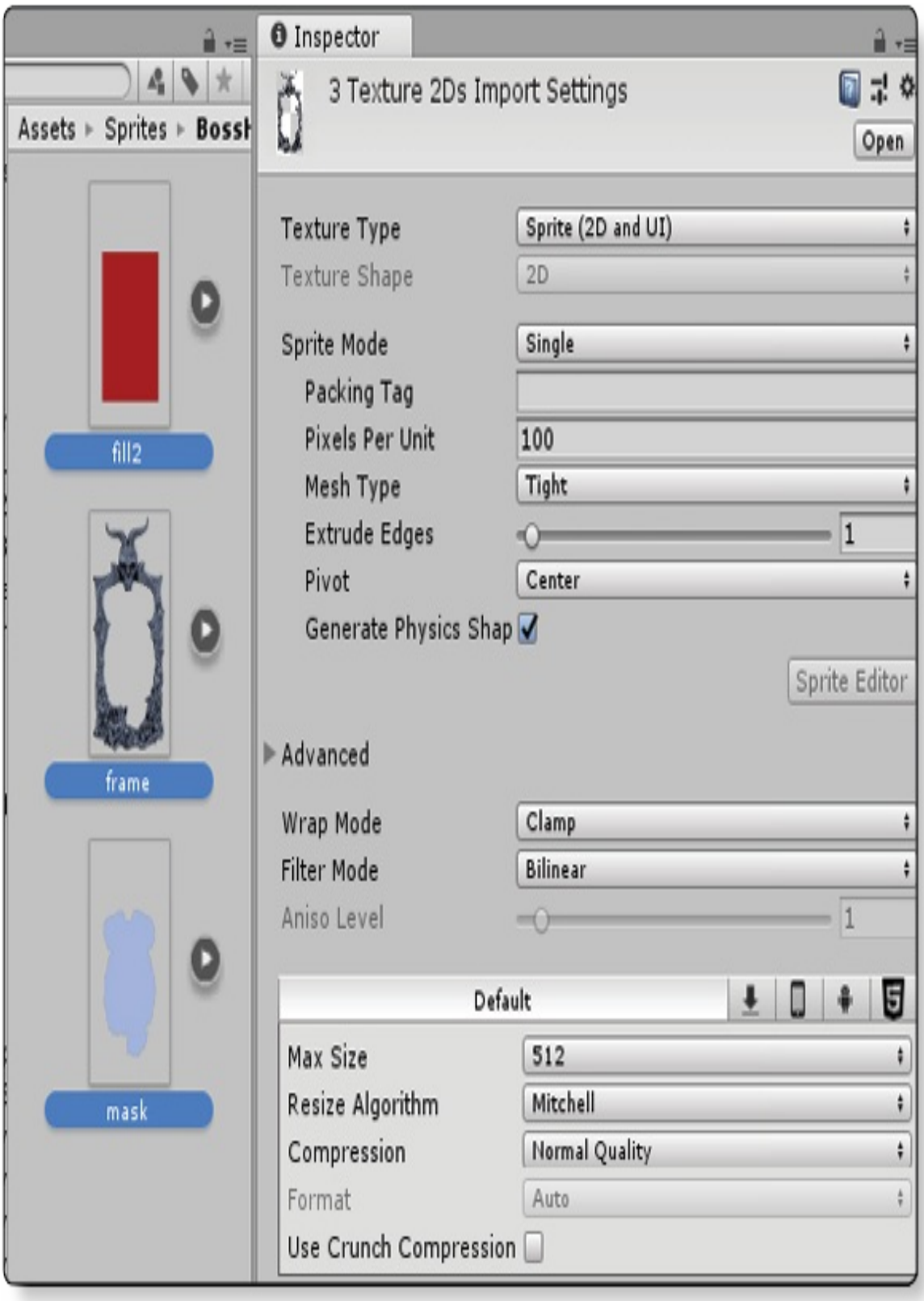
Previsualice el juego para observar que al entrar en la fase de ataque se activa el “BoxCollider” de la espada.

Creación y programación de la barra de salud del jefe

En este apartado crearemos elementos GUI para el indicador de vida a partir de elementos gráficos externos y los ligaremos al sistema de vida para que visualmente podamos observar su funcionamiento.

Vaya a la carpeta de Sprites y haga una nueva carpeta y nómbrela “BossHealthSprites”.

Importe de la carpeta de “resources” del capítulo los gráficos que se muestran y conviértalos en textura “Sprite (2D and UI).


Inspector
Assets ▸ Sprites ▸ Bossh
fill2
frame
mask
3 Texture 2Ds Import Settings
Open
Texture Type
Sprite (2D and UI)
Texture Shape
2D
Sprite Mode
Single
Packing Tag
Pixels Per Unit
100
Mesh Type
Tight
Extrude Edges
1
Pivot
Center
Generate Physics Shap
Sprite Editor
Advanced
Wrap Mode
Clamp
Filter Mode
Bilinear
Aniso Level
1
Default
Max Size
512
Resize Algorithm
Mitchell
Compression
Normal Quality
Format
Auto
Use Crunch Compression

Importación de los elementos gráficos

En “Hierarchy”, clic derecho en el objeto “GUI”, haga un nuevo “EmptyObject” y nómbrelo “BossHealthBar”.

GUI
HealthBar
LifeSystem
GameOverScreen
BossHealthBar

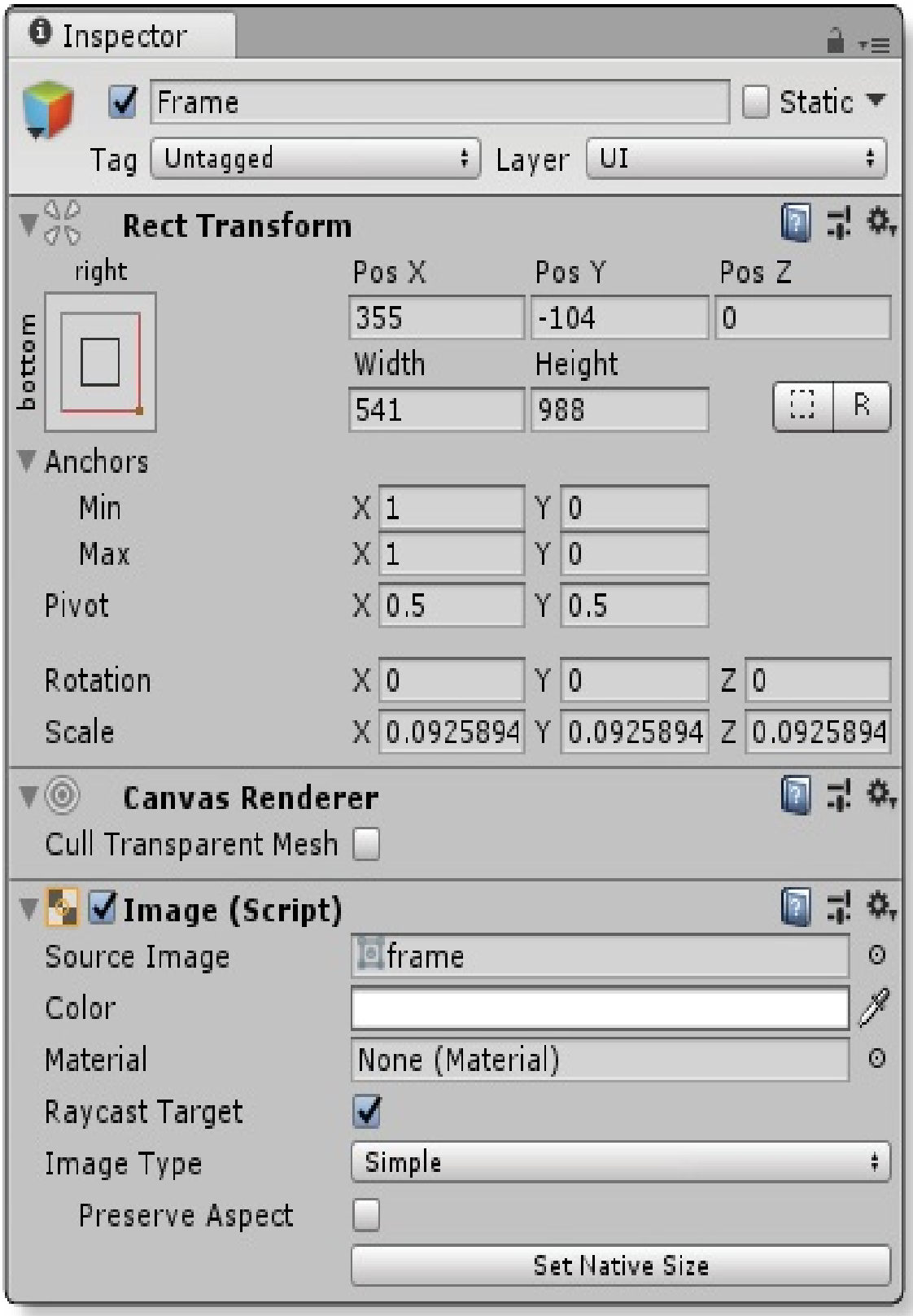
Creación de BossHealthBar

Con “BossHealthBar” seleccionado, asigne clic derecho y haga un nuevo “UI/ Image” y renómbrela “Frame”.

GUI
HealthBar
LifeSystem
GameOverScreen
BossHealthBar
Frame

Creación de Frame

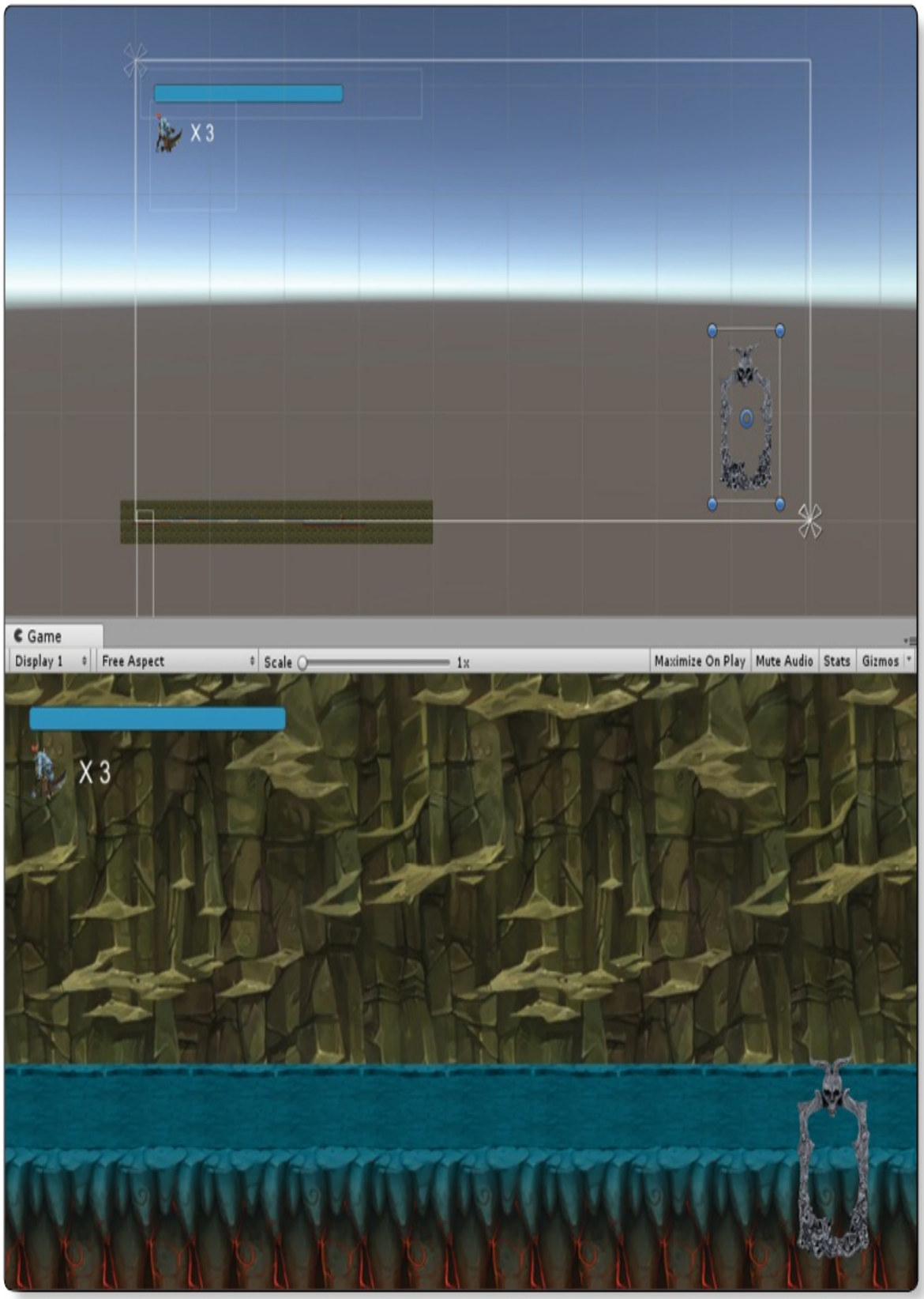
Con “Frame” seleccionado asigne la imagen “frame” en “Inspector”, asigne los valores mostrados y coloque la imagen en la pantalla como se indica.


Inspector
Frame
Static
Tag
Untagged
Layer
UI
Rect Transform
right
bottom
Pos X
Pos Y
Pos Z
355
-104
0
Width
Height
541
988
R
Anchors
Min
X 1
Y 0
Max
X 1
Y 0
Pivot
X 0.5
Y 0.5
Rotation
X 0
Y 0
Z 0
Scale
X 0.0925894
Y 0.0925894
Z 0.0925894
Canvas Renderer
Cull Transparent Mesh
Image (Script)
Source Image
frame
Color
Material
None (Material)
Raycast Target
Image Type
Simple
Preserve Aspect
Set Native Size

Valores para Frame

NOTA:

Clic en el botón “Set Native Size” permitirá visualizar el tamaño natural de la imagen.


X 3
Game
Display 1
Free Aspect
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
X 3

Correcta colocación de los elementos en la Interfase

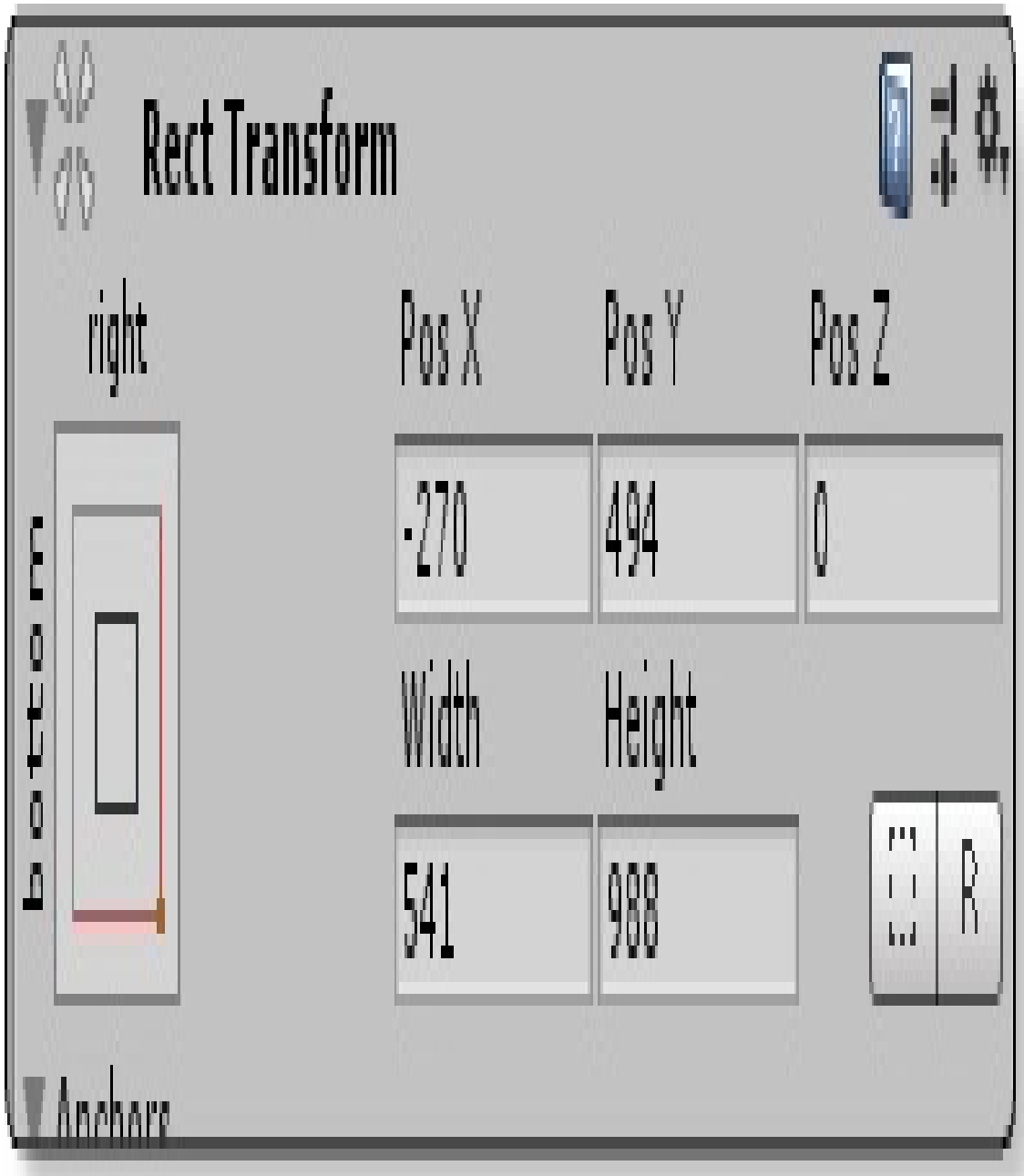
Con “Frame” seleccionado, clic derecho para crear otro “UI /Image” y renómbrelo “Mask”.

GUI
HealthBar
LifeSystem
GameOverScreen
BossHealthBar
Frame
Mask

Creación de Mask

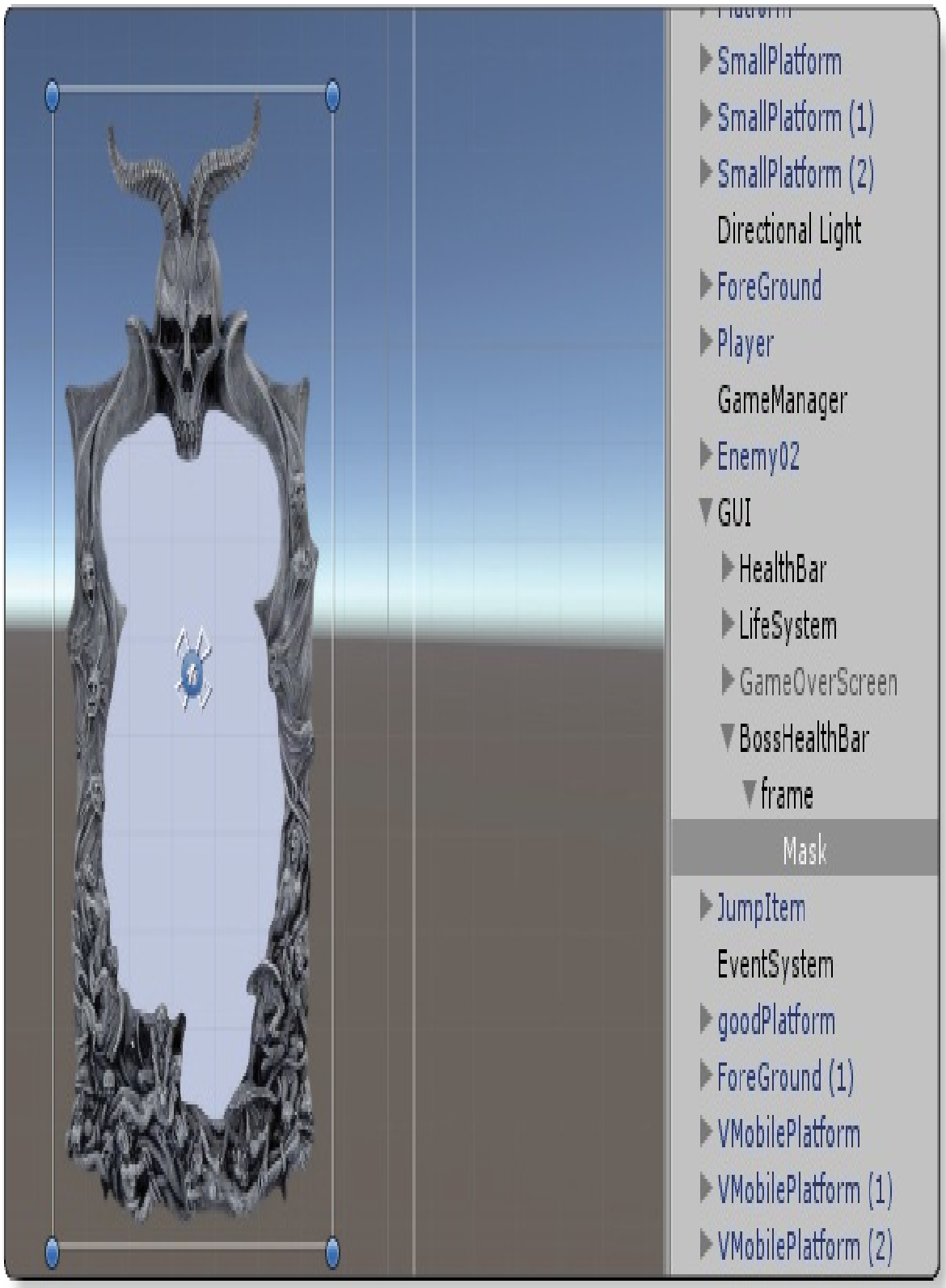
Necesitamos que los valores de transformación del objeto “Mask” sean los mismos que los del objeto “Frame”.

Seleccione el objeto “Frame” y en Inspector copie sus valores de transformación.


Rect Transform
right
Pos X
Pos Y
Pos Z
-270
494
0
Width
Height
541
988
R
bottom
Anchors

Valores de transformación de Frame

Seleccione el objeto “Mask” y pegue los valores de transformación de “Frame”.


SmallPlatform
SmallPlatform (1)
SmallPlatform (2)
Directional Light
ForeGround
Player
GameManager
Enemy02
GUI
HealthBar
LifeSystem
GameOverScreen
BossHealthBar
frame
Mask
JumpItem
EventSystem
goodPlatform
ForeGround (1)
VMobilePlatform
VMobilePlatform (1)
VMobilePlatform (2)

Correcta posición de Mask

Con “Mask” seleccionado, haga otro nuevo elemento “UI / Image”, renómbrelo “Fill” y asegúrese que se encuentre posicionado con los mismos valores de transformación que “Mask” y asignando su grafico correspondiente.

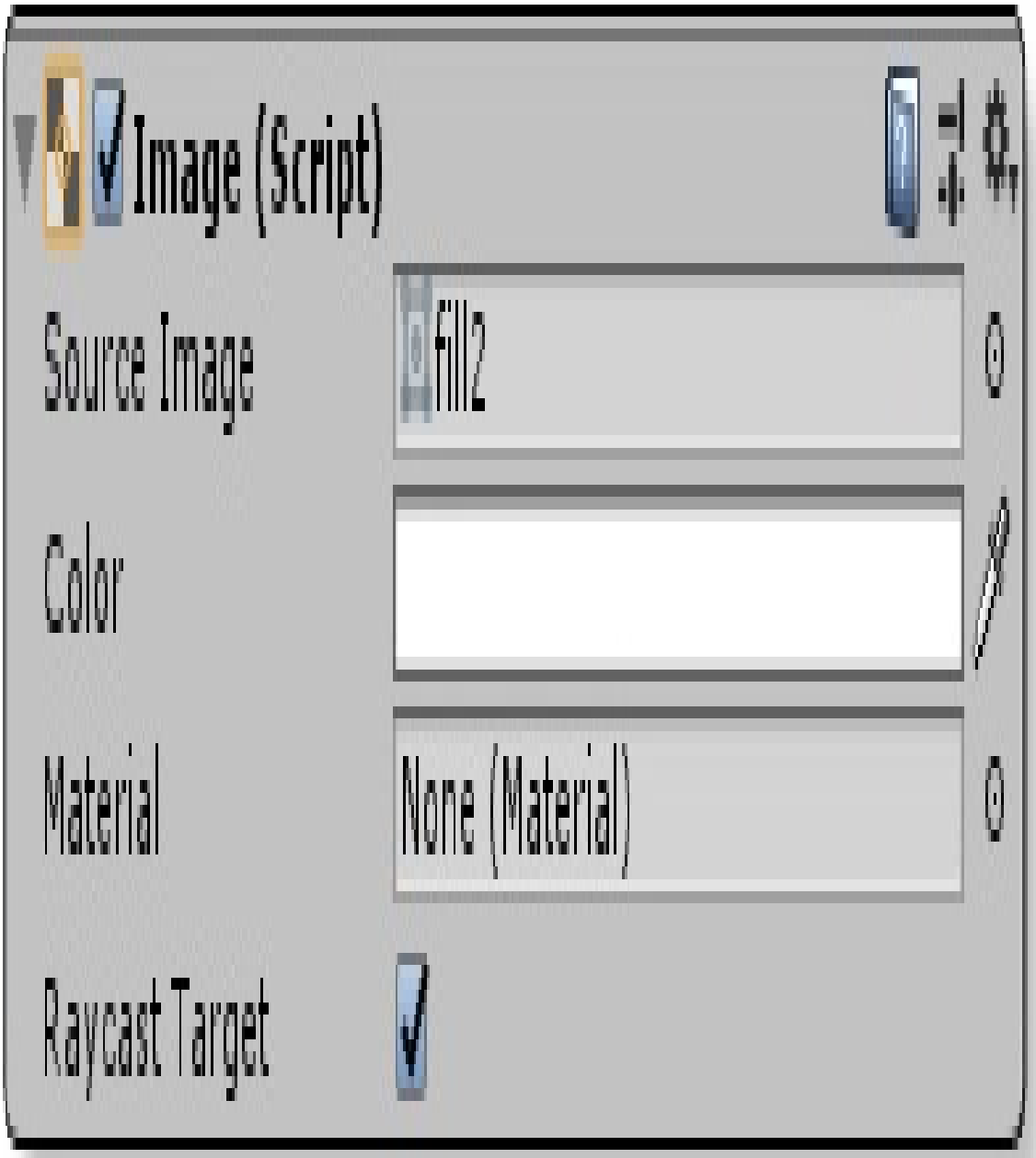
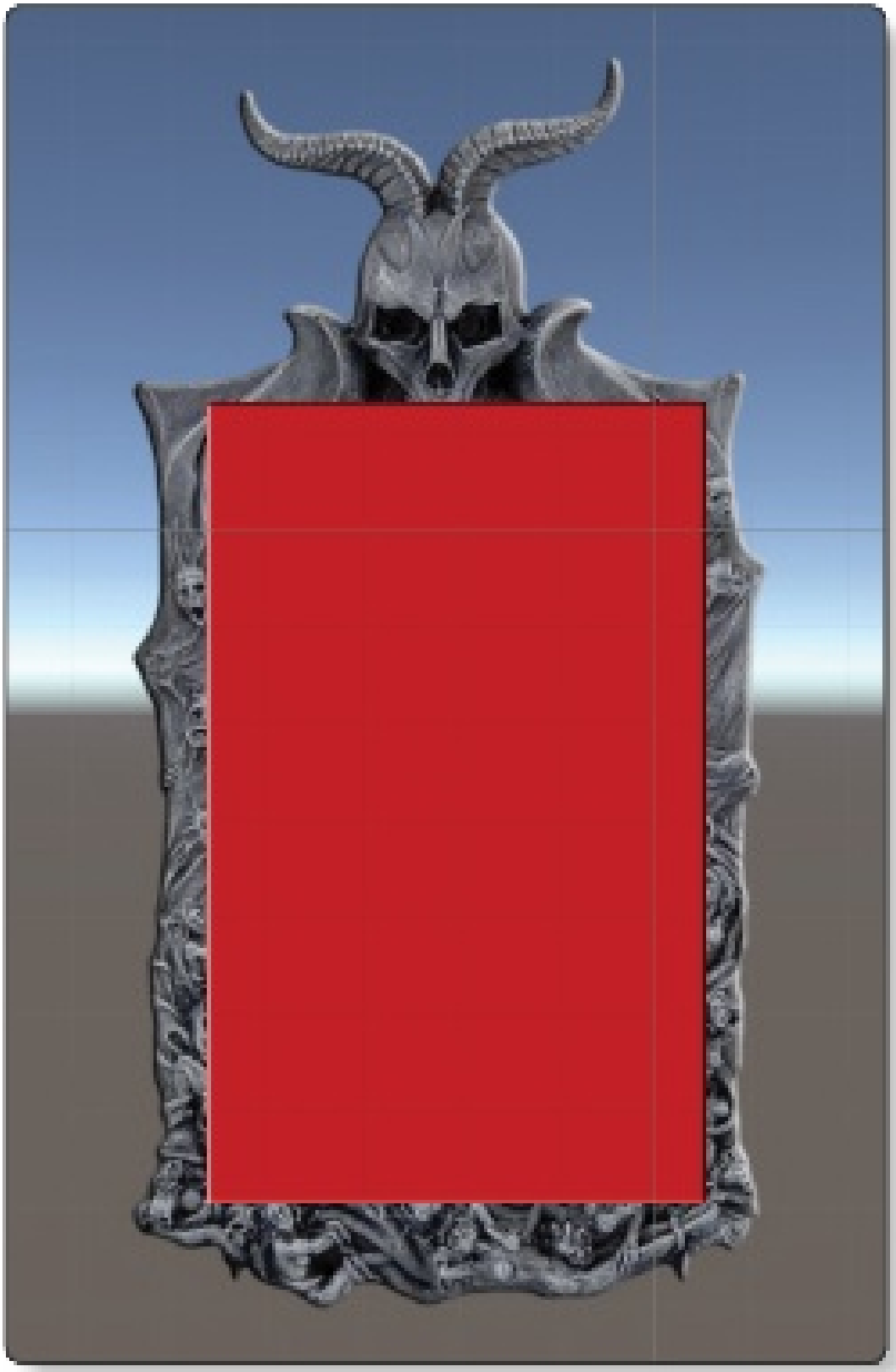


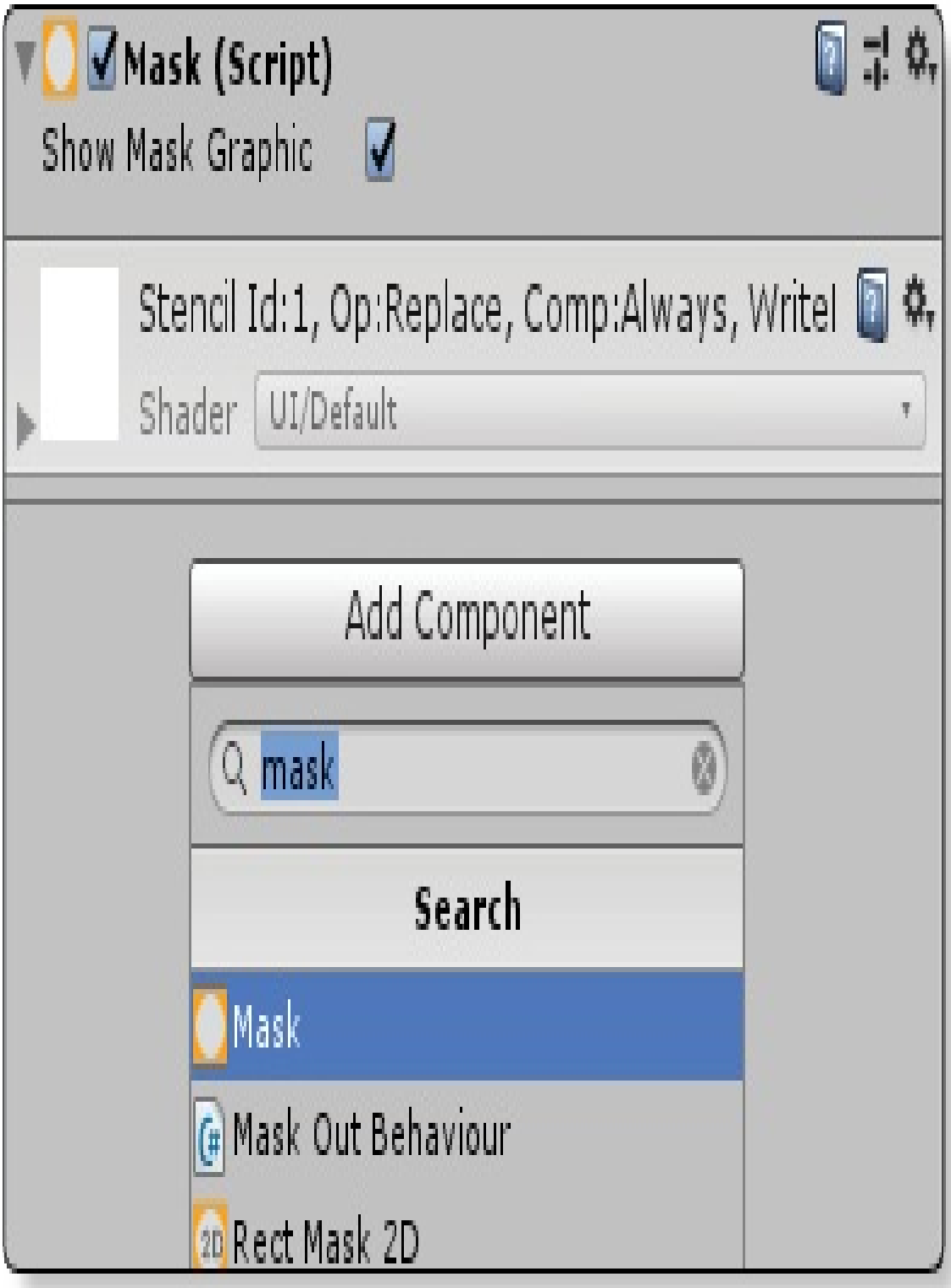
Image (Script)
Source Image
Fill2
Color
Material
None (Material)
Raycast Target

Asignación de elemento para Fill

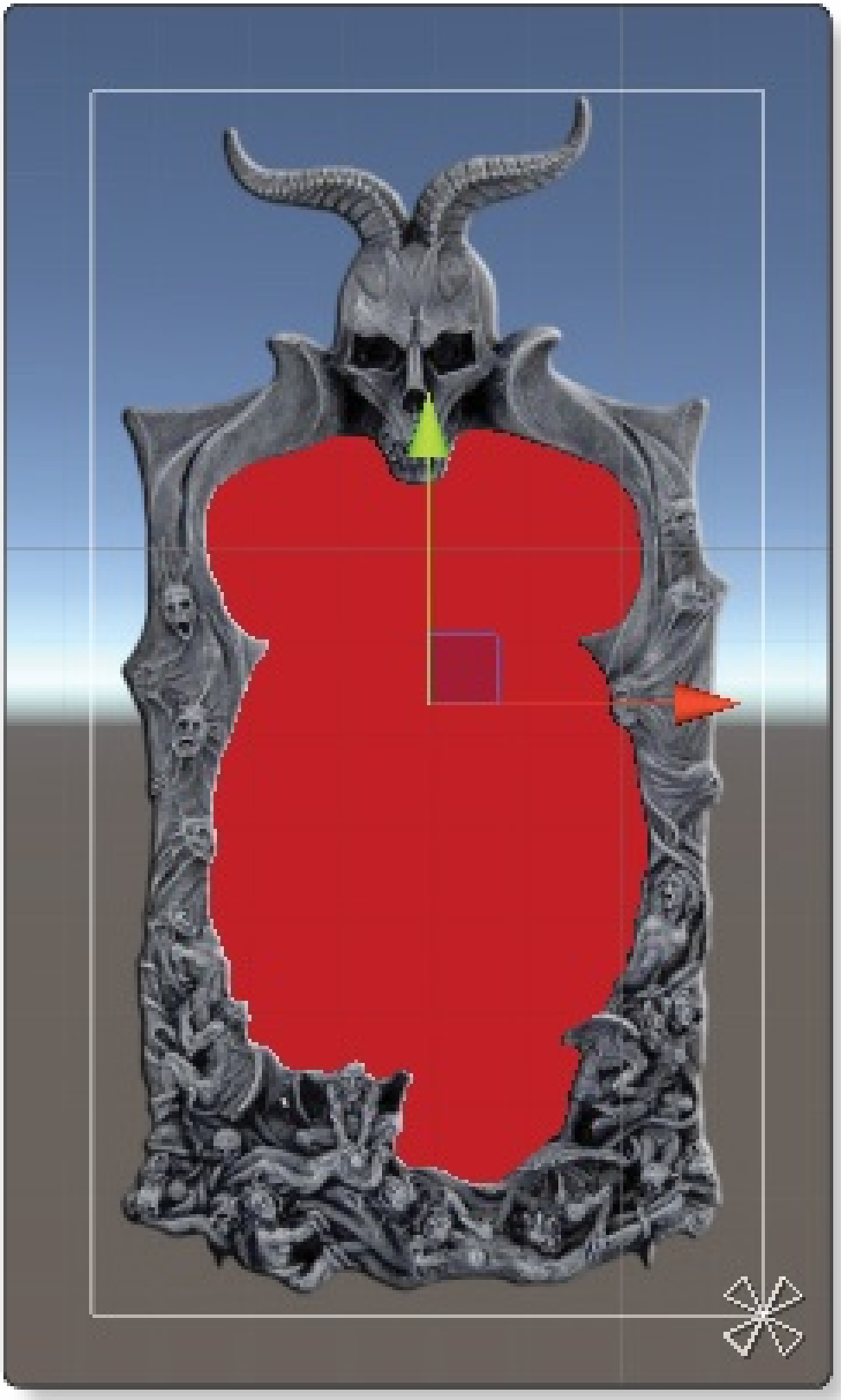


Previsualización de Fill

Seleccione el objeto “Mask”, asigne un componente “mask”.


Mask (Script)
Show Mask Graphic
Stencil Id:1, Op:Replace, Comp:Always, Writel
Shader
UI/Default
Add Component
mask
Search
Mask
Mask Out Behaviour
Rect Mask 2D

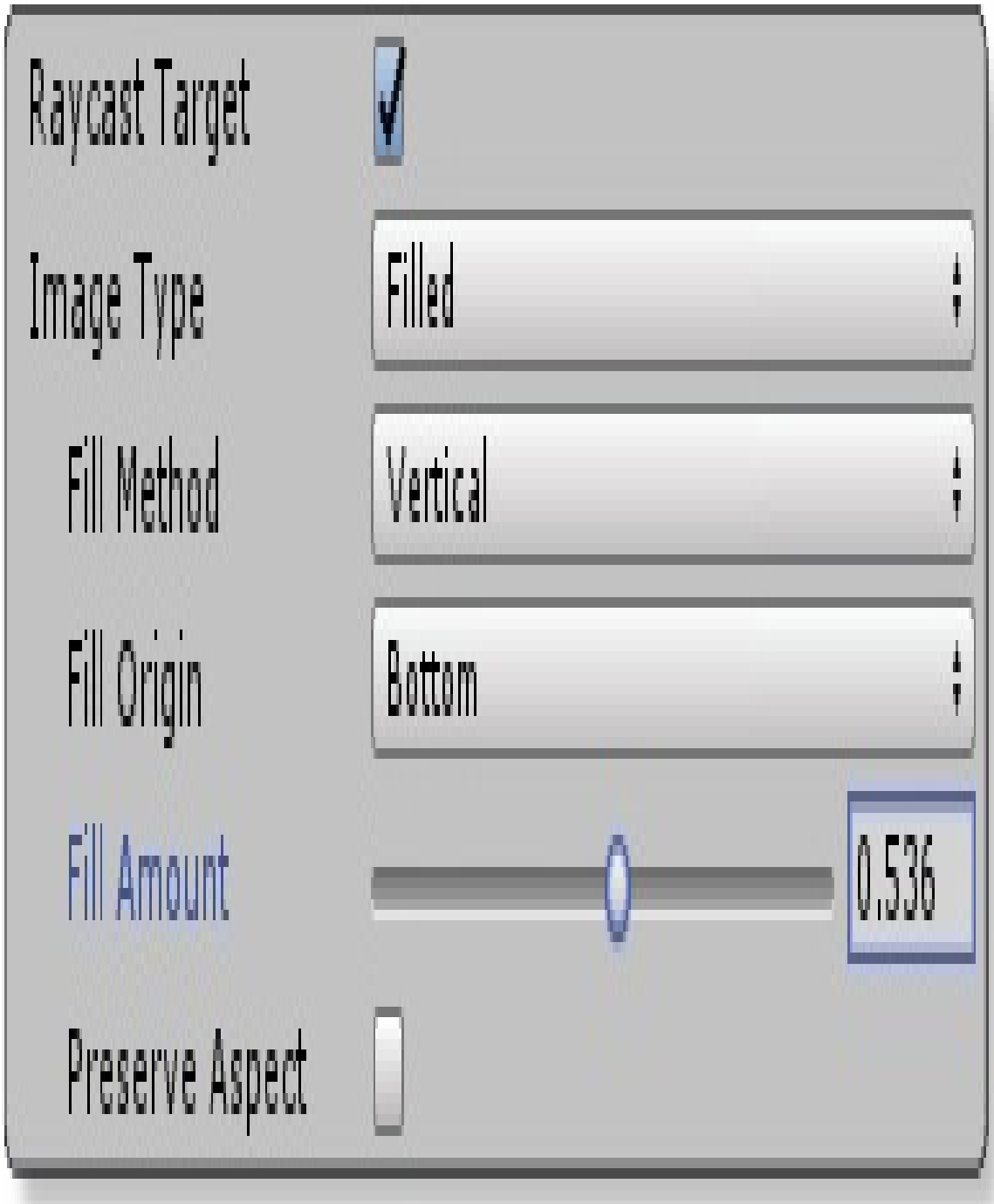
Componente Mask



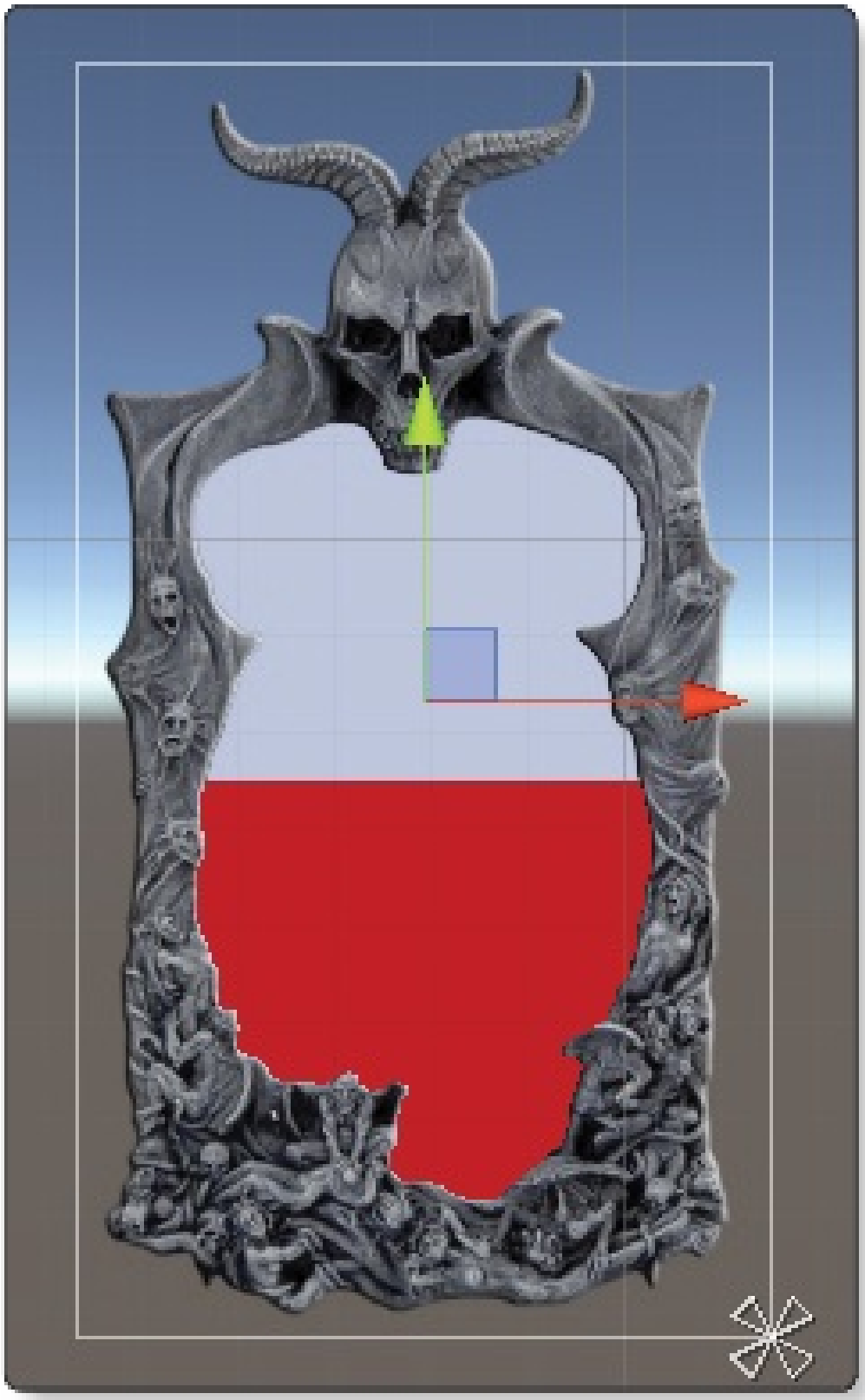
Previsualización de la máscara

Queremos que el gráfico “Fill” vaya bajando horizontalmente.

Seleccione “Fill”, asigne los valores que se muestran en su componente “Image” y mueva su indicador de “Fill Amount” para observar como va bajando (al finalizar la previsualización devuelva su valor a 1).


Raycast Target
Image Type
Filled
Fill Method
Vertical
Fill Origin
Bottom
Fill Amount
0.536
Preserve Aspect

Valores de la imagen de Fill



Previsualización del funcionamiento de Fill Amount

A continuación, necesitamos enlazar el nivel de “Amount” de “Fill” a nuestro script.

Vaya al folder de los scripts del “Boss”, haga un nuevo C#Script y nómbrelo “BossHBar”.

Asigne el script “BossHBar” al objeto “BossHealthBar”.

Abra el script “BossHBar” y escriba el siguiente código:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class BossHBar : MonoBehaviour {

    [SerializeField] private float fillAmount;
    [SerializeField] private Image fill;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        HandleBar();
    }

    private void HandleBar (){
        fill.fillAmount = fillAmount;
    }
}
```

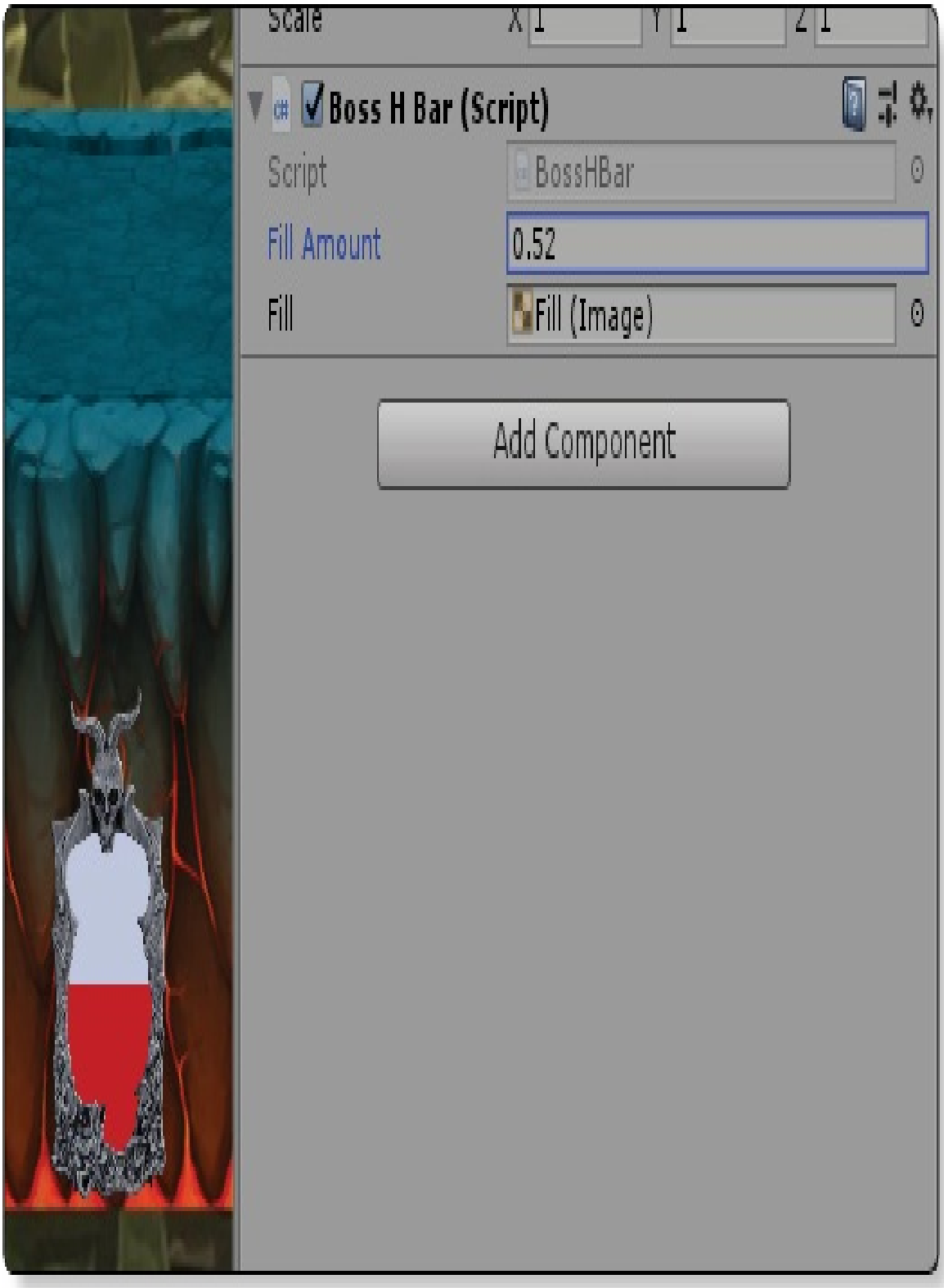
Script BossHBar

Vaya a Unity y asigne los valores que se muestran en el script.

Boss H Bar (Script)
Script
BossHBar
Fill Amount
1
Fill
Fill (Image)

Asignación de valores en script

Previsualice el juego para observar que “Fill Amount” del script ahora está ligado al “Fill” de la barra de salud del “Boss”.


Scale
X 1
Y 1
Z 1
Boss H Bar (Script)
Script
BossHBar
Fill Amount
0.52
Fill
Fill (Image)
Add Component

Previsualización de funcionalidad del script

A continuación, debemos de enlazar la salud de nuestro “Boss” al “FillAmount” de nuestra imagen.

Para ello debemos de tomar en cuenta los siguientes aspectos:

La salud total de nuestro jefe es de 20

El indicador de salud UI máximo es de 1

Debemos de realizar una operación en el código la cual nos haga la relación entre la salud actual y el indicador de salud UI:

fillAmount = (currentHealth / MaxHealth)

Vaya al script “BossHBar”, escriba e inicialice las variables que se muestran.

```
[SerializeField] private Image fill;


public GameObject headTrigger;
private BossHealth health;
public int currentHealth;
public float MaxHealth;


// Use this for initialization
void Start () {
    health = headTrigger.GetComponent<BossHealth>();
}
```

Creación e inicialización de variables

Vaya a la función “HandleBar” y añada el código:

```
private void HandleBar (){
    fill.fillAmount = fillAmount;


    if (currentHealth >= 0) {
        currentHealth = health.bossHealth;
        fillAmount = (currentHealth / MaxHealth);
    }
}
```

Código funcional

Vaya a Unity, seleccione el objeto “BossHealthBar” y asigne los siguientes valores en el script.

Boss H Bar (Script)
Script
BossHBar
Fill Amount
1
Fill
Fill (Image)
Head Trigger
HeadTrigger
Current Health
0
Max Health
20

Asignación de valores para script

Previsualice el juego para observar como baja la barra de vida del jefe a medida que su “HeadTrigger” es colisionado por un proyectil de nuestro Player.

Previsualización de funcionalidad de UI

Queremos que la barra de vida de nuestro jefe vaya bajando con un movimiento transitorio más suave, para ello utilizaremos el “Lerp”.

Vaya al script “BossHBar” y asigne el siguiente código:

[SerializeField] private float fillAmount;
[SerializeField] private Image fill;
[SerializeField] private float lerpSpeed;

Creación de variable

```
private void HandleBar (){
    fill.fillAmount = Mathf.Lerp
    (fill.fillAmount, fillAmount, Time.deltaTime * lerpSpeed );


    if (currentHealth >= 0) {
        currentHealth = health.bossHealth;
        fillAmount = (currentHealth / MaxHealth);
    }
}
```

Edición de código para función Lerp

Vaya a Unity y asigne el valor de 2 para “LerpSpeed”.

Previsualice el juego para observar cómo va disminuyendo más suavemente la barra de vida de nuestro jefe.

PROBLEMA: El indicador de salud del jefe aparece durante todo el juego.

SOLUCIÓN: Vaya al script “BossController” y escriba las siguientes líneas de código para desactivar la barra y únicamente activarla cuando inicie la batalla final.

```
private BoxCollider swordTrigger;
public GameObject bossHealthBar;


// Use this for initialization
void Start () {
    anim = GetComponent<Animator>();
    bossHealth = GetComponent<BossHealth>();
    swordTrigger = GameObject.Find ("Boss").G
    bossHealthBar.SetActive (false);
}
```

Creación e inicialización de variables

```
void Update () {


    if (bossAwake){
        print ("Boss is awake!");
        anim.SetBool("bossAwake", true);
        bossHealthBar.SetActive (true);
```

Código para activar elemento

Vaya a Unity, seleccione el objeto “Boss” y asigne en su script “BossController” el objeto “BossHealthBar” como se muestra

Attack Timer
0
Attack Wait Time
4
Boss Health Bar
BossHealthBar
0

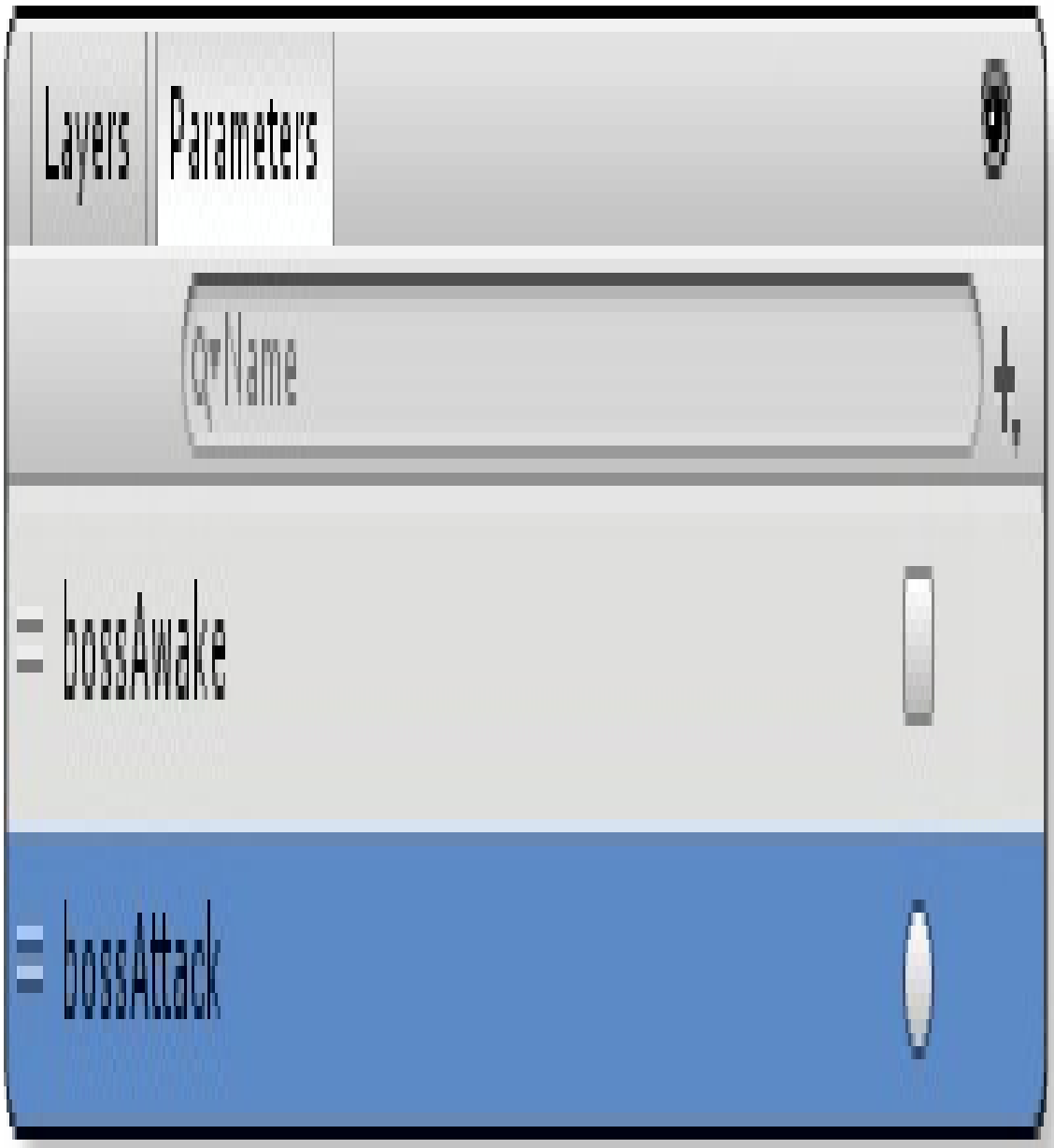
Programación del primer ataque del jefe y reseteo de comportamientos

En este apartado programaremos el primer ataque del jefe asignando su animación de ataque.

A su vez necesitamos crear todas las condiciones de “Reset” de nuestro Jefe cuando el “Player” muera y regrese nuevamente a la batalla final.

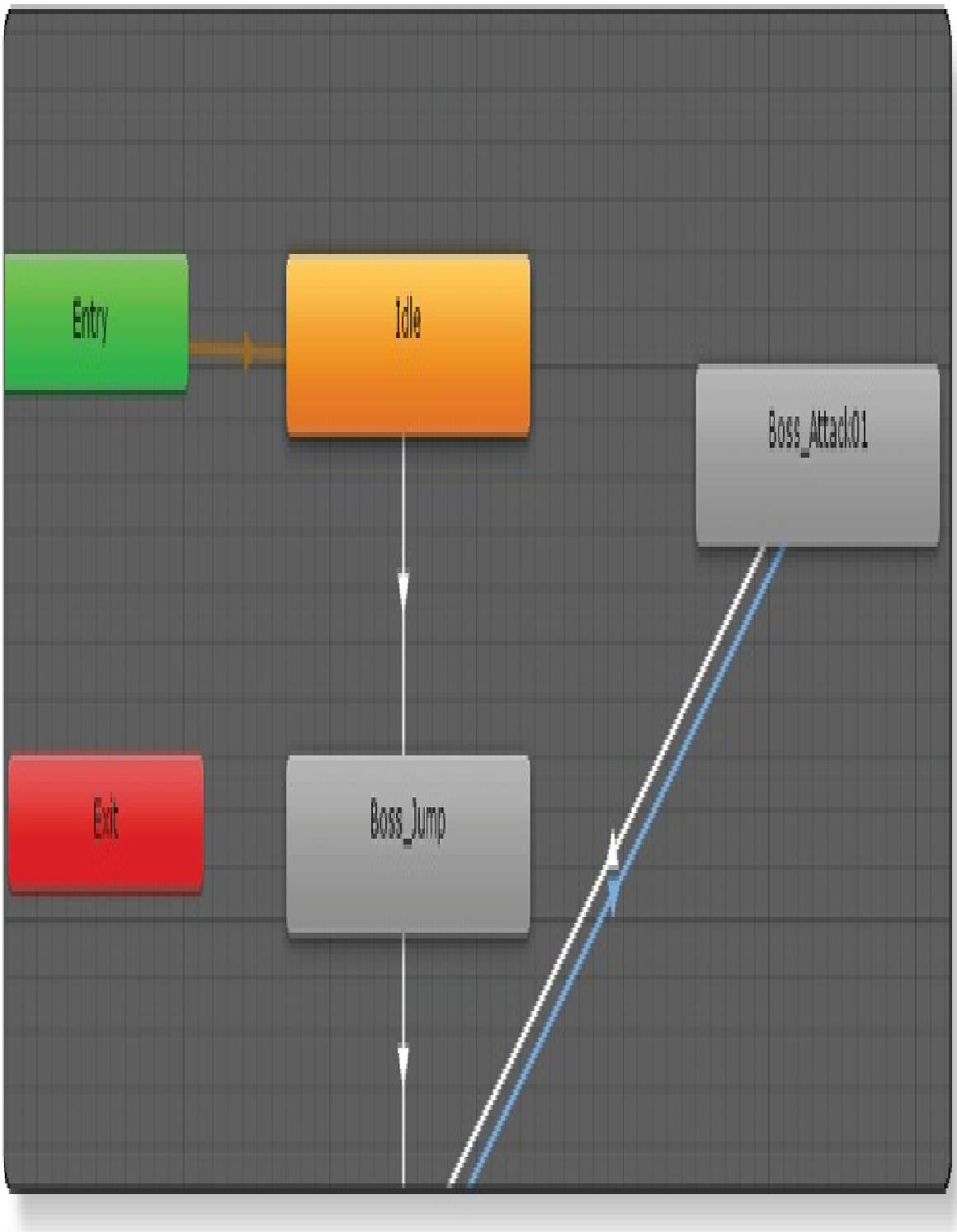
Seleccione el objeto “Sword” y asigne el tag “Weapon” para que le haga daño al “Player”.

Seleccione el objeto “Boss”, vaya a la ventana “Animator”, haga un nuevo parámetro tipo “Trigger” y nómbrelo “bossAttack”.

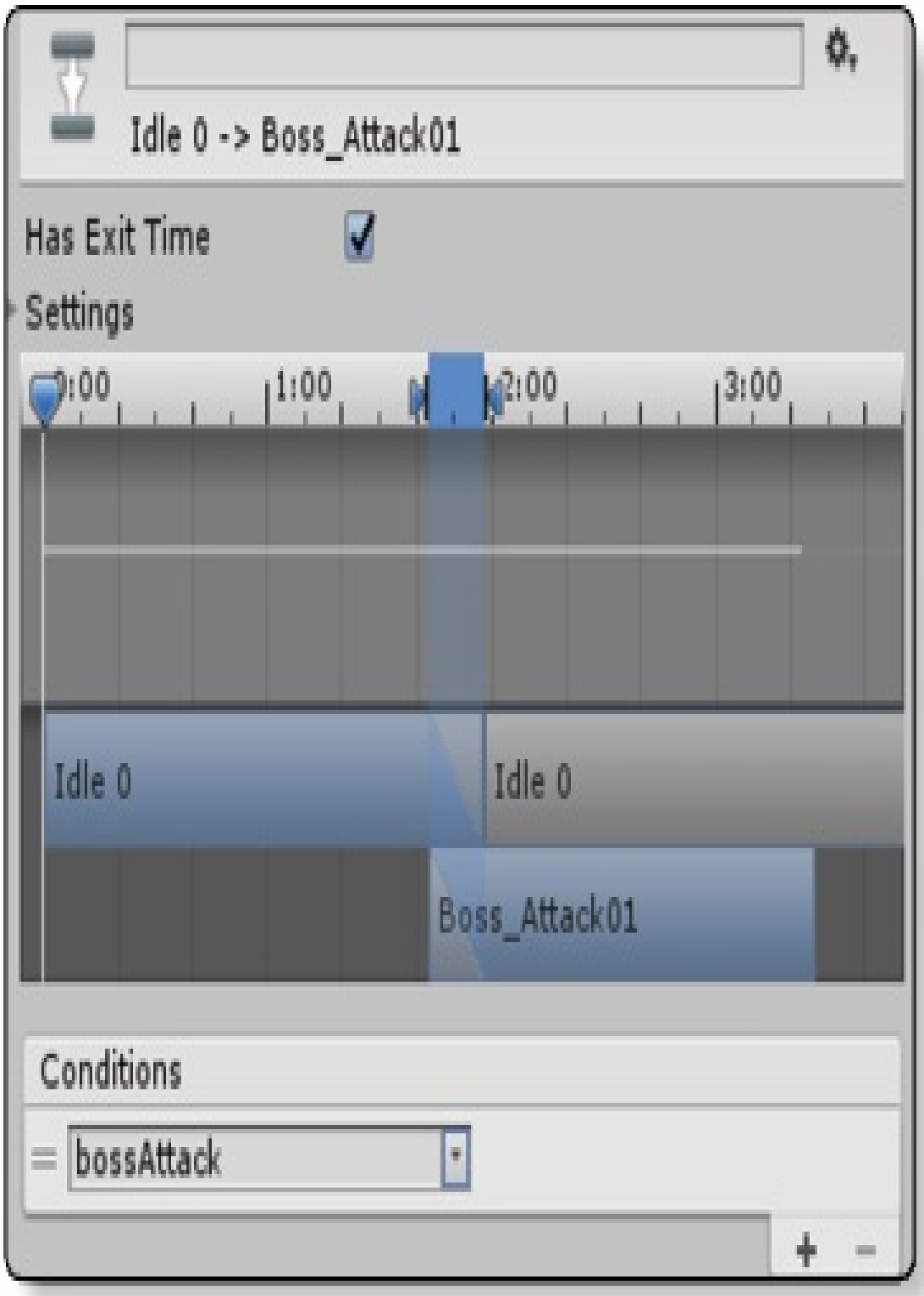

Layers
Parameters
qrName
bossAwake
bossAttack

Creación de parámetro en Animator

En Animator asigne el clip de animación “Boss_Attack01” con las transiciones que se muestran y con su parámetro “bossAttack”.


Entry
Idle
Boss_Attack01
Exit
Boss_Jump

Asignación de clip de animación de ataque


Idle 0 -> Boss_Attack01
Has Exit Time
Settings
0:00
1:00
2:00
3:00
Idle 0
Idle 0
Boss_Attack01
Conditions
bossAttack

Asignación de parámetro en transición

NOTA:

En la transición de regreso a Idle no requiere de ningún parámetro, pero asegúrese de activar “Has Exit Time”.

Vaya al script “BossController” y añada el siguiente código para indicar que corra la animación de ataque.

```
if (inBattle) {
    if(!attacking) {
        idleTimer += Time.deltaTime;
    } else {
        idleTimer = 0.0f;
        attackTimer += Time.deltaTime;
        if (attackTimer >= attackWaitTime) {
            attacking = false;
            anim.SetTrigger ("bossAttack");
            attackTimer = 0.0f;
            print ("BOSS SMASH!");
            swordTrigger.enabled = true;
            print ("SwordTrigger is enabled");
        }
```

Código para mandar llamar la animación de ataque

Previsualice el juego para observar que cuando se active su fase de ataque el jefe realiza su primer ataque y luego regresa a “Idle”.

Previsualización de ataque

PROBLEMA: El ataque no afecta a nuestro Player.

SOLUCIÓN: Deberá de escalar el BoxCollider de la espada .

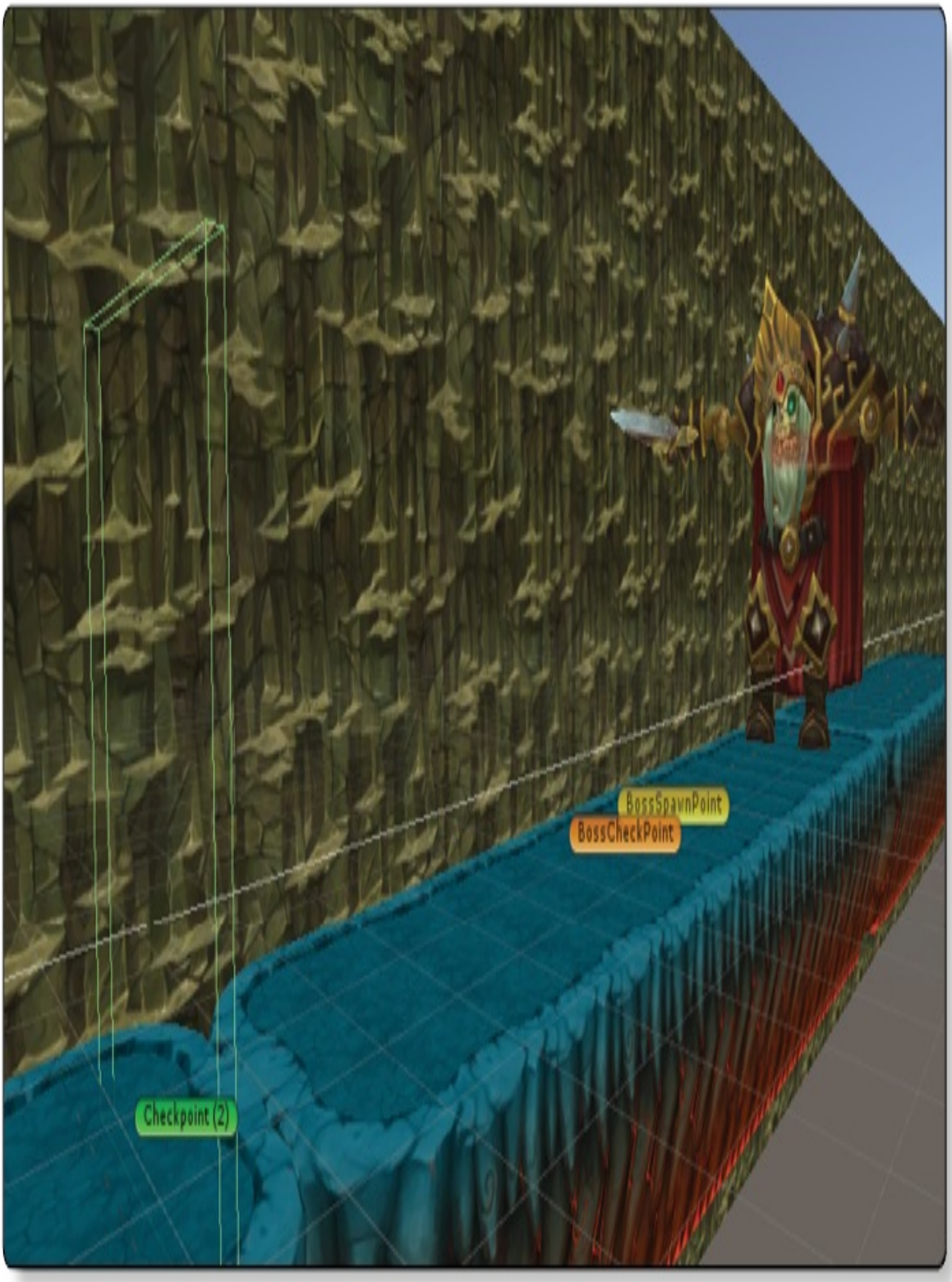
HeadTrigger

Escala de “BoxCollider” de la espada

SOLUCIÓN: Rotar un poco el objeto “Skeleton_King” (hijo de “Boss”) para que en el movimiento de ataque pueda colisionar con nuestro “Player” y reposicionar su “HeadTrigger” conforme a la cabeza de nuestro “Boss”.

PROBLEMA: Cuando nuestro “Player” muere debe de aparecer localizado en un punto anterior a la batalla final.

SOLUCIÓN: Haga una copia del ultimo “Checkpoint” (sin contar “BossCheckpoint) y colóquelo justo antes del “BossCheckpoint”.


BossSpawnPoint
BossCheckPoint
Checkpoint (2)

Colocación de nuevo CheckPoint

PROBLEMA: Cuando muere nuestro “Player” la cámara continua en su estado de “BossCameraTarget”.

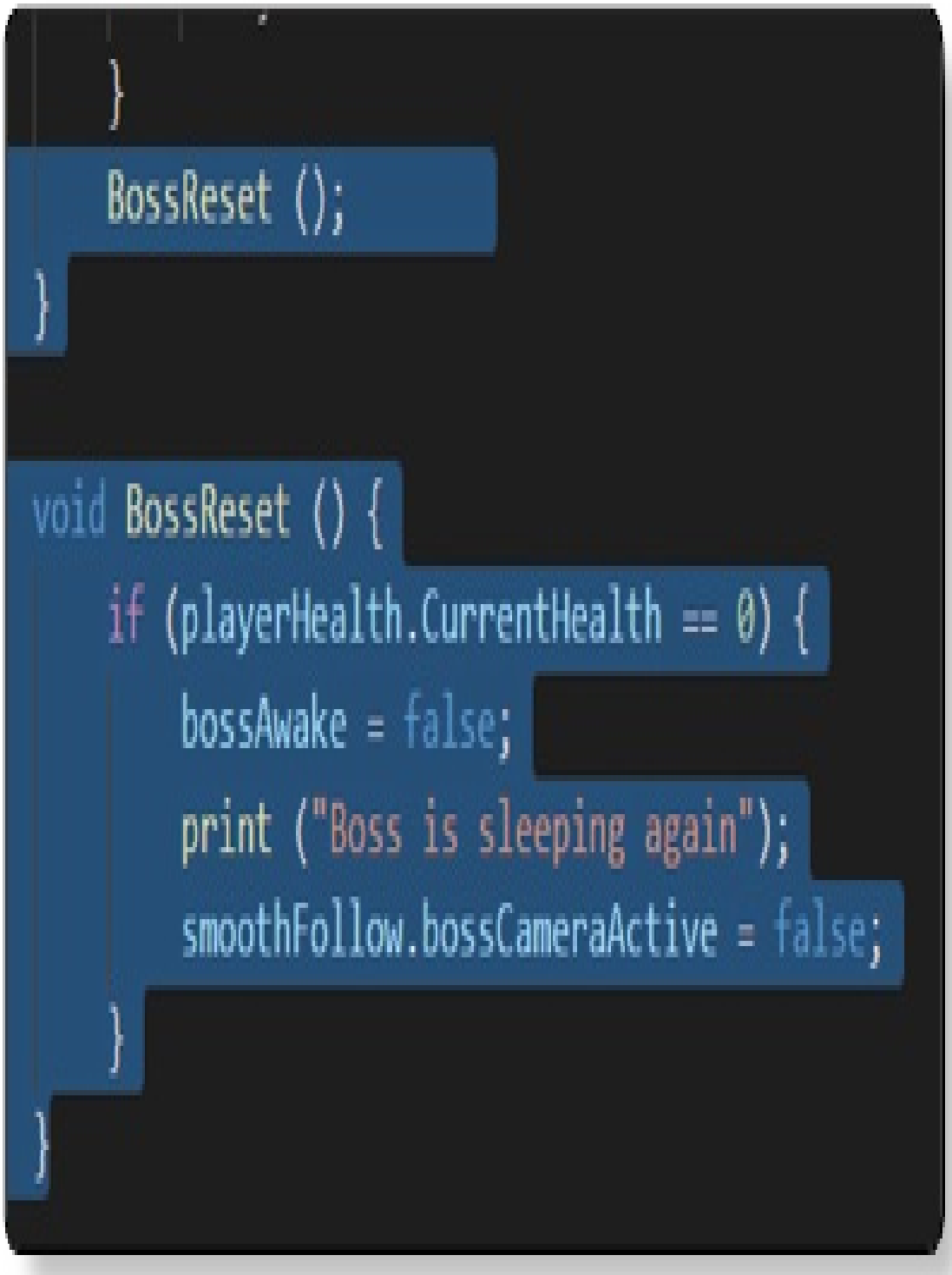
SOLUCIÓN: Vaya al script “BossController”, escriba e inicialice las siguientes variables.

```
private BoxCollider swordTrigger;

private SmoothFollow smoothFollow;
private GameObject player;
private PlayerHealth playerHealth;

// Use this for initialization
void Start () {
    anim = GetComponent<Animator> ();
    bossHealthBar.SetActive (false);
    swordTrigger = GameObject.Find ("Boss").GetComponentInChildren<BoxCollider>();
    smoothFollow= GameObject.FindGameObjectWithTag ("MainCamera").GetComponent<SmoothFollow>();
    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealth>();
}
```

Creación e inicialización de variables

A continuación, debemos de crear una función llamada “BossReset” en el mismo script “BossController”, que contendrá todas las condiciones para la segunda vuelta de la batalla final.

Haga la siguiente función y mándela llamar en la función “Update” .


}
BossReset ();
}
void BossReset () {
if (playerHealth.CurrentHealth == 0) {
bossAwake = false;
print ("Boss is sleeping again");
smoothFollow.bossCameraActive = false;
}
}

Función BossReset

Previsualice el juego para observar que al morir nuestro “Player” se desactiva “BossCameraTarget” en el objeto “MainCamera”, volviendo a la cámara original.

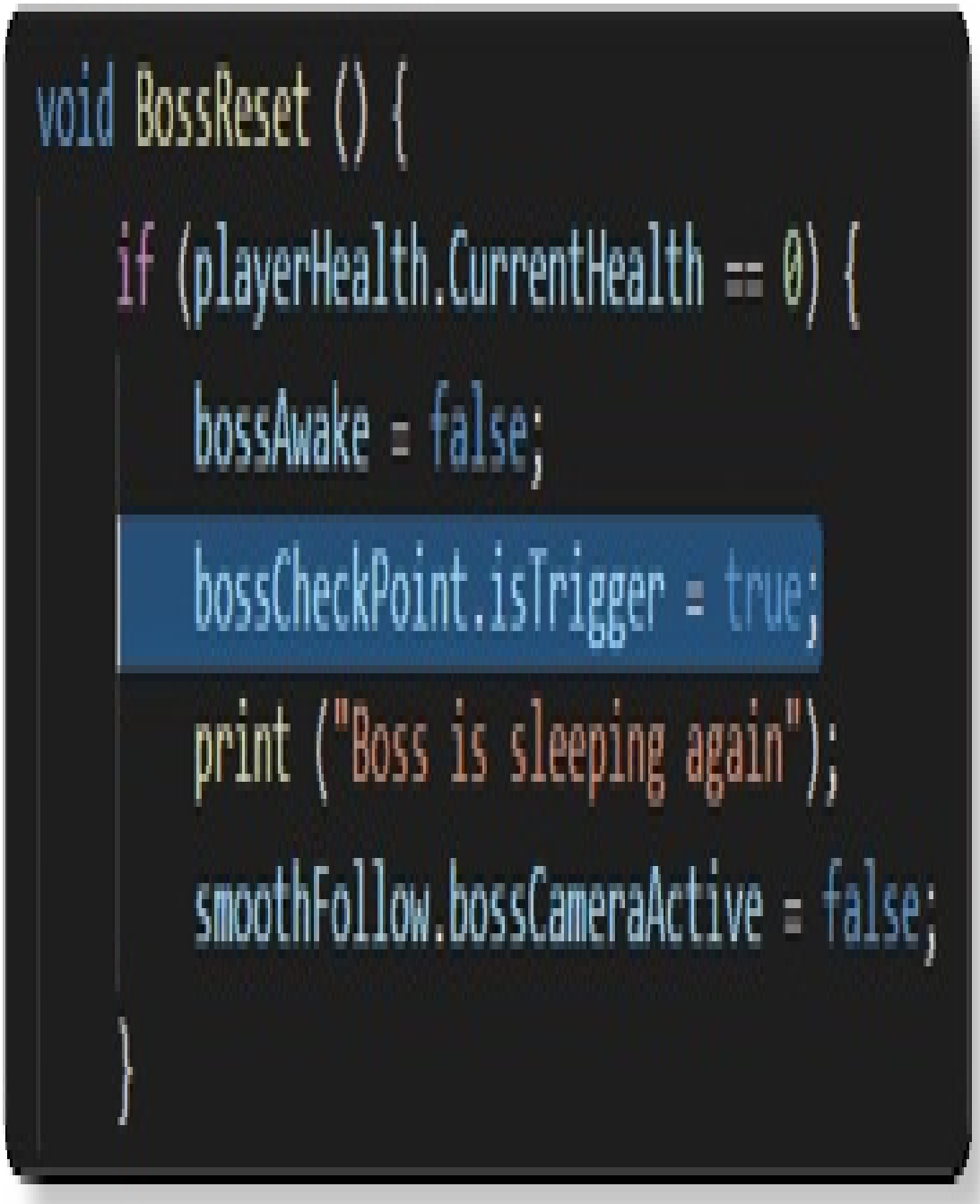
PROBLEMA: Después de que muere nuestro “Player” y reaparece para intentar matar nuevamente al jefe, no puede pasar por “BossCheckpoint” ya que desactivamos su funcionalidad de “IsTrigger” convirtiéndolo en “Collider.

SOLUCIÓN: Vaya al script “BossController”, escriba e inicialice la siguiente variable.

```
private PlayerHealth playerHealth;


private BoxCollider bossCheckPoint;


// Use this for initialization
void Start () {
    anim = GetComponent<Animator> ();
    bossHealthBar.SetActive (false);
    swordTrigger = GameObject.Find ("Boss").GetComponentInChildren<BoxCollider>();
    smoothFollow= GameObject.FindGameObjectWithTag ("MainCamera").GetComponent<SmoothF
    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealth>();
    bossCheckPoint = GameObject.Find ("BossCheckPoint").GetComponent<BoxCollider>();
```

Creación e inicialización de variables

Vaya a la función “BossReset” y añada la siguiente línea de código:


void BossReset () {
if (playerHealth.CurrentHealth == 0) {
bossAwake = false;
bossCheckPoint.isTrigger = true;
print ("Boss is sleeping again");
smoothFollow.bossCameraActive = false;
}

Código para reactivar “IsTrigger”

Previsualice el juego para observar que nuestro “Player” puede volver a pasar por el “BossCheckPoint” después de que lo han matado una vez.

PROBLEMA: El componente “CharacterMovement” de nuestro “Player” se encuentra inactivo la segunda vez que pasa por el “BossCheckPoint”.

SOLUCIÓN: Recordando que la activación del script “CharacterMovement” de nuestro “Player” se encuentra en el “AnimationEvent” del salto de nuestro jefe, en el script “BossController”, vaya a la función “BossReset” y escriba el siguiente código:

```
void BossReset () {
    if (playerHealth.CurrentHealth == 0) {
        bossAwake = false;
        bossCheckPoint.isTrigger = true;
        print ("Boss is sleeping again");
        smoothFollow.bossCameraActive = false;
        anim.Play("Idle");
        anim.SetBool("bossAwake", false);
    }
```

Código para mandar llamar animaciones

Previsualice el juego para observar lo siguiente cuando nuestro “Player” pasa nuevamente por el “BossCheckPoint”.

Nuestro jefe da nuevamente el salto inicial

Se reactiva el script de “CharacterMovement” de nuestro “Player”

PROBLEMA: A la siguiente vida del Player, el jefe no le hace daño.

SOLUCIÓN: Vaya al script “PlayerHealth” y comente el código que se muestra.

```
void takeHit () {
    if (currentHealth >= 0  ) {
        GameManager.instance.PlayerHit (currentHealth);
        anim.Play ("Hurt");
        currentHealth -= 10;
        healthSlider.value = currentHealth;
        audio.PlayOneShot (hurtAudio);
    }

    if (currentHealth <= 0) {
        //GameManager.instance.PlayerHit (currentHealth);
        anim.SetTrigger ("isDead");
        characterMovement.enabled = false;
        audio.PlayOneShot (deadAudio);
    }
}
```

Comentar código para desactivarlo

PROBLEMA: Cuando reaparece la cámara tiene un movimiento extraño y reaparece en una vista diferente a la original.

SOLUCIÓN: Vaya al script “SmoothFollow y edite el código de la función “TrackPlayer”.

```
if (bossCameraActive){
    transform.position = new Vector3
    (Mathf.Lerp (transform.position.x, bossCameraTarget.position.x, 1.0f / cameraSpeed)
    ,Mathf.Lerp (transform.position.y, bossCameraTarget.position.y, 1.0f / cameraSpeed),
    Mathf.Lerp (transform.position.z, bossCameraTarget.position.z, 1.0f / cameraSpeed));
} else {
    transform.position = new Vector3
    (Mathf.Lerp (transform.position.x, cameraTarget.position.x, 1.0f / cameraSpeed)
    ,Mathf.Lerp (transform.position.y, cameraTarget.position.y, 1.0f / cameraSpeed),
    Mathf.Lerp (transform.position.z, cameraTarget.position.z, 1.0f / cameraSpeed));
}
```

Edición de función TrackPlayer

Previsualice el juego, al reaparecer el “Player” después de su muerte reposicione manualmente la cámara y tome sus lecturas de transformación.

Detenga la previsualización y asigne las lecturas tomadas para “CameraTarget”.

PROBLEMA: La salud del jefe no se reinicia después de que muere nuestro “Player”.

SOLUCIÓN: Vaya al script “BossController”, escriba e inicialice la siguiente variable.

```
private BoxCollider bossCheckpoint;

private BossHealth bossHealth;


// Use this for initialization
void Start () {


    anim = GetComponent<Animator> ();
    bossHealthBar.SetActive (false);
    bossHealth = GetComponentInChildren<BossHealth>();
    swordTrigger = GameObject.Find ("Boss").GetComponentInCh
```

Creación e inicialización de variables

Vaya a la función “BossReset” y añada la siguiente línea de código:

```
void BossReset (){

    if (playerHealth.CurrentHealth == 0) {
        bossAwake = false;
        bossCheckpoint.isTrigger = true;
        print ("Boss is sleeping again");
        anim.Play("Idle");
        anim.SetBool("bossAwake", false);
        bossHealthBar.SetActive (false);
        swordTrigger.enabled = false;
        bossHealth.bossHealth = 20;
```

Código para resetear la salud del jefe

Previsualice el juego para observar que todos los problemas se han resuelto.

Programación del segundo y tercer ataque del jefe

En este apartado programaremos los siguientes ataques de nuestro jefe y los encapsularemos para mandarlos llamar por separado para previsualizarlos.

Lo primero que haremos será encapsular el primer método de ataque.

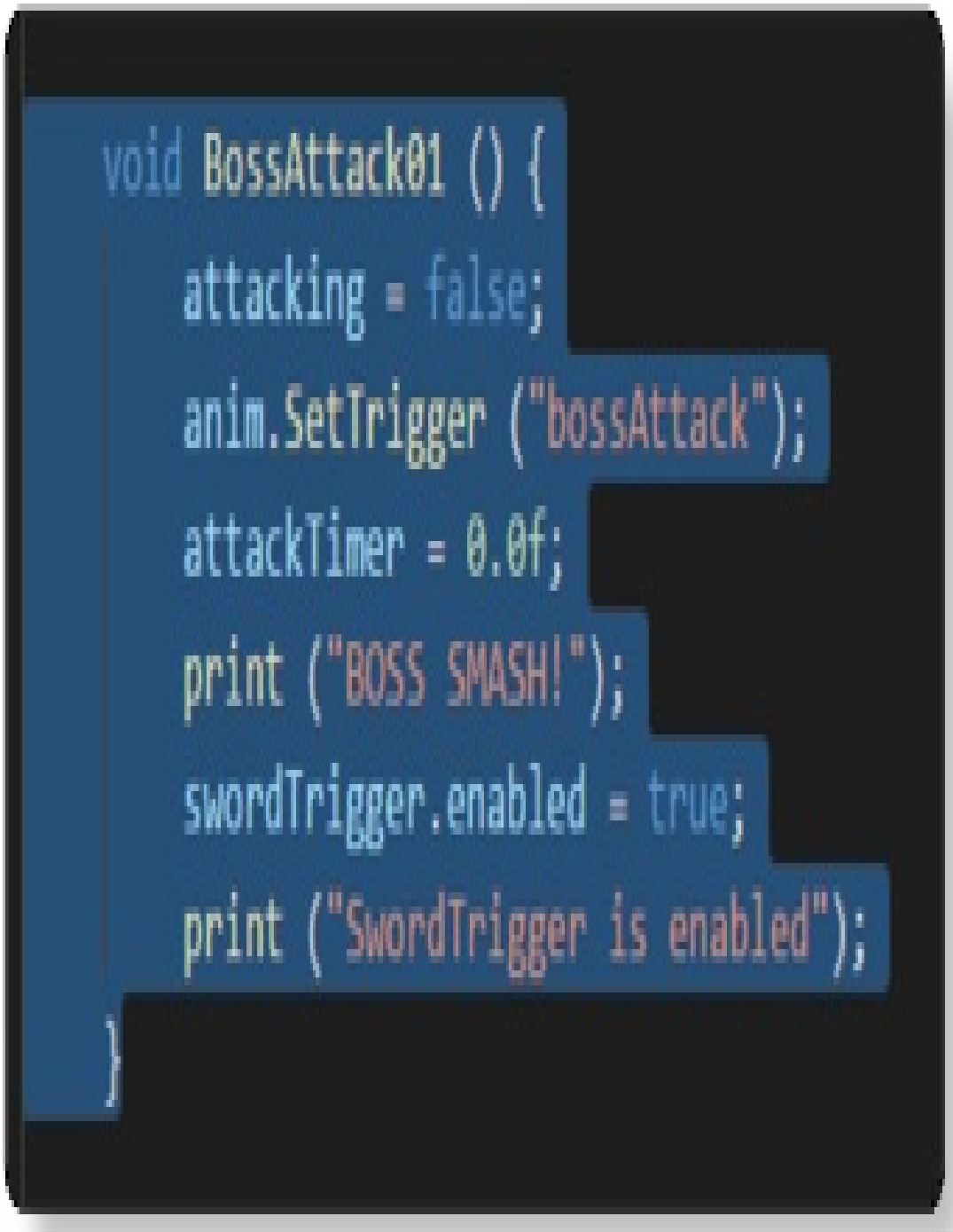
Vaya al script “BossController” y seleccione el código que se muestra.

```
void Update () {

    if(bossAwake)
    {
        print("Boss is awake");
        anim.SetBool("bossAwake", true);
        bossHealthBar.SetActive (true);

        if (inBattle) {
            if(!attacking) {
                idleTimer += Time.deltaTime;
            } else {
                idleTimer = 0.0f;
                attackTimer += Time.deltaTime;
                if (attackTimer >= attackWaitTime) {
                    attacking = false;
                    anim.SetTrigger ("bossAttack");
                    attackTimer = 0.0f;
                    print ("BOSS SMASH!");
                    swordTrigger.enabled = true;
                    print ("SwordTrigger is enabled");
                }
            }
            if(idleTimer >= idleWaitTime ){
                print ("Boss attack 01");
```

Código del primer ataque

Corte el código (Control + X).

Haga una nueva función, nómbrela “BossAttack01” y pegue (Control + V) el código cortado.


void BossAttack01 () {
attacking = false;
anim.SetTrigger ("bossAttack");
attackTimer = 0.0f;
print ("BOSS SMASH!");
swordTrigger.enabled = true;
print ("SwordTrigger is enabled");

Función BossAttack01

Vaya a la parte del código en donde corto las líneas y mande llamar la función nueva.

```
if (inBattle) {
    if(!attacking) {
        idleTimer += Time.deltaTime;
    } else {
        idleTimer = 0.0f;
        attackTimer += Time.deltaTime;
        if (attackTimer >= attackWaitTime) {
            BossAttack01 ();
        }
    }
```

Mandar llamar la función del primer ataque

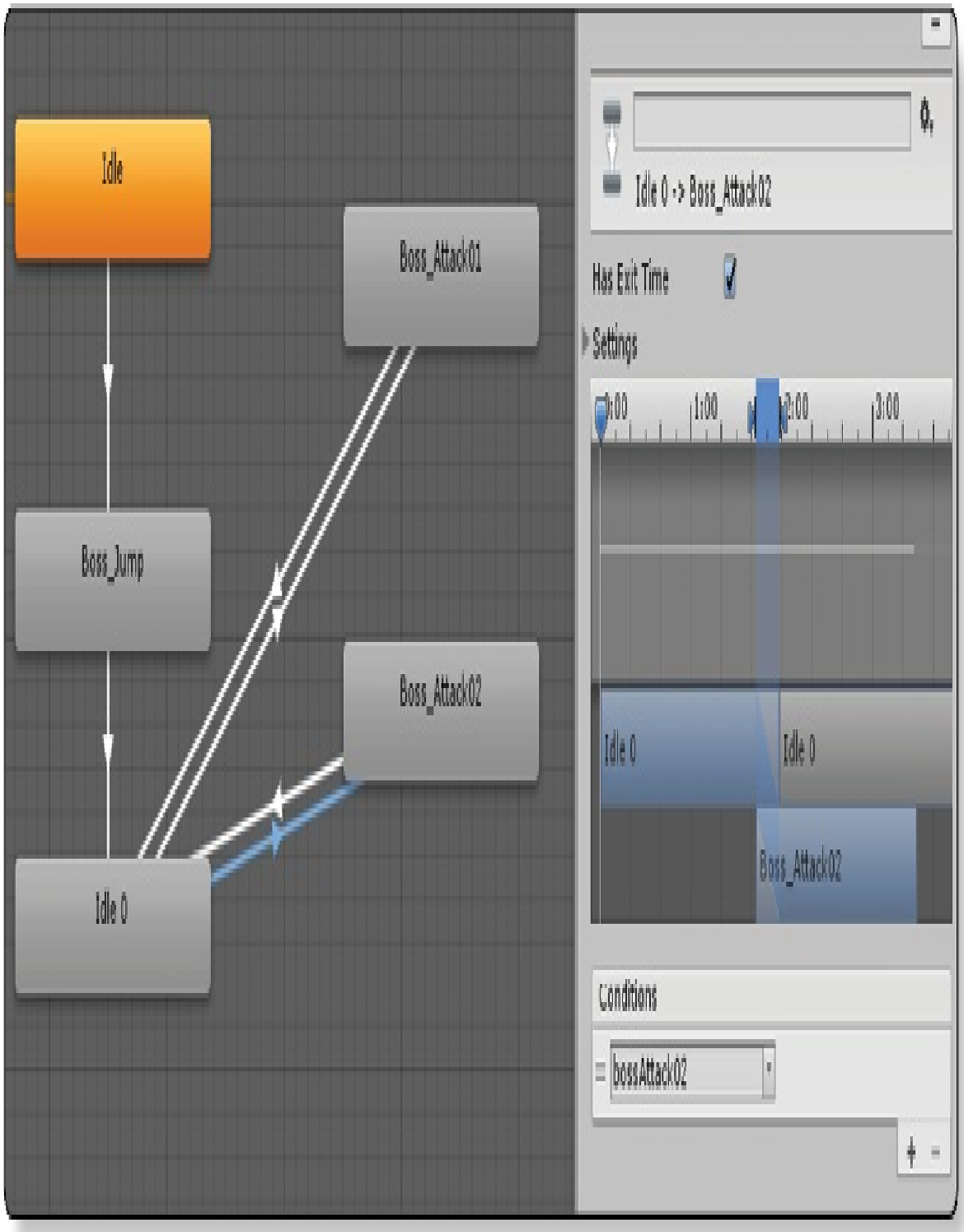
Previsualice el juego para comprobar que todo sigue funcionando correctamente.

A continuación, programaremos el segundo ataque del jefe.

Vaya a Unity, seleccione el objeto “Boss” y abra su ventana de “Animator”.

En Animator haga un nuevo parámetro tipo “Trigger” y nómbrelo “BossAttack02”.

Asigne el clip de animación “Boss_Attack02” sus transiciones como se muestra y asigne el parámetro “BossAttack02” en su transición de entrada.


Idle
Boss_Attack01
Boss_Jump
Boss_Attack02
Idle 0
Idle 0 -> Boss_Attack02
Has Exit Time
Settings
0:00
1:00
2:00
3:00
Idle 0
Idle 0
Boss_Attack02
Conditions
bossAttack02

Asignación de animación y parámetro de 2o ataque

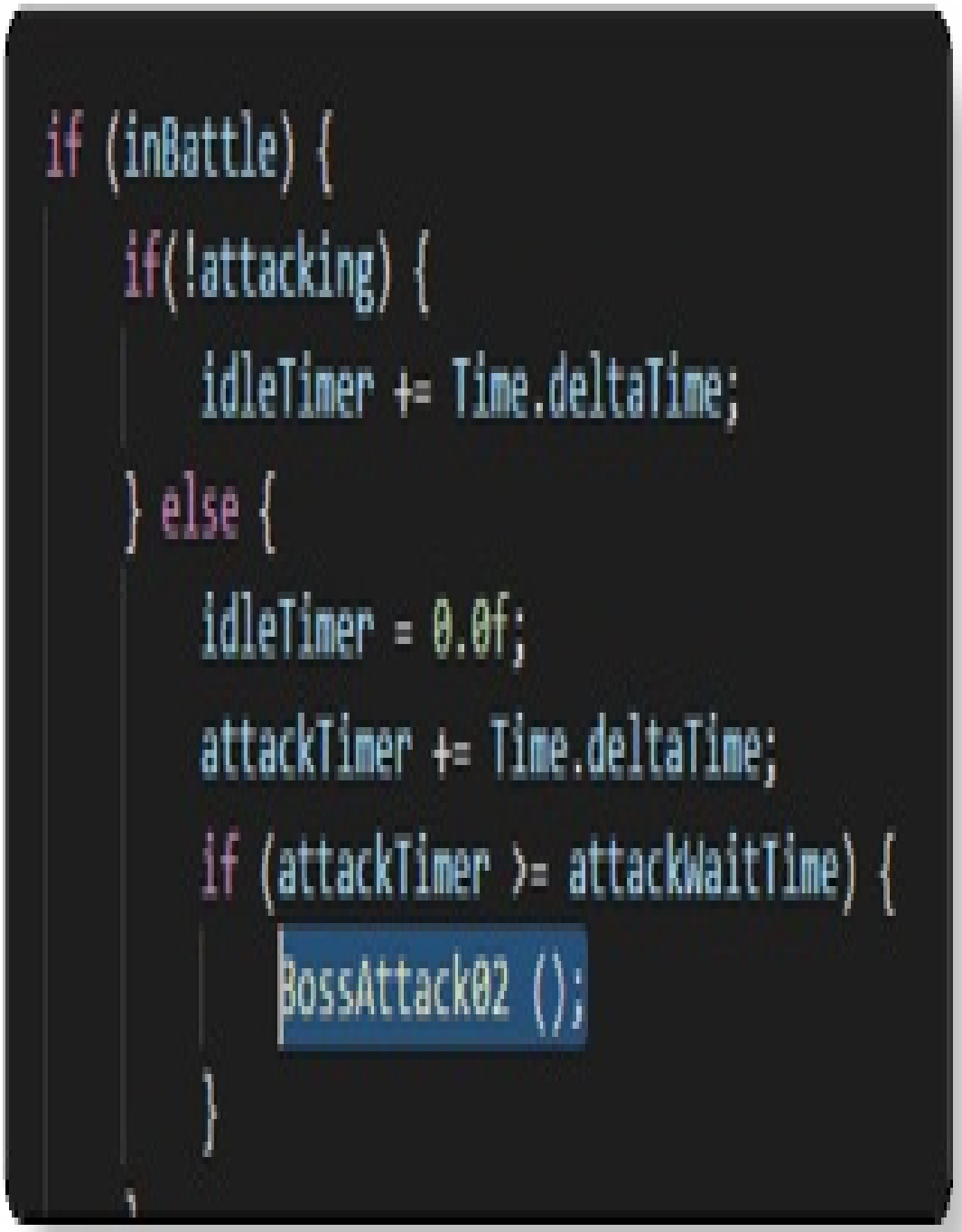
Vaya al script “BossController” y haga la siguiente función.

```
void BossAttack01 () {
    attacking = false;
    anim.SetTrigger ("bossAttack");
    attackTimer = 0.0f;
    print ("Boss Attack 01");
    swordTrigger.enabled = true;
}

void BossAttack02 () {
    attacking = false;
    anim.SetTrigger ("bossAttack02");
    attackTimer = 0.0f;
    print ("Boss Attack 02");
    swordTrigger.enabled = true;
}
```

Creación de la función de BossAttack02

Antes de programar los ataques aleatorios, queremos hacer “test” del nuevo ataque, para ello reemplazaremos el ataque01 por el ataque02.

Reemplace el código que se muestra.

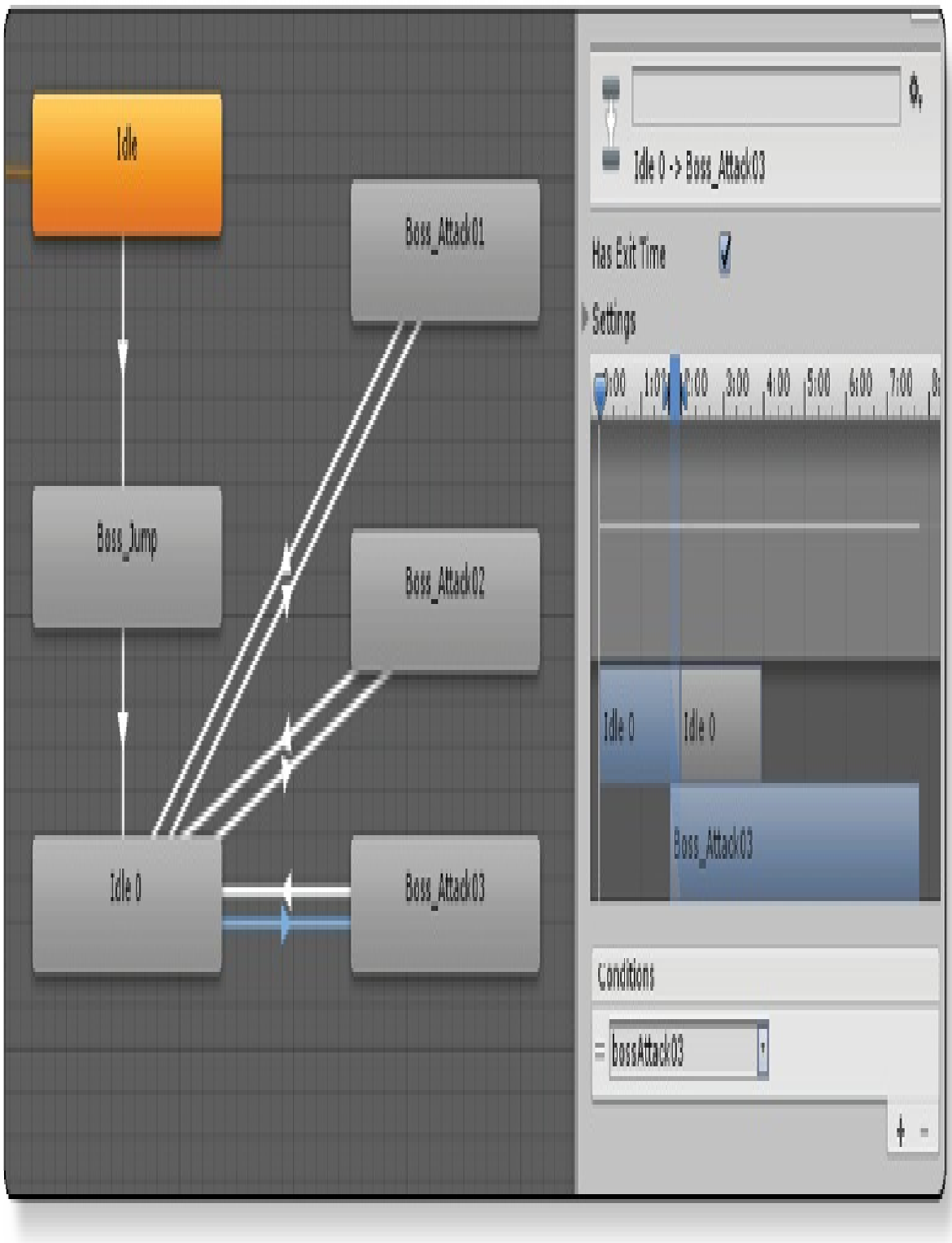

if (inBattle) {
if(!attacking) {
idleTimer += Time.deltaTime;
} else {
idleTimer = 0.0f;
attackTimer += Time.deltaTime;
if (attackTimer >= attackWaitTime) {
BossAttack02 ();
}

Mandar llamar el segundo ataque

Previsualice el juego para observar el segundo ataque.

Previsualización de segundo ataque

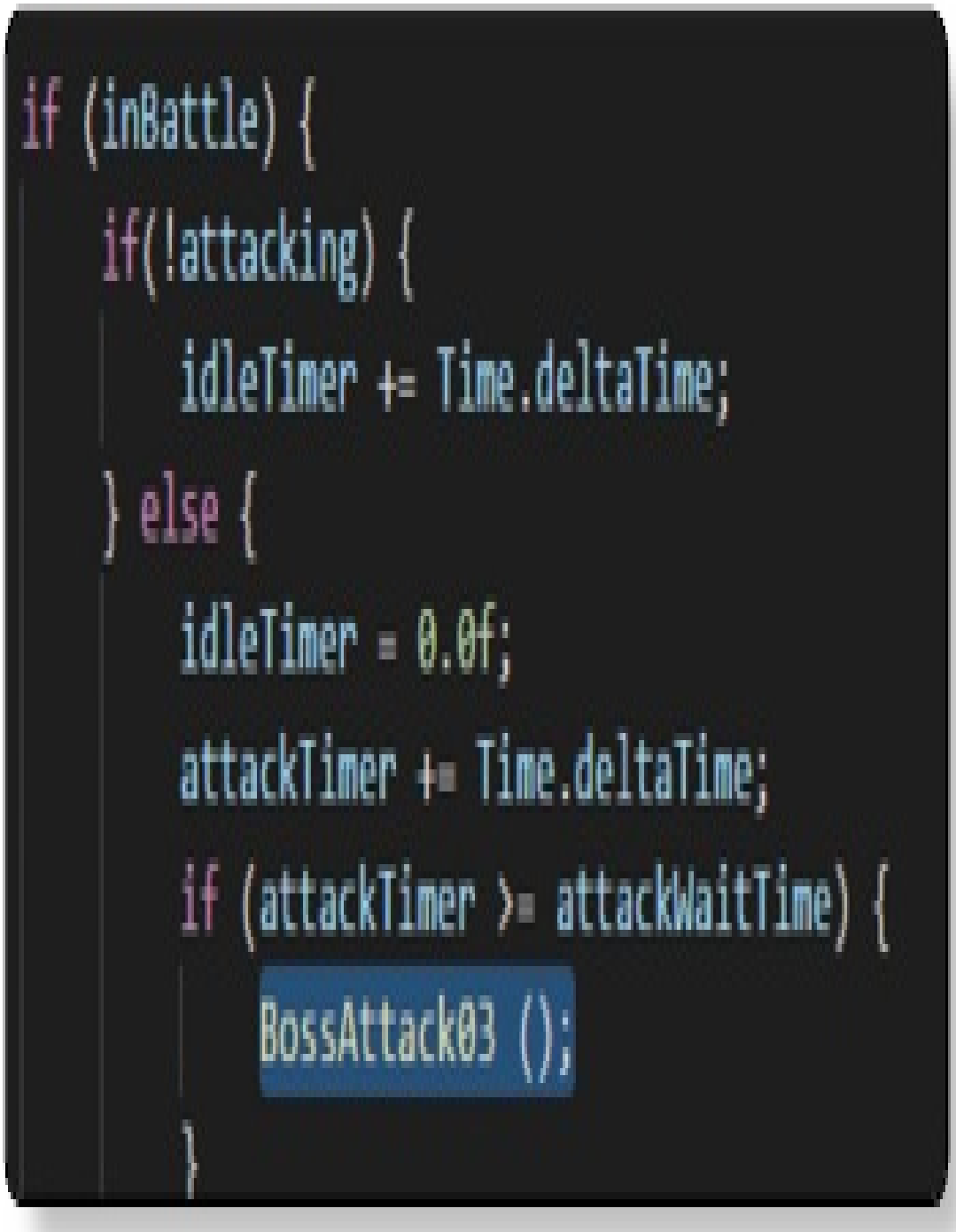
Repita el proceso para crear el tercer ataque .


Idle
Boss_Attack01
Boss_Jump
Boss_Attack02
Idle 0
Boss_Attack03
Idle 0 -> Boss_Attack03
Has Exit Time
Settings
Idle 0
Idle 0
Boss_Attack03
Conditions
bossAttack03

Asignación de animación y parámetro de 3er ataque

```
void BossAttack02 () {
    attacking = false;
    anim.SetTrigger ("bossAttack02");
    attackTimer = 0.0f;
    print ("Boss Attack 02");
    swordTrigger.enabled = true;
}

void BossAttack03 () {
    attacking = false;
    anim.SetTrigger ("bossAttack03");
    attackTimer = 0.0f;
    print ("Boss Attack 03");
    swordTrigger.enabled = true;
}
```

Creación de función “BossAttack03”


if (inBattle) {
if(!attacking) {
idleTimer += Time.deltaTime;
} else {
idleTimer = 0.0f;
attackTimer += Time.deltaTime;
if (attackTimer >= attackWaitTime) {
BossAttack03 ();
}

Mandar llamar el tercer ataque

Previsualice el juego para observar el tercer ataque.

X2

Previsualización de tercer ataque

Programación de la generación de rocas cayendo

En este apartado crearemos un sistema de partículas para la generación de piedras cayendo que formarán parte del tercer ataque de nuestro jefe.

Nos adentraremos en aspectos avanzados de partículas asignando un “mesh” poligonal a las partículas y controlaremos su duración, aparición y sus componentes para que le hagan daño a nuestro “Player”.

Importe de la carpeta de “resources” el modelo “Rock01_lava” a la carpeta de “Models”.

Haga un nuevo “EmptyObject”, renómbrelo “RockPS” (por ParticleSystem), asigne un label verde y colóquelo en la escena donde se muestra.

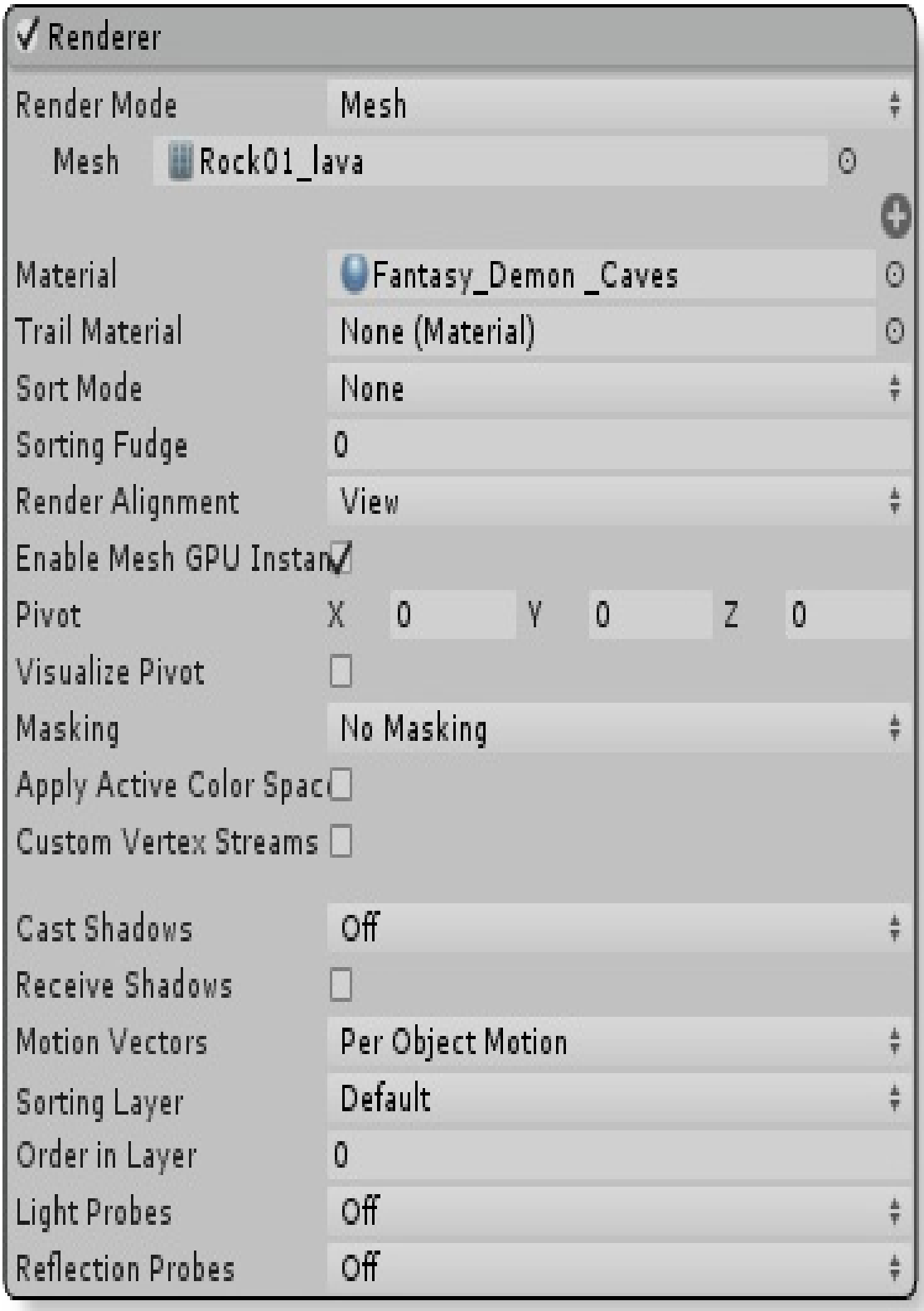
Rock_PS
HeadTrigger
BossSpavnPoint

Colocación de objeto “Rock_PS”

Asigne al nuevo objeto un componente “Particle System”.

A continuación, debemos de asignar los valores necesarios para nuestro sistema de partículas.

En el componente “ParticleSystem”, expanda la pestaña “Renderer” para asignarle el “mesh” “Rock01_lava” a las partículas, así como asignar su textura correspondiente (el material previamente creado “Fantasy_Demon_Caves).


Renderer
Render Mode
Mesh
Mesh
Rock01_lava
Material
Fantasy_Demon _Caves
Trail Material
None (Material)
Sort Mode
None
Sorting Fudge
0
Render Alignment
View
Enable Mesh GPU Instan
Pivot
X 0
Y 0
Z 0
Visualize Pivot
Masking
No Masking
Apply Active Color Spac
Custom Vertex Streams
Cast Shadows
Off
Receive Shadows
Motion Vectors
Per Object Motion
Sorting Layer
Default
Order in Layer
0
Light Probes
Off
Reflection Probes
Off

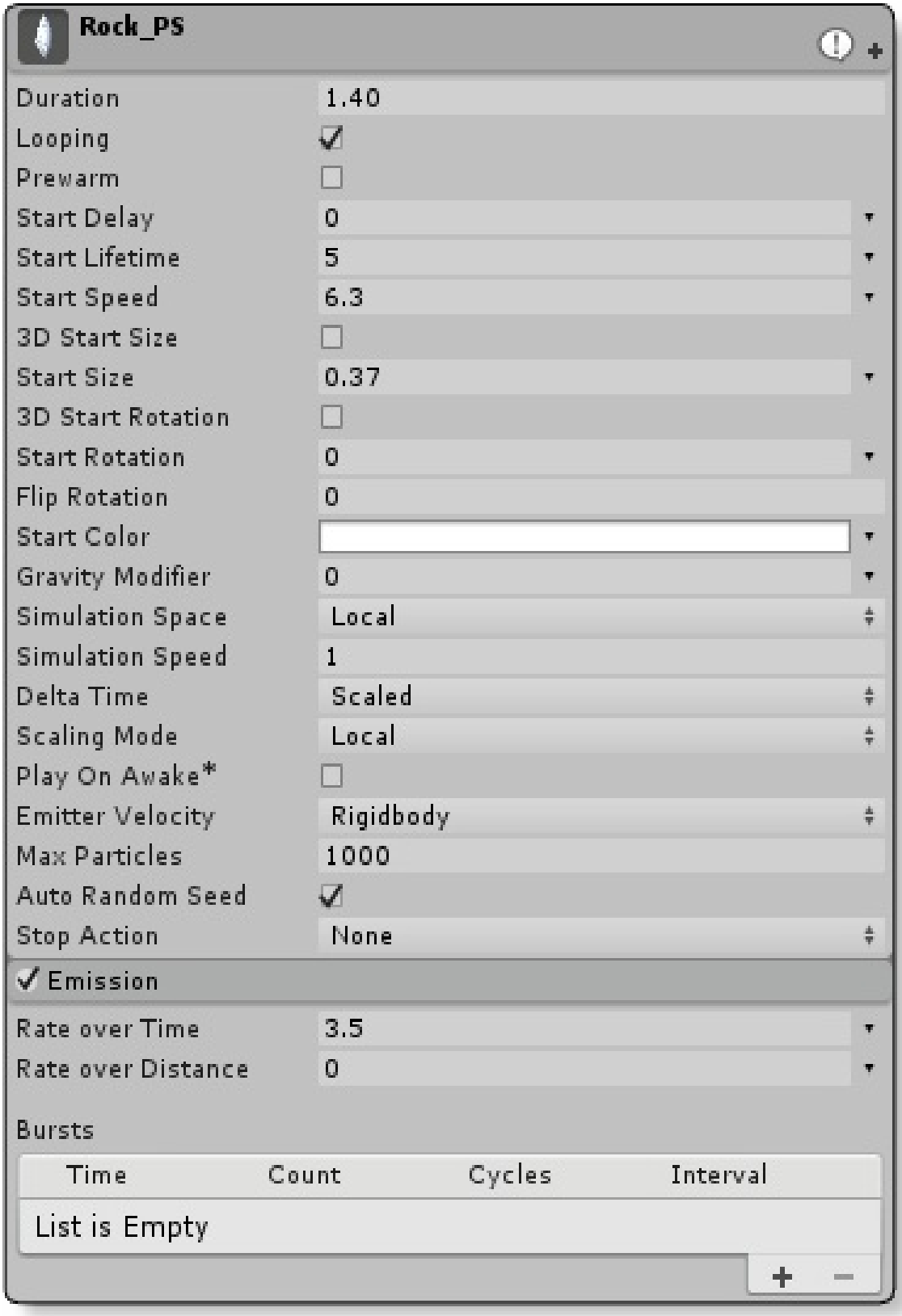
Pestaña Renderer de ParticleSystem

NOTA:

El símbolo de “check” (paloma) que se encuentra marcado junto a la ventana “Renderer” nos indican las pestañas que se encontrarán activas de nuestro sistema de partículas.

A continuación, debemos de establecer la duración de las partículas, el “looping”, la velocidad, el tamaño de inicio, el modificador de gravedad y la emisión.

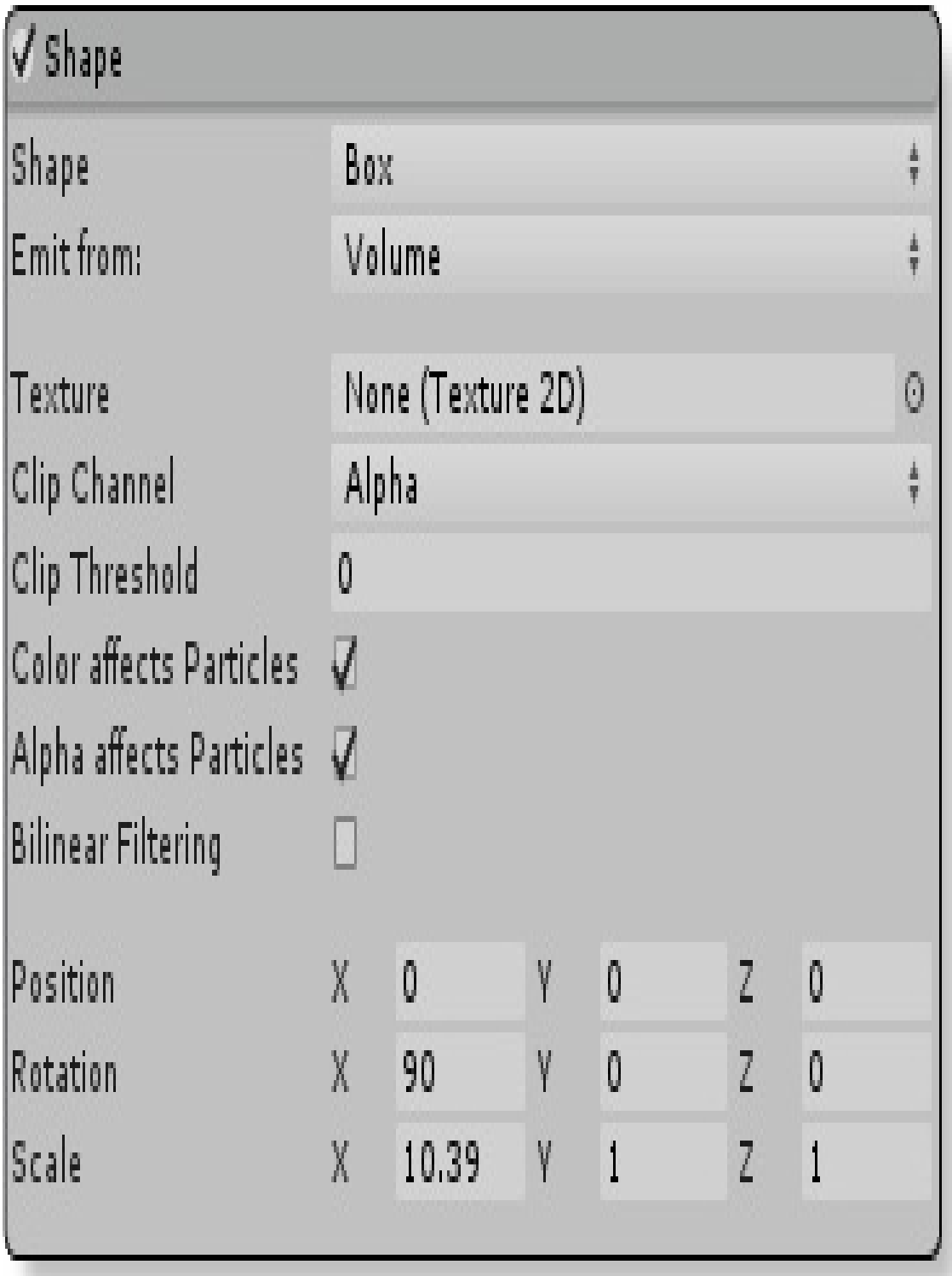
Expanda la pestaña inicial y la de “Emission” y asigne los valores que se muestran.


Rock_PS
Duration 1.40
Looping
Prewarm
Start Delay 0
Start Lifetime 5
Start Speed 6.3
3D Start Size
Start Size 0.37
3D Start Rotation
Start Rotation 0
Flip Rotation 0
Start Color
Gravity Modifier 0
Simulation Space Local
Simulation Speed 1
Delta Time Scaled
Scaling Mode Local
Play On Awake*
Emitter Velocity Rigidbody
Max Particles 1000
Auto Random Seed
Stop Action None
Emission
Rate over Time 3.5
Rate over Distance 0
Bursts
Time
Count
Cycles
Interval
List is Empty

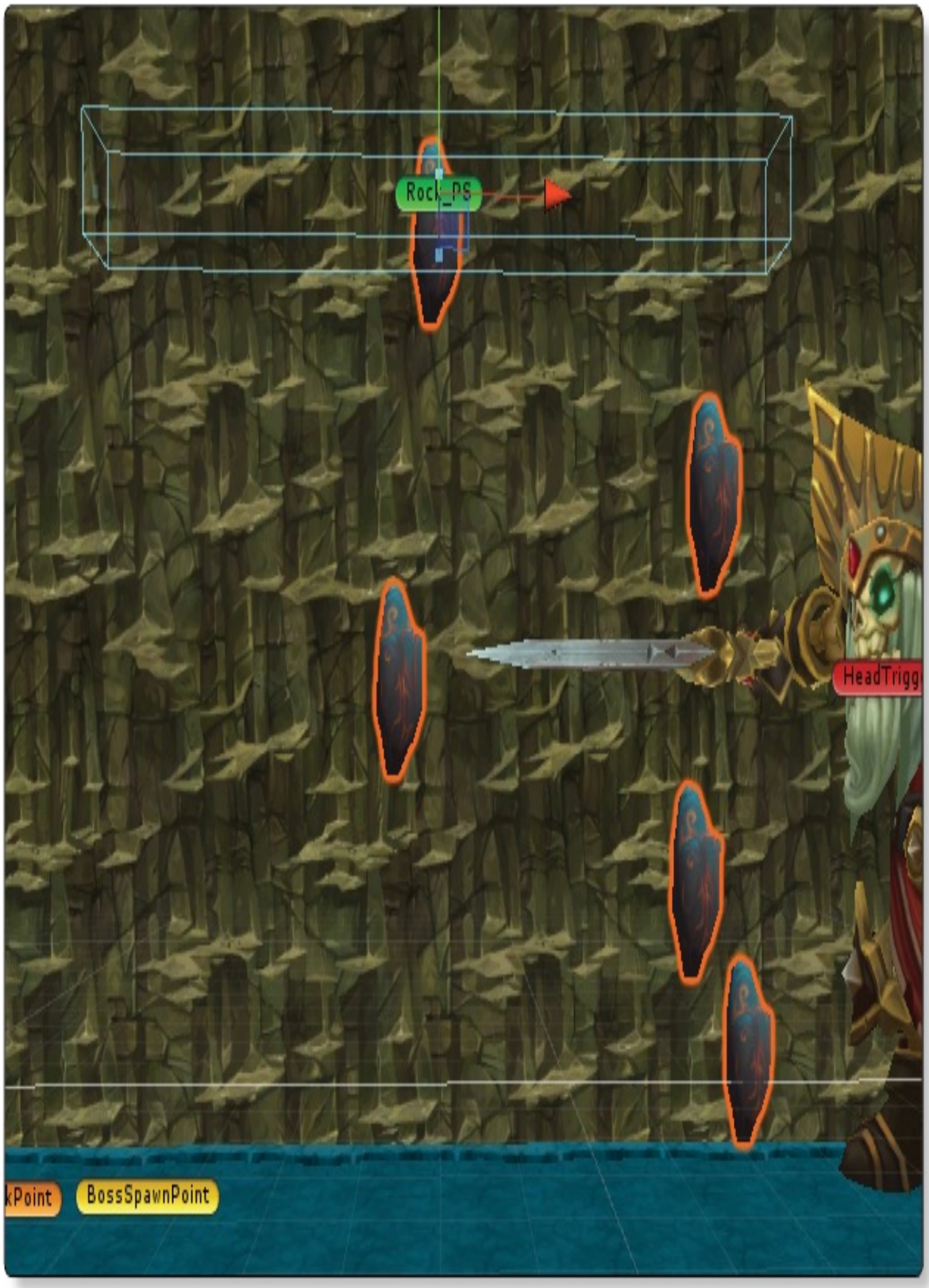
Valores de ParticleSystem y Emission

A continuación, debemos de asignar un tamaño de escala a nuestro emisor y eso lo hacemos en la pestaña “Shape”.

Expanda la pestaña “Shape”, marque su activación y asigne los valores que se muestran.


Shape
Shape
Box
Emit from:
Volume
Texture
None (Texture 2D)
Clip Channel
Alpha
Clip Threshold
0
Color affects Particles
Alpha affects Particles
Bilinear Filtering
Position
X 0
Y 0
Z 0
Rotation
X 90
Y 0
Z 0
Scale
X 10.39
Y 1
Z 1

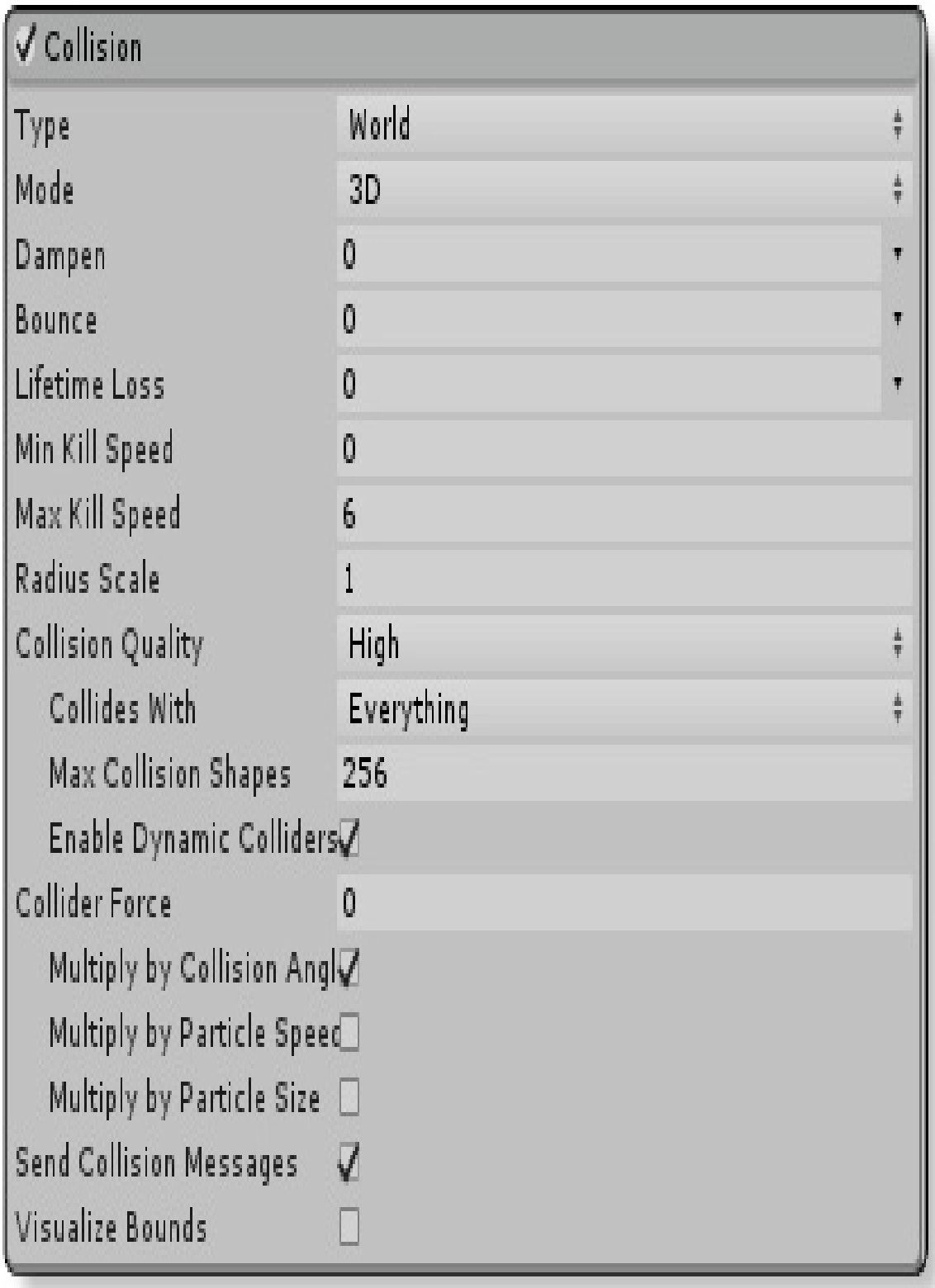
Pestaña Shape de ParticleSystem


Rock_ps
HeadTrigg
kPoint
BossSpawnPoint

Previsualización de la forma del emisor de partículas

A continuación, debemos de asignarle valores de colisión para sus comportamientos.

Expanda y active la pestaña “Collision” y asigne los siguientes valores.

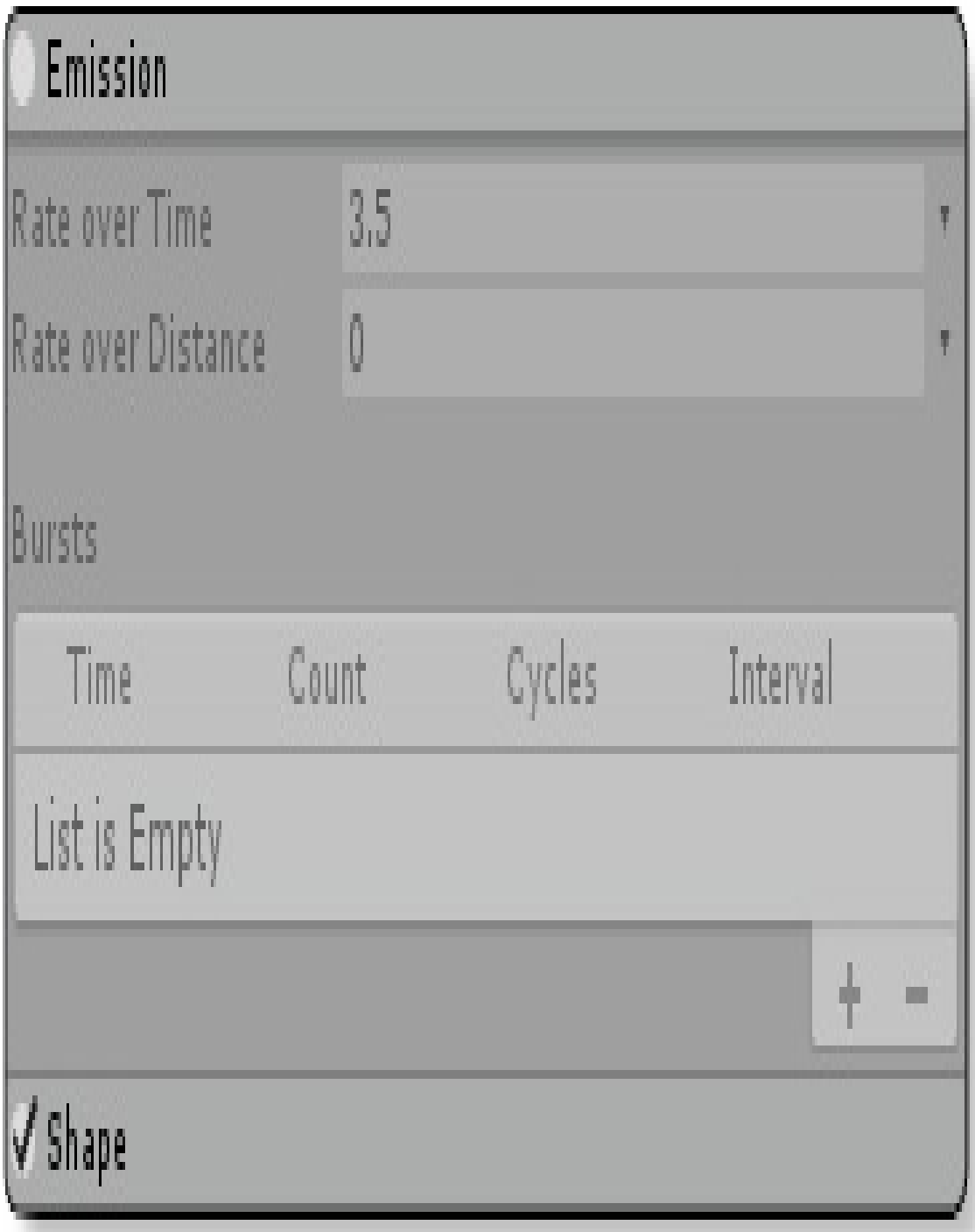

Collision
Type
World
Mode
3D
Dampen
0
Bounce
0
Lifetime Loss
0
Min Kill Speed
0
Max Kill Speed
6
Radius Scale
1
Collision Quality
High
Collides With
Everything
Max Collision Shapes
256
Enable Dynamic Colliders
Collider Force
0
Multiply by Collision Angl
Multiply by Particle Speed
Multiply by Particle Size
Send Collision Messages
Visualize Bounds

Pestaña Collision de ParticleSystem

En este punto ya debe de estar previsualizando las rocas cayendo como partículas.

Para programar la emisión de partículas necesitamos primeramente deshabilitarla.

Deshabilite la pestaña “Emission”.


Emission
Rate over Time
3.5
Rate over Distance
0
Bursts
Time
Count
Cycles
Interval
List is Empty
Shape

Desactivar la función de “Emission”

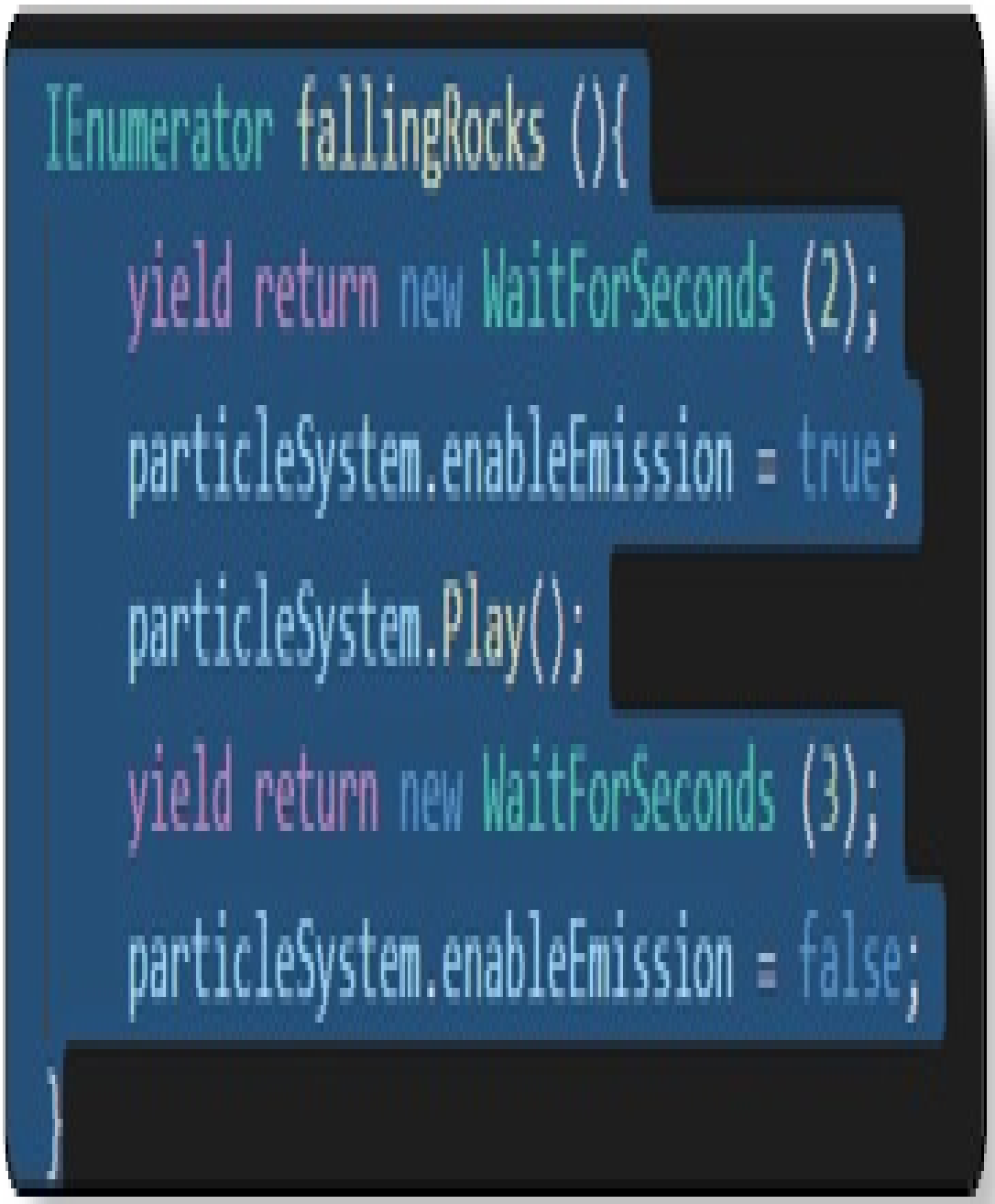
Vaya al script “BossController”, haga e inicialice la siguiente variable.

```
private BossHealth bossHealth;
private ParticleSystem particleSystem;


// Use this for initialization
void Start () {
    anim = GetComponent<Animator> ();
    bossHealthBar.SetActive (false);
    swordTrigger = GameObject.Find ("Boss").GetComponentInChildren<BoxCollider>()
    smoothFollow= GameObject.FindGameObjectWithTag ("MainCamera").GetComponent<Sm
    player = GameManager.instance.Player;
    playerHealth = player.GetComponent<PlayerHealth>();
    bossCheckPoint = GameObject.Find ("BossCheckPoint").GetComponent<BoxCollider>
    bossHealth = GetComponentInChildren<BossHealth>();
    particleSystem = GameObject.Find("Rock_PS").GetComponent<ParticleSystem>();
```

Creación e inicialización de variables

Necesitamos controlar el tiempo en el que caen las rocas, para ello crearemos una función “IEnumerator”.

Escriba la siguiente función debajo de la función “BossAttack03”.


IEnumerator fallingRocks (){
yield return new WaitForSeconds (2);
particleSystem.enableEmission = true;
particleSystem.Play();
yield return new WaitForSeconds (3);
particleSystem.enableEmission = false;
}

IEnumerator para controlar los tiempos de emisión de partículas

Vaya a la función “BossAttack03” y mande llamar la “Coroutine”.

```
void BossAttack03(){
    attacking = false;
    anim.SetTrigger ("bossAttack03");
    attackTimer = 0.0f;
    print ("Boss Attack 03");
    swordTrigger.enabled = true;
    StartCoroutine (fallingRocks ());
}
```

Mandar llamar la Coroutine

Previsualice el juego para observar que cuando el jefe realiza el tercer ataque caen las piedras y cuando se termina el ataque dejan de caer.

2

Previsualización de partículas

A continuación, debemos de hacer que las piedras dañen a nuestro “Player”.

Haga un nuevo C#Script, nómbrelo “RockCollision” y asígnelo al objeto “Rock_PS”.

Abra el script y escriba el siguiente código para que las rocas colisionen con el “Player”.

```
public class RockCollision : MonoBehaviour {

    private GameObject player;

    // Use this for initialization
    void Start () {
        player = GameManager.instance.Player;
    }

    // Update is called once per frame
    void Update () {

    }

    void OnParticleCollision (GameObject player ) {
        Debug.Log ("Player Hit");
    }
}
```

Script RockCollision

En Unity seleccione el objeto “Player” y asigne su layer “Player”.

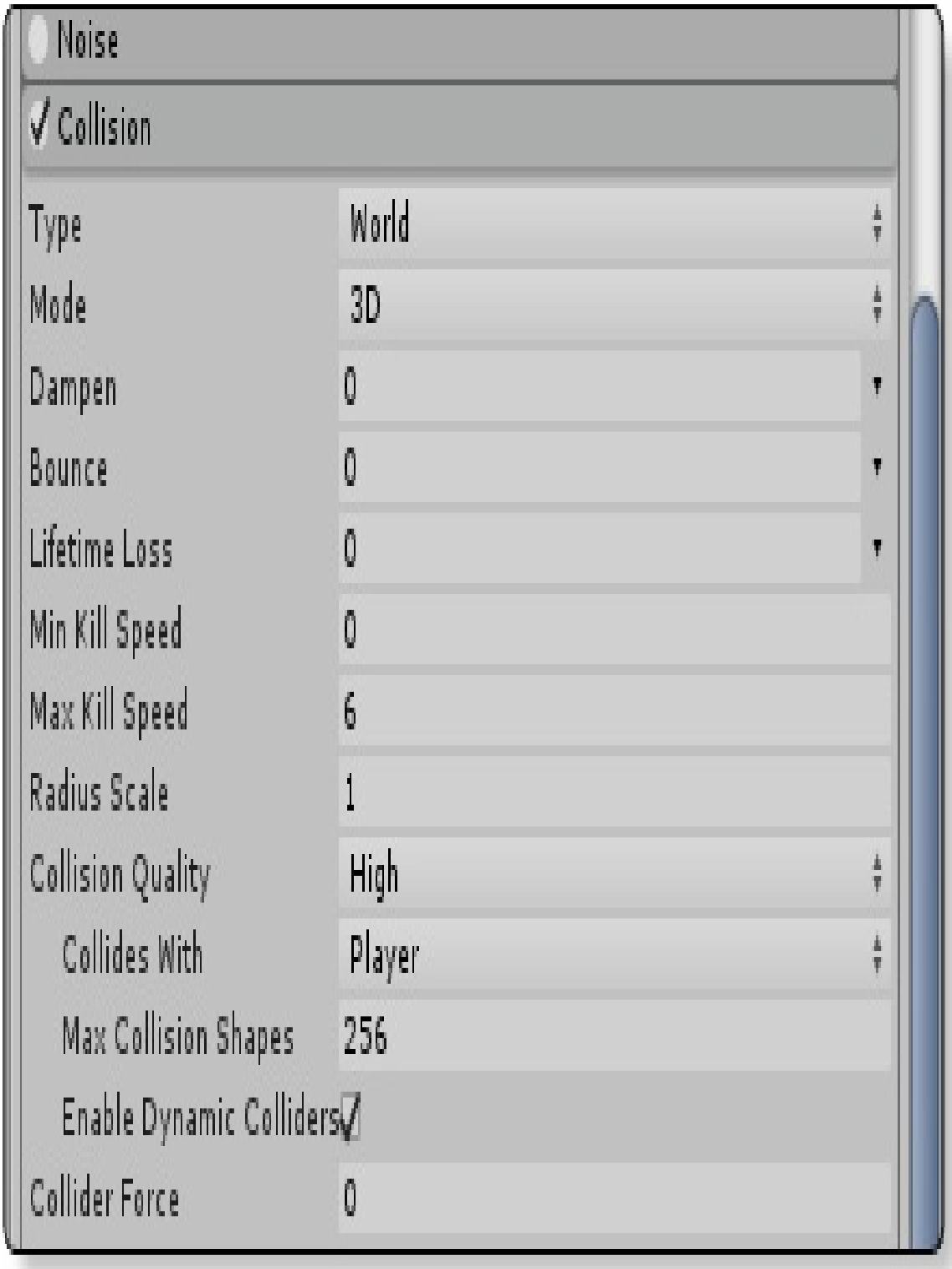
Inspector
Player
Static
Tag
Player
Layer
Player
Prefab
Select
Revert
Apply

Asignación de layer Player

NOTA:

El layer Player no hay que crearlo, ya existe por Default.

Seleccione el objeto “Rock_PS”, vaya a su componente “ParticleSystem”, expanda la pestaña “Collision” y asigne el layer “Player” a la colisión.


Noise
Collision
Type
World
Mode
3D
Dampen
0
Bounce
0
Lifetime Loss
0
Min Kill Speed
0
Max Kill Speed
6
Radius Scale
1
Collision Quality
High
Collides With
Player
Max Collision Shapes
256
Enable Dynamic Colliders
Collider Force
0

Asignación de layer para colisión

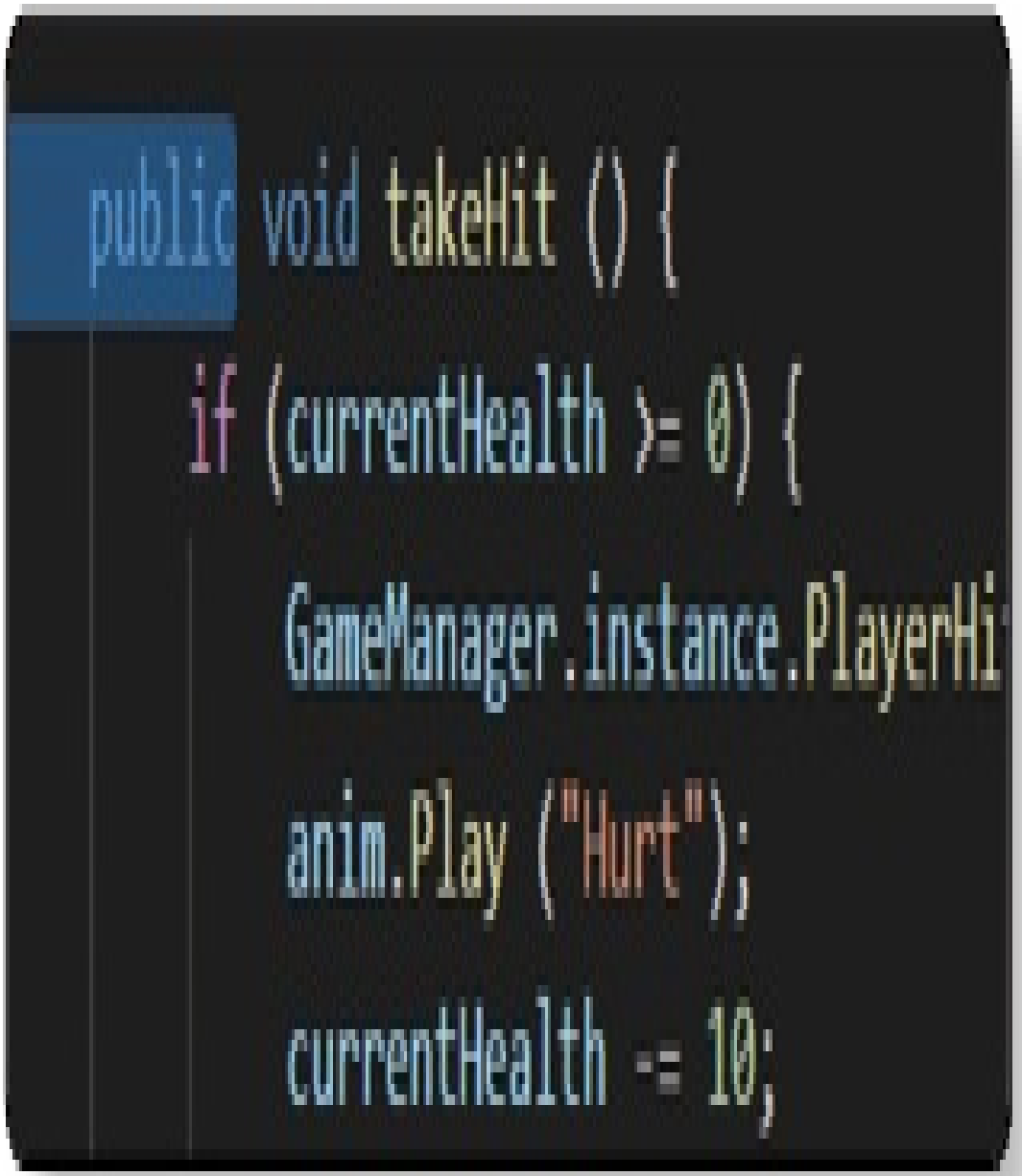
Previsualice el juego para observar que cuando las rocas colisionan con el “Player” se imprime en la consola “Player Hit”.

Player Hit
UnityEngine.MonoBehaviour:print(Object)

Comprobación de colisión en consola

A continuación, vamos a programar el daño al “Player”, vamos a mandar llamar la función “TakeHit”, pero hasta este momento es privada, vamos a hacerla pública.

Vaya al script “PlayerHealth” y haga pública la función “TakeHit”.


public void takeHit () {
if (currentHealth >= 0) {
GameManager.instance.PlayerHi
anim.Play ("Hurt");
currentHealth -= 10;

Hacer pública una función

Vaya al script “RockCollision” y añada las siguientes líneas de código:

```
public class RockCollision : MonoBehaviour {

    private GameObject player;
    private PlayerHealth playerHealth;

    // Use this for initialization
    void Start () {
        player = GameManager.instance.Player;
        playerHealth = player.GetComponent<PlayerHealth> ();
    }

    // Update is called once per frame
    void Update () {

    }

    void OnParticleCollision (GameObject player ) {
        playerHealth.takeHit ();
        Debug.Log ("Player Hit");
    }
}
```

Código para hacer daño al Player

Previsualice el juego para observar que las rocas le hacen daño a nuestro “Player”.

Previsualización de daño al Player

Programación aleatoria de los ataques

En este apartado conoceremos y utilizaremos una nueva estructura de control llamada “Switch / Case” para programar los ataques de forma aleatoria.

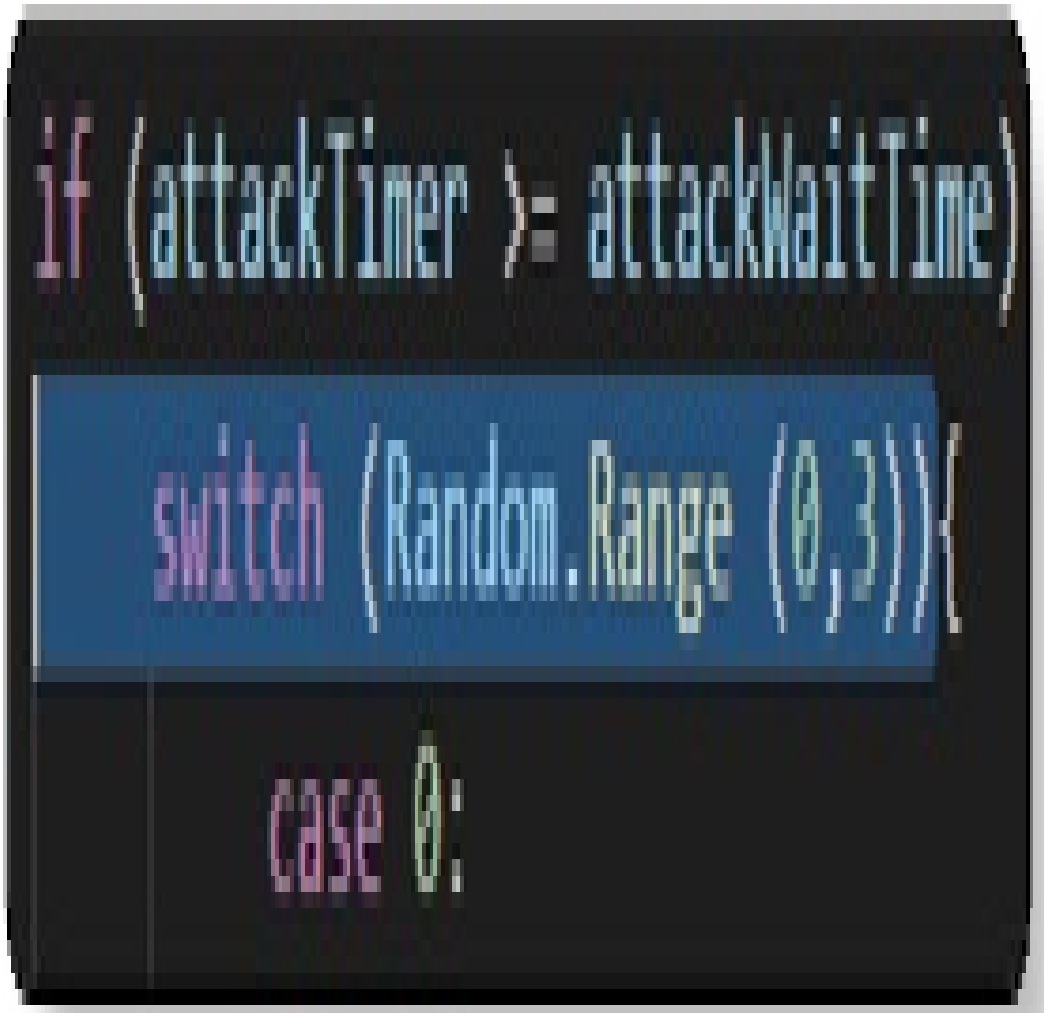
Con la estructura de control “Switch / Case” podemos encapsular diversos comportamientos y/o acciones para que sean llamadas de manera aleatoria.

En el script “BossController” escriba el siguiente código:

```
if (inBattle) {
    if(!attacking) {
        idleTimer += Time.deltaTime;
    } else {
        idleTimer = 0.0f;
        attackTimer += Time.deltaTime;
        if (attackTimer >= attackWaitTime) {
            switch (Random.Range(0,3)) {
                case 0:
                    BossAttack01();
                    break;
                case 1:
                    BossAttack02();
                    break;
                case 2:
                    BossAttack03();
                    break;
                default: break;
            }
        }
```

Estructura Switch/Case

Para inicializar una estructura “Switch / Case” debemos de llamar el comando “switch” seguido de la declaración de un rango aleatorio.


if (attackTiner >= attackWaitTime)
switch (Random.Range (0,3)){
case 0:

Declaración de switch

Seguido de la declaración de los “Cases” que van a tener como contenido los comportamientos o funciones de cada uno de ellos.

```
switch (Random.Range (0
    case 0:
    BossAttack01();
    break;
    case 1:
    BossAttack02();
    break;
    case 2:
    BossAttack03();
    break;
    default: break;
```

Declaración de “Cases”

Como nomenclatura debemos de asignar el comando “break” después de cada “case”.

Finalizando la estructura con el código “default: break; “.

Previsualice el juego para observar el cambio aleatorio de los ataques del jefe

A continuación, queremos incrementar la velocidad de los ataques a medida que vaya bajando la salud de nuestro jefe.

Vaya al script “BossController” y cambie el valor de la variable que se muestran.

public float attackTimer = 0.0f;
public float attackWaitTime = 4.0f;

Cambio de valor a variable

Escriba la siguiente estructura condicional para que conforme baje la salud del jefe, se incremente la velocidad de los ataques.

```
else {
    idleTimer = 0.0f;
}
    if (bossHealth.bossHealth > 0 && playerHealth.CurrentHealth > 0){
        if (bossHealth.bossHealth > 15){
            attackWaitTime = 4.0f;
        }
        if (bossHealth.bossHealth > 10 && bossHealth.bossHealth < 16){
            attackWaitTime = 3.0f;
        }
        if (bossHealth.bossHealth > 5 && bossHealth.bossHealth < 11){
            attackWaitTime = 2.0f;
        }
        if (bossHealth.bossHealth >= 1 && bossHealth.bossHealth < 6){
            attackWaitTime = 1.0f;
        }
    }
```

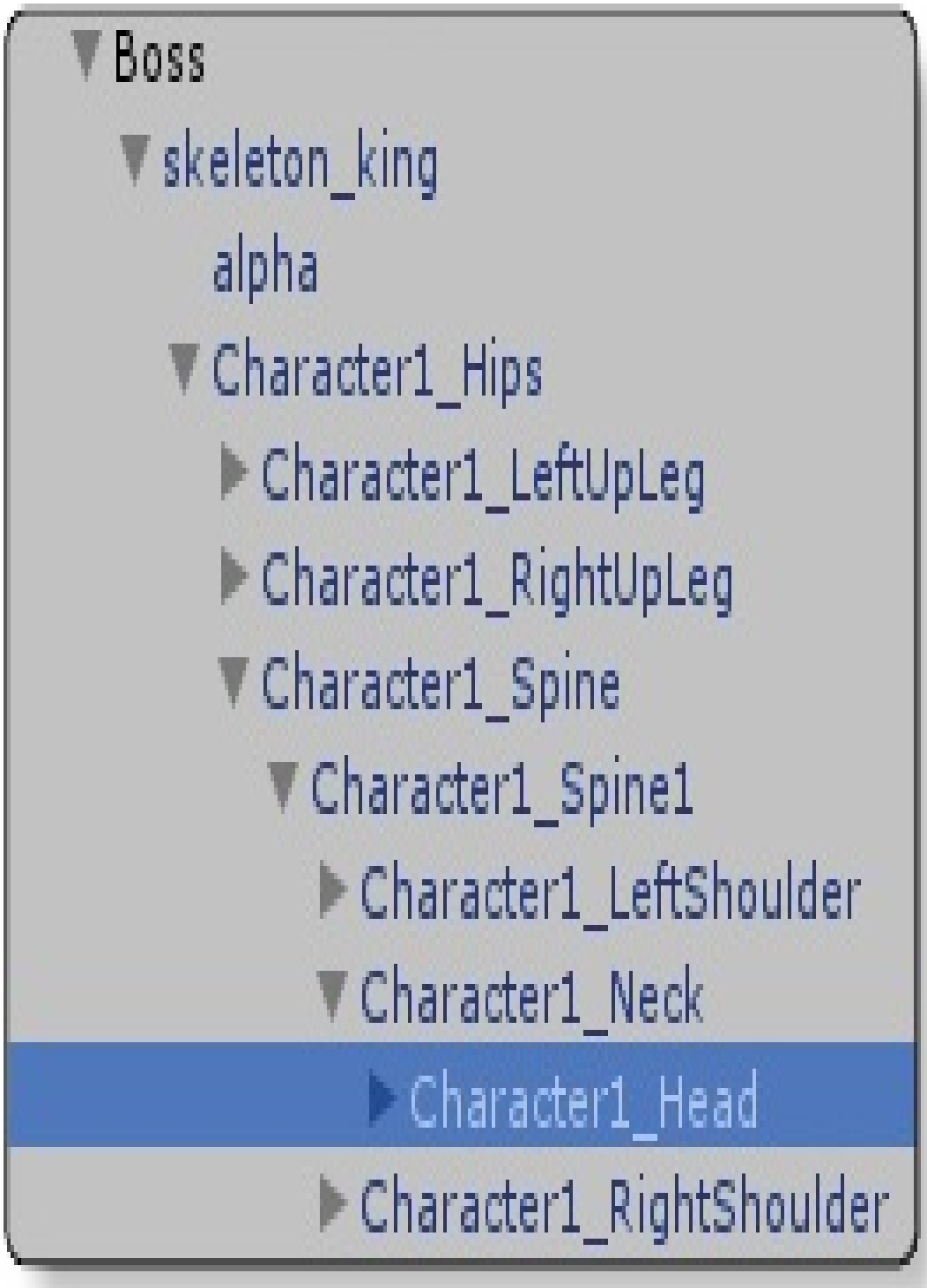
Código de estructura condicional en “BossController”

Previsualice el juego para observar que conforme baja la salud del jefe, se incrementa la velocidad de los ataques.

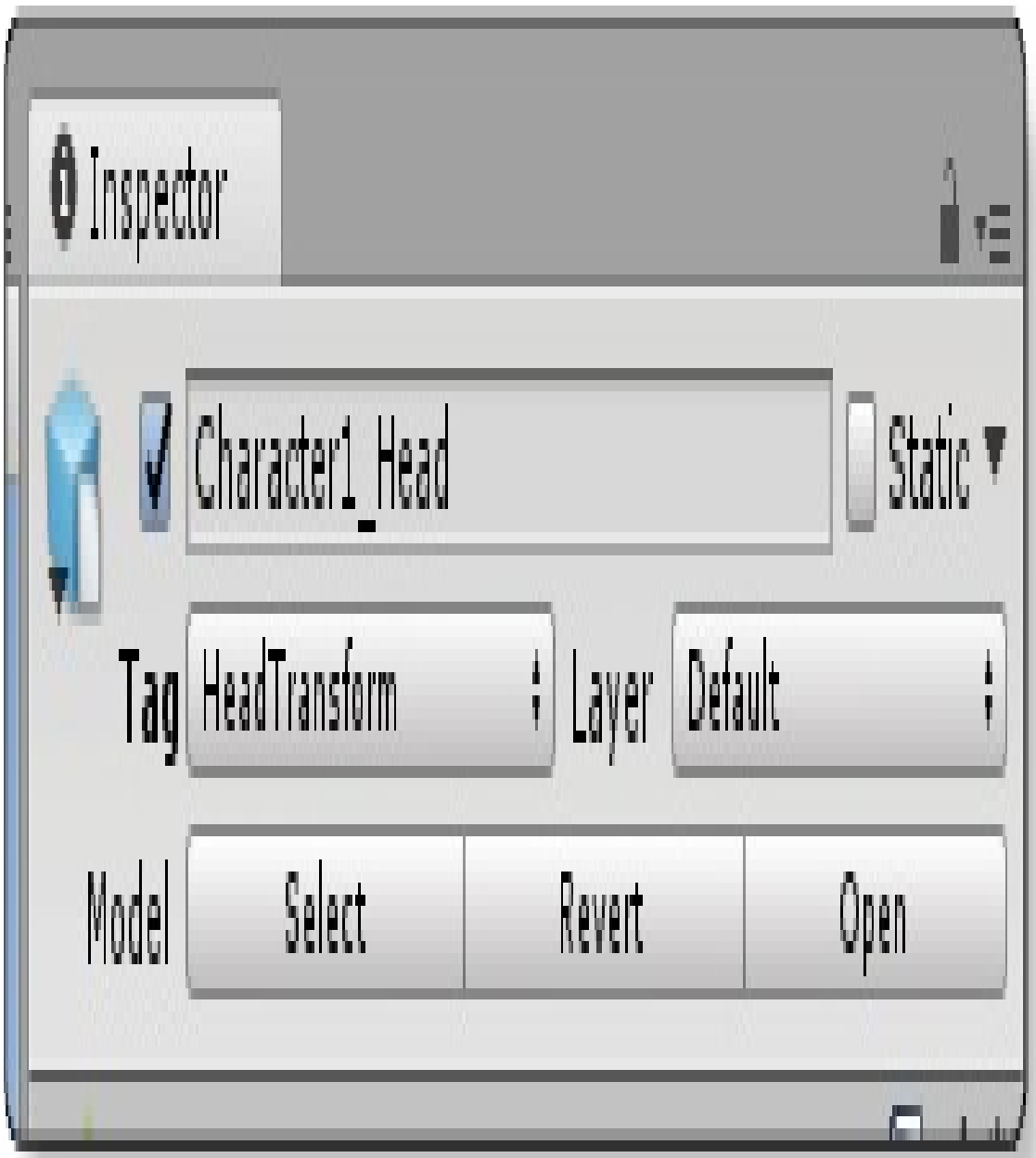
Uno de los detalles importantes hasta el momento es que el objeto “HeadTrigger” no sigue el movimiento de la cabeza del jefe.

A continuación, vamos a solucionar ese detalle.

Haga un nuevo “Tag” y nómbrelo “HeadTransform” y asígnelo al joint relativo a la cabeza de nuestro jefe.


Boss
skeleton_king
alpha
Character1_Hips
Character1_LeftUpLeg
Character1_RightUpLeg
Character1_Spine
Character1_Spine1
Character1_LeftShoulder
Character1_Neck
Character1_Head
Character1_RightShoulder

Joint relativo a la cabeza


Inspector
Character1_Head
Static
Tag
HeadTransform
Layer
Default
Model
Select
Revert
Open

Tag “HeadTransform”

Haga un nuevo C#Script, nómbrelo “FollowTarget” y asígnelo al objeto “HeadTrigger”.

Abra el script y escriba el siguiente código:

```
public class FollowTarget : MonoBehaviour {

    public Transform target;

    // Use this for initialization
    void Start () {

        target = GameObject.FindGameObjectWithTag("HeadTransform").transform;

    }

    // Update is called once per frame
    void FixedUpdate () {

        TrackTarget ();
    }

    void TrackTarget(){
        float targetX = target.position.x;
        float targetY = target.position.y;

        transform.position = new Vector2 (targetX, targetY);
    }
}
```

Script “FollowTarget”

Previsualice el juego para observar que el “HeadTrigger” sigue a la cabeza del jefe.

HeadTrigger

Previsualización de “FollowTarget”

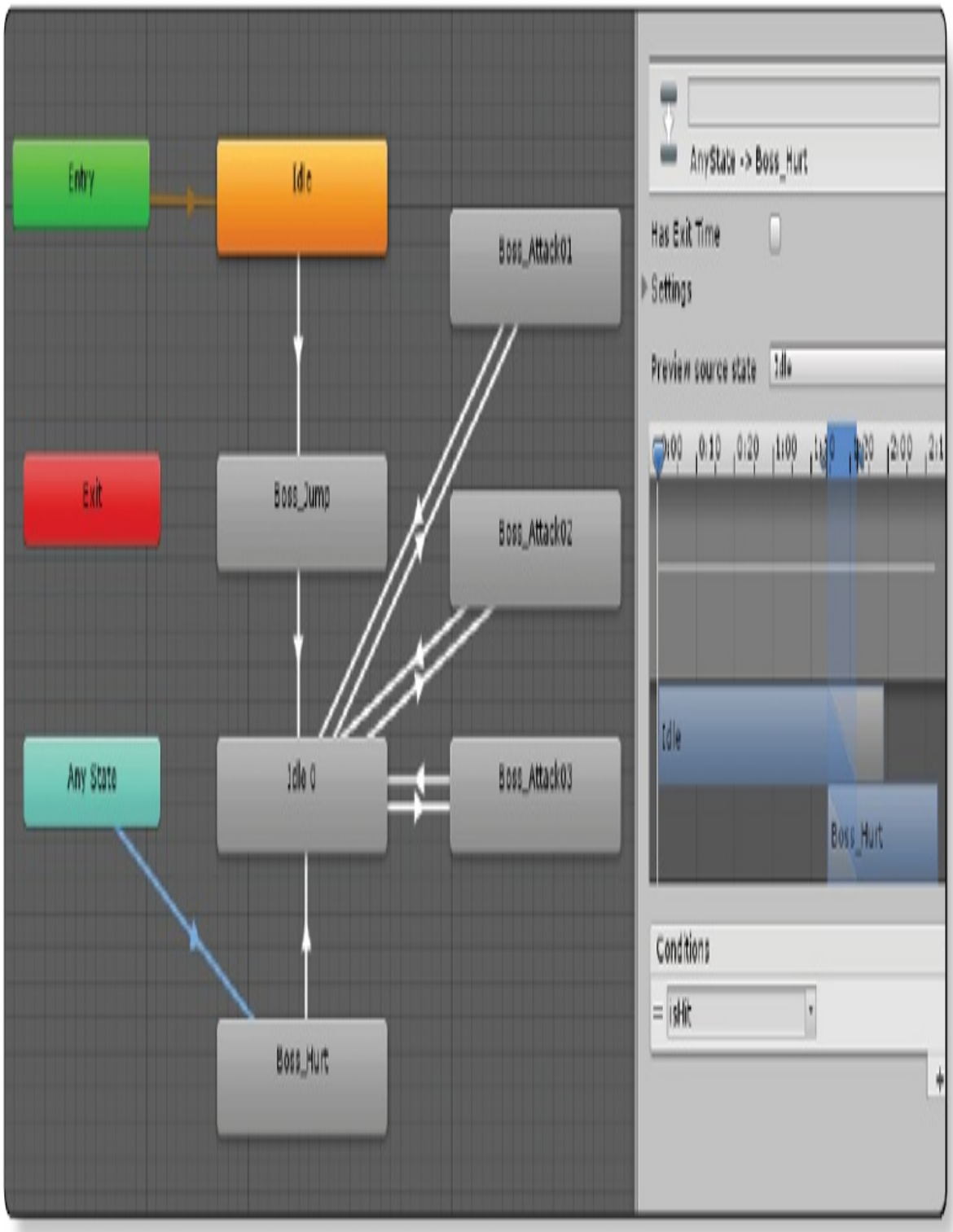
Funcionalidad de daño y muerte del jefe

En este apartado programaremos las funcionalidades y animaciones relativas al daño y a la muerte de nuestro jefe cuando sea alcanzado por el proyectil de nuestro “Player” y cuando se acabe su salud.

A su vez nos ocuparemos de desactivar funciones del jefe al momento de su muerte.

Seleccione el objeto “Boss”, abra la ventana “Animator” y haga un nuevo parámetro tipo “Trigger” y nómbrelo “isHit”.

Asigne la animación “Boss_Hurt” como se muestra con sus transiciones y asigne el parámetro “isHit” en la transición de “AnyState” a “Boss_Hurt”.


Entry
Idle
Boss_Attack01
Exit
Boss_Jump
Boss_Attack02
Any State
Idle 0
Boss_Attack03
Boss_Hurt
AnyState -> Boss_Hurt
Has Exit Time
Settings
Preview source state
Idle
Boss_Hurt
Conditions

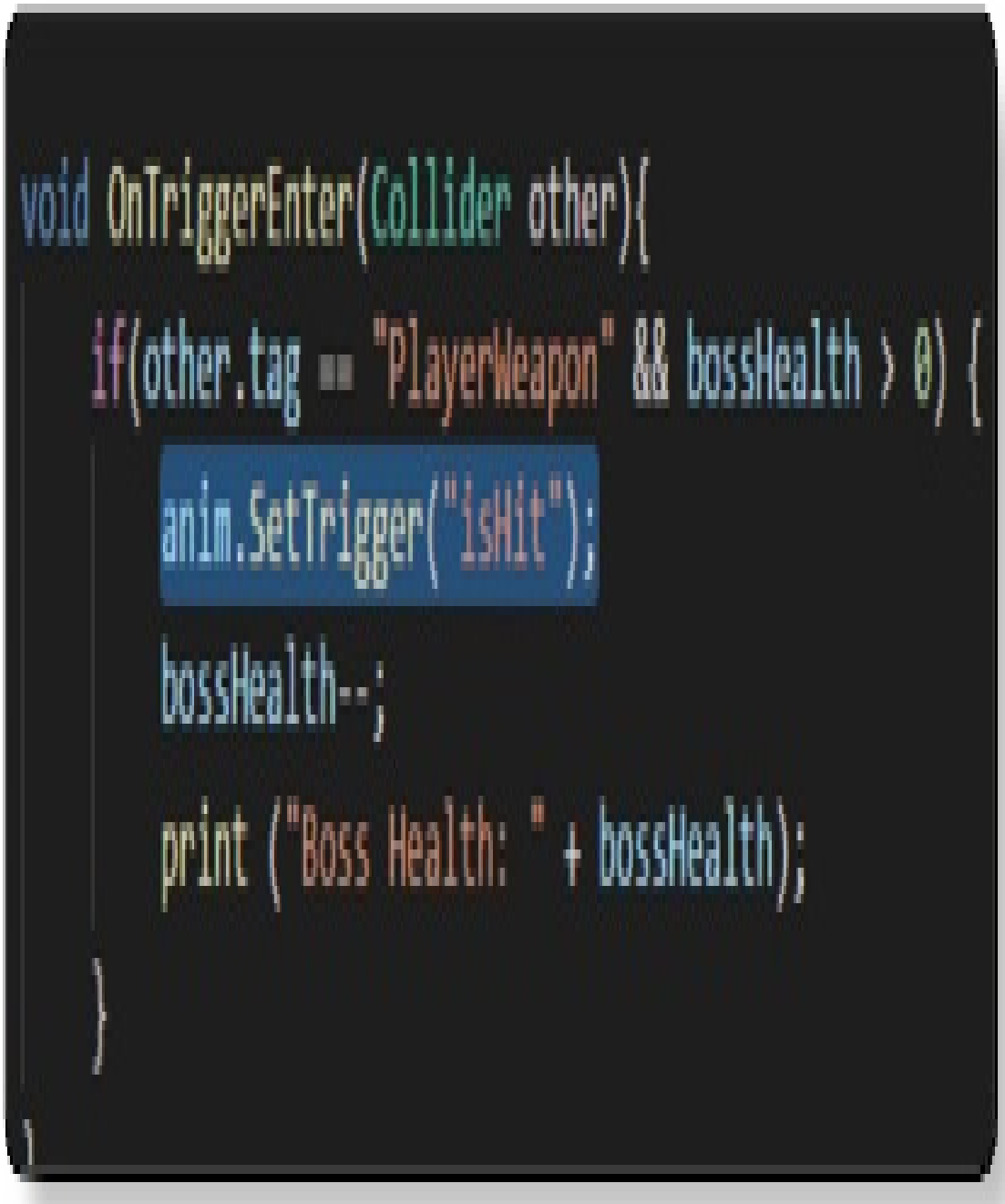
Asignación de animación de daño

Vaya al script “BossHealth”, haga la siguiente variable e inicialícela .

```
public class BossHealth : MonoBehaviour {

    public int bossHealth = 20;
    private Animator anim;

    // Use this for initialization
    void Start () {

        anim = GameObject.Find ("Boss").GetComponent<Animator>();
```

Creación e inicialización de variable

Escriba el siguiente código en la función “OnTriggerEnter” para que cuando decrezca la salud del jefe, corra el parámetro de animación “isHit”.


void OnTriggerEnter(Collider other){
if(other.tag == "PlayerWeapon" && bossHealth > 0) {
anim.SetTrigger("isHit");
bossHealth--;
print ("Boss Health: " + bossHealth);
}
}

Código para mandar llamar la animación del daño

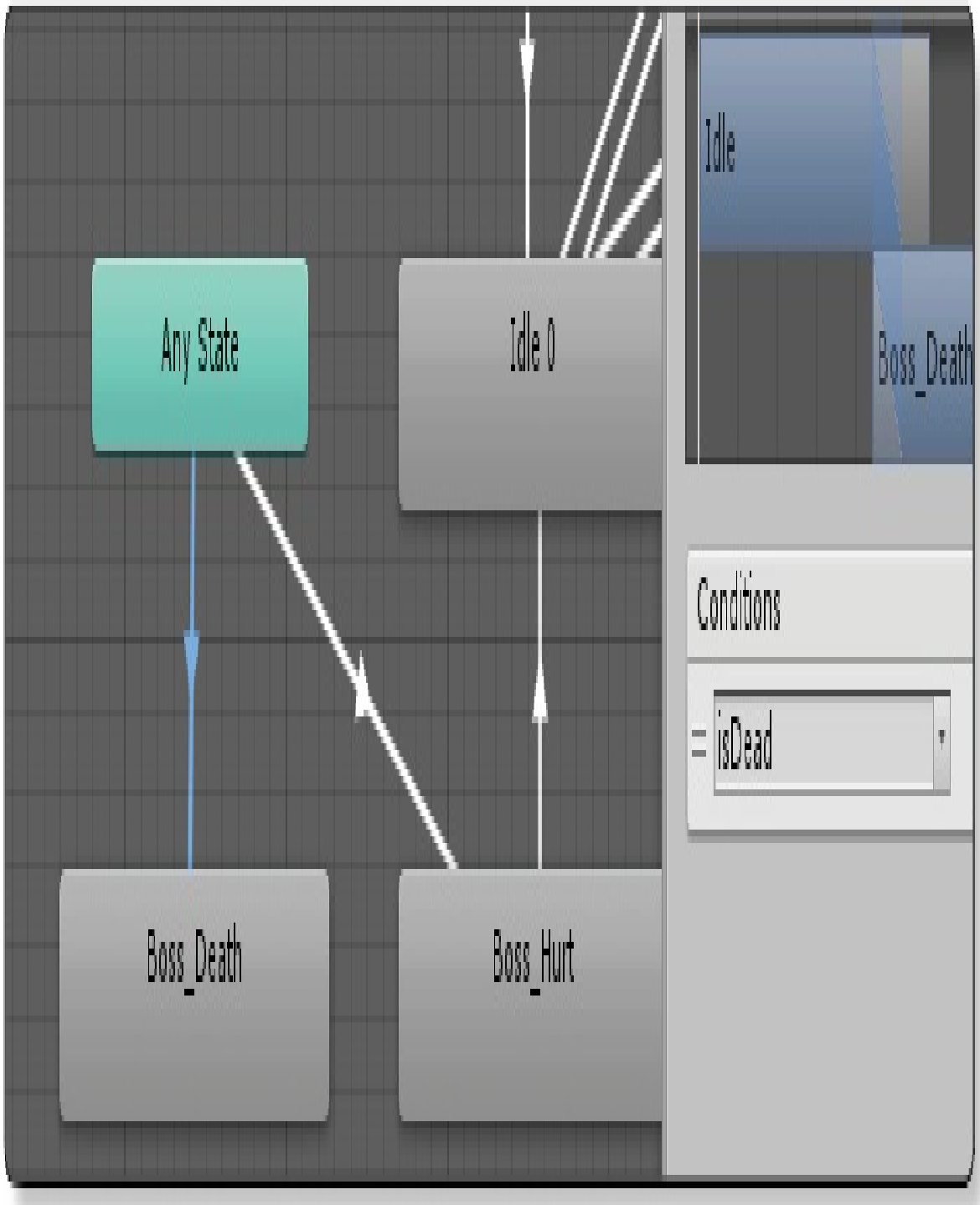
Previsualice el juego para observar la animación de “Boss_Hurt” cuando su cabeza es alcanzada por un proyectil del “Player”.

Previsualización de animación de daño

A continuación, programaremos la funcionalidad y animación de la muerte del jefe.

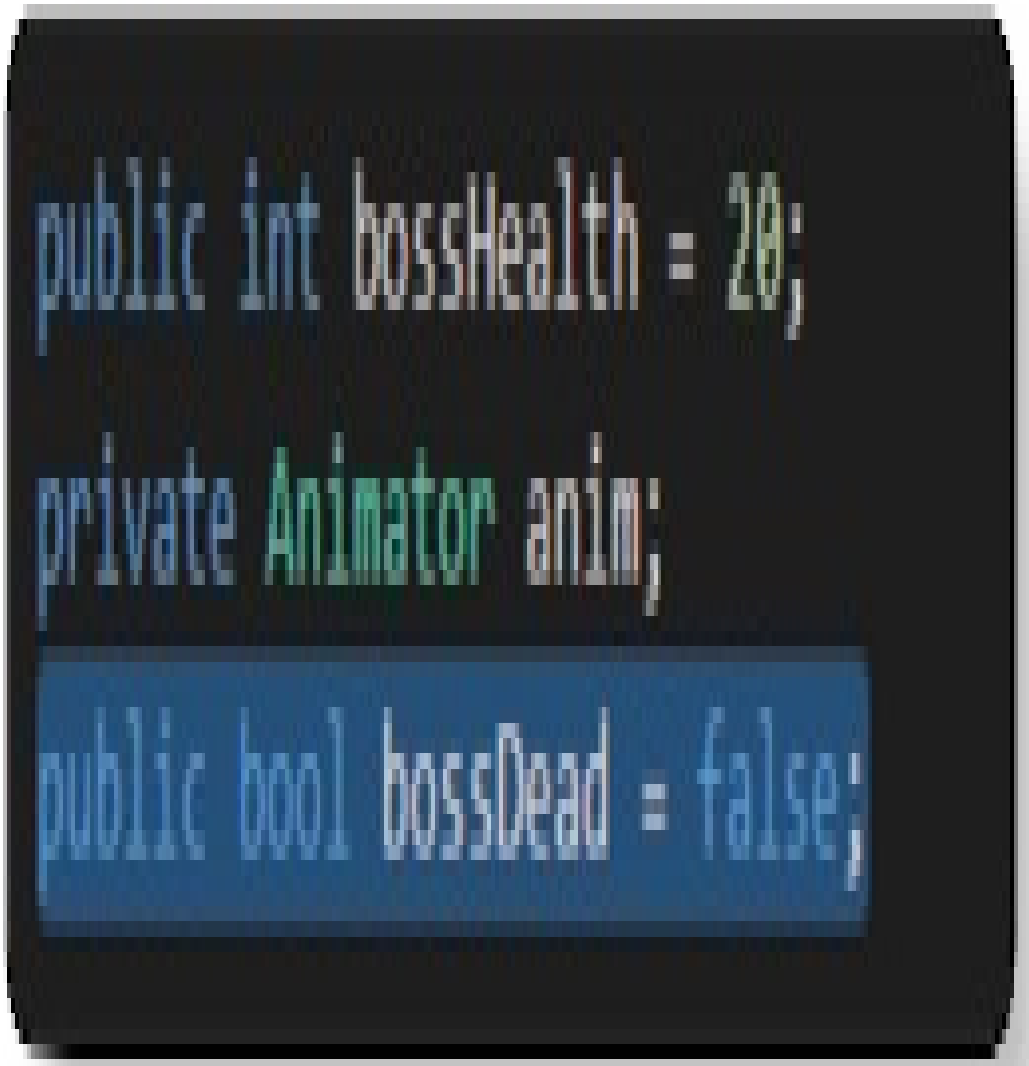
Vaya a Animator con “Boss” seleccionado, haga un nuevo parámetro tipo “Trigger” y nómbrelo “isDead”.

Asigne el clip de animación “Boss_Death” y asigne el parámetro “isDead” en la transición de “AnyState” a “Boss_Death”.


Idle
Boss_Death
Any State
Idle 0
Conditions
isDead
Boss_Death
Boss_Hurt

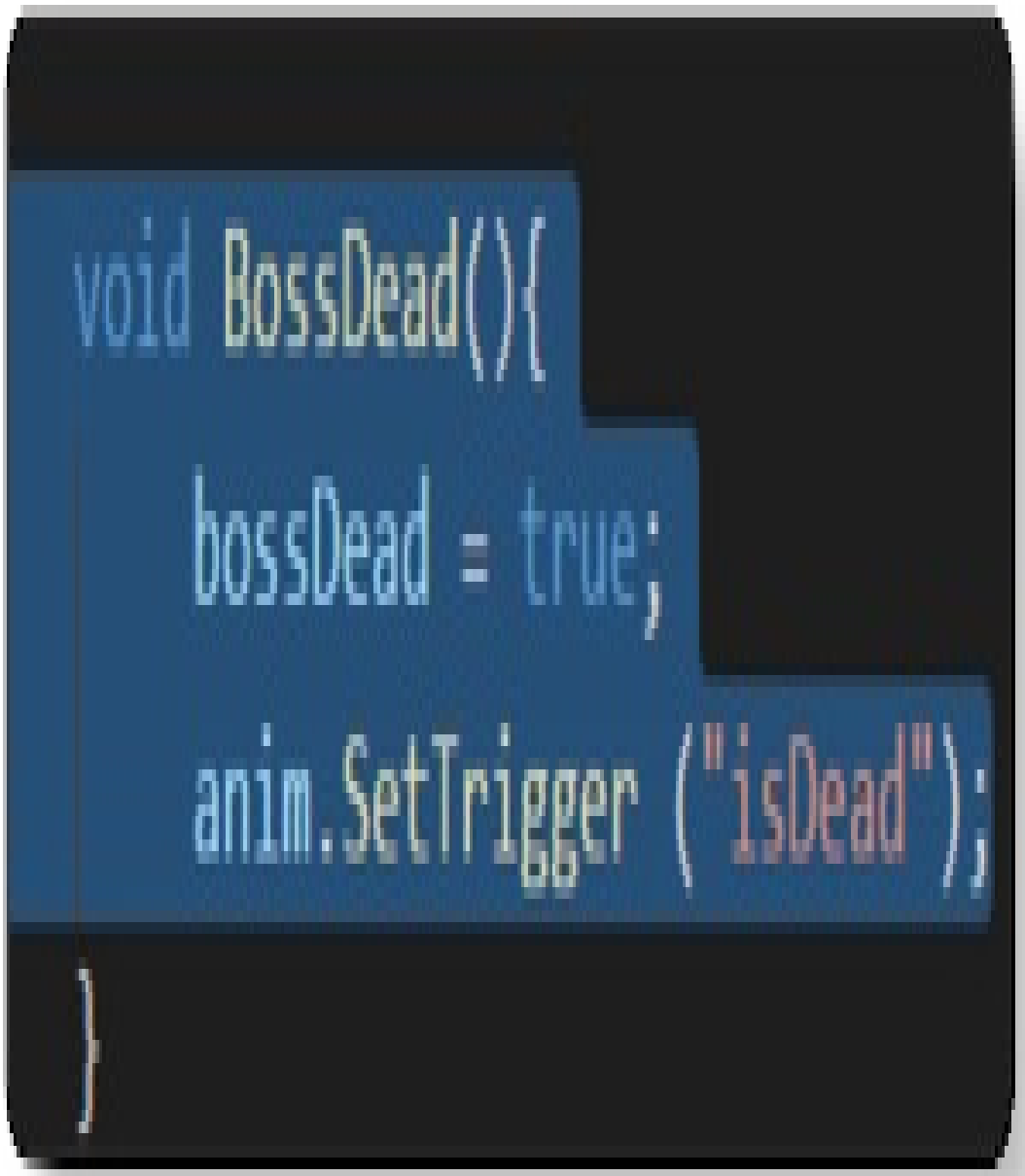
Asignación de animación de muerte

Vaya al script “BossHealth” y haga la siguiente variable.


public int bossHealth = 20;
private Animator anim;
public bool bossDead = false;

Creación de variable booleana

Escriba la siguiente función.


void BossDead(){
bossDead = true;
anim.SetTrigger ("isDead");
}

Creación de función “BossDead”

Mande llamar la función creada en la función “OnTriggerEnter”.

void OnTriggerEnter(Collider other){
if(other.tag == "PlayerWeapon" && t
anim.SetTrigger("isHit");
bossHealth--;
print ("Boss Health: " + bossHe
} else {
BossDead();
}
}

Mandar llamar la función “BossDead”

Previsualice el juego para observar la animación de muerte del jefe cuando su salud llega a cero.

Previsualización de la muerte del jefe

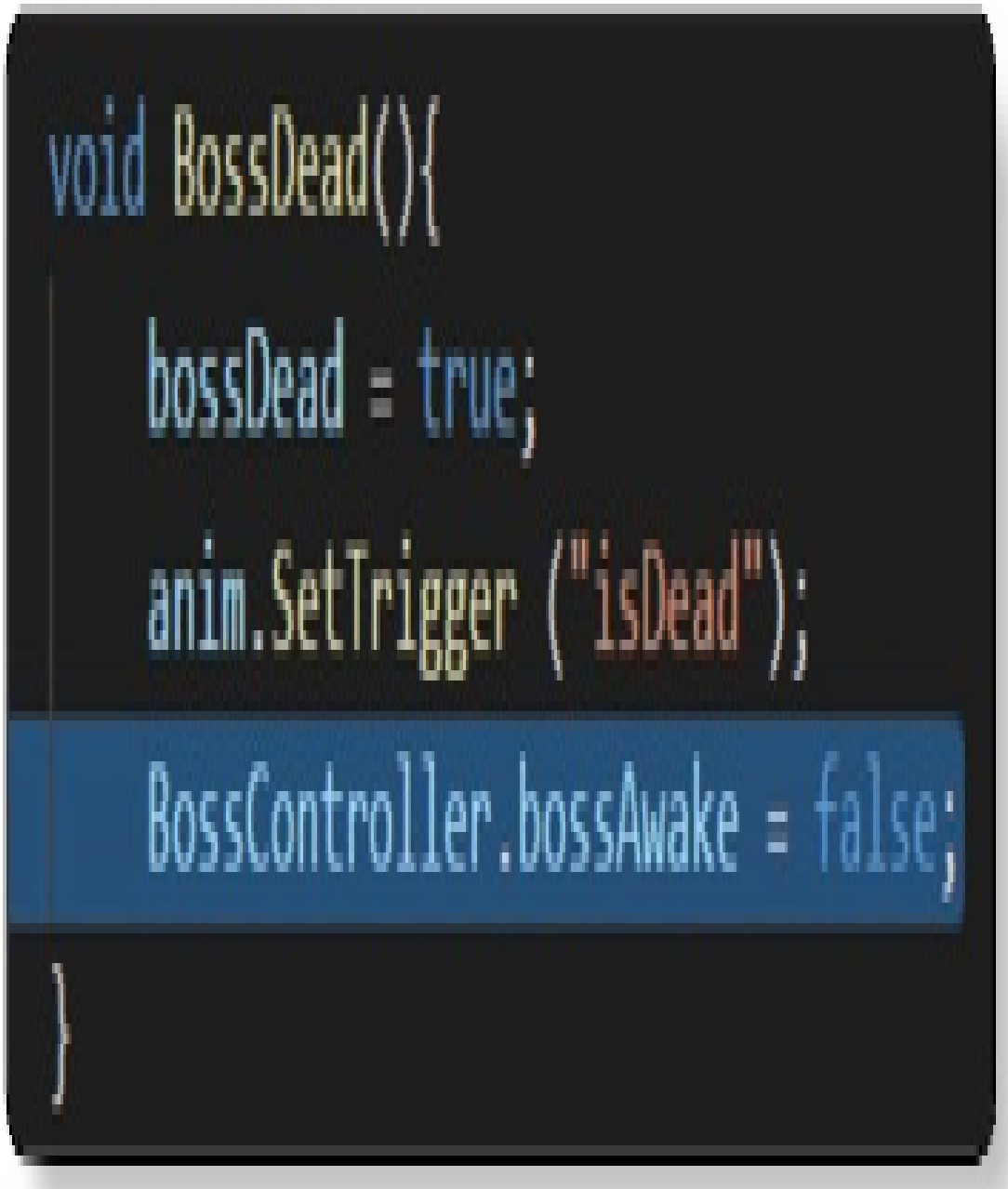
PROBLEMA: Cuando muere nuestro jefe, su sistema de ataque continúa corriendo y sus “Colliders” permanecen activos causando que, si nuestro “Player” toca la espada, éste obtendrá daño.

SOLUCIÓN: Para detener el sistema de ataque cuando muere nuestro jefe, vaya al script “BossHealth”, haga e inicialice las siguientes variables.

```
public bool bossDead = false;
public BossController BossController;


// Use this for initialization
void Start () {

    anim = GameObject.Find ("Boss").GetComponent<Animator>();
    BossController = GameObject.Find ("Boss").GetComponent<BossController>();
}
```

Creación e inicialización de variables

Vaya a la función “BossDead” y escriba el siguiente código:


void BossDead(){
bossDead = true;
anim.SetTrigger ("isDead");
BossController.bossAwake = false;
}

Código para cambiar el valor de variable booleana

Para deshabilitar sus “Colliders” haga e inicialice las variables que se muestran.

```
public bool bossDead = false;
public BossController bossController;

private CapsuleCollider capsuleCollider;
private BoxCollider weaponCollider;
private SphereCollider triggerCollider;

// Use this for initialization
void Start () {

    anim = GameObject.Find ("Boss").GetComponent<Animator>();
    bossController = GameObject.Find ("Boss").GetComponent<BossController>();
    capsuleCollider = GameObject. Find("Boss").GetComponent<CapsuleCollider>();
    weaponCollider = GameObject.Find("Boss").GetComponentInChildren<BoxCollider>();
    triggerCollider =GameObject.Find("Boss").GetComponentInChildren<SphereCollider>();
}
```

Creación e inicialización de variables

Vaya a la función “BossDead” y escriba el siguiente código:

```
void BossDead (){
    bossDead = true;
    anim.SetTrigger ("isDead");
    bossController.bossAwake = false;
    capsuleCollider.enabled = false;
    weaponCollider.enabled = false;
    triggerCollider.enabled = false;
}
```

Código para deshabilitar Colliders

Previsualice el juego para observar que cuando muere el jefe:

Se detiene su sistema de ataque

Los “Colliders” del jefe se deshabilitan

Con los conceptos aprendidos deshabilite el Sistema de partículas al momento de la muerte del Jefe

Detalles Finales

En este apartado nos ocuparemos de los siguientes detalles finales:

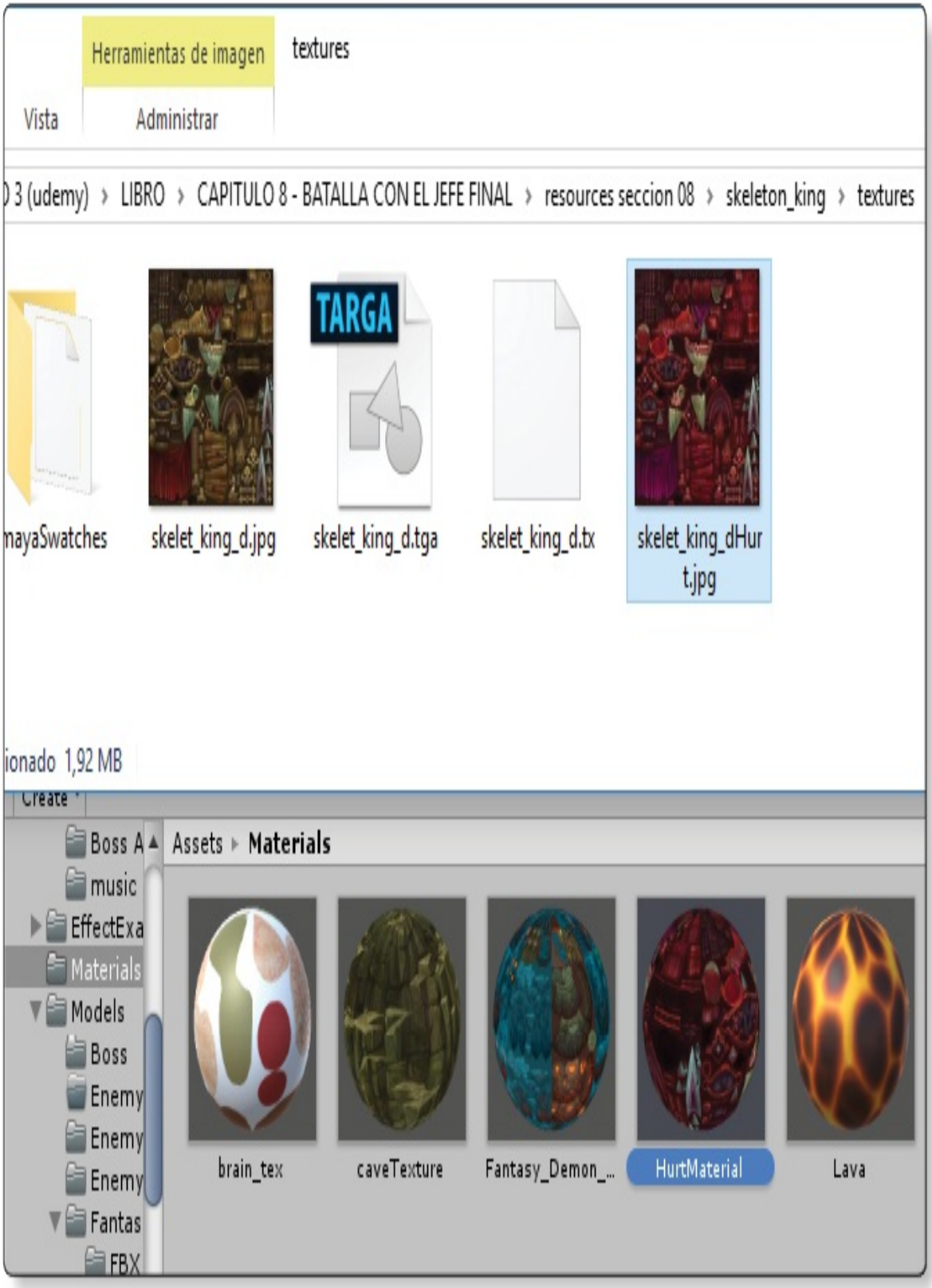
Cambiaremos el material del jefe cuando se haya bajado considerablemente su salud

Cambiaremos la música del juego cuando comience la batalla final

El estudiante pondrá en práctica los conceptos aprendidos creando elementos extras a su juego

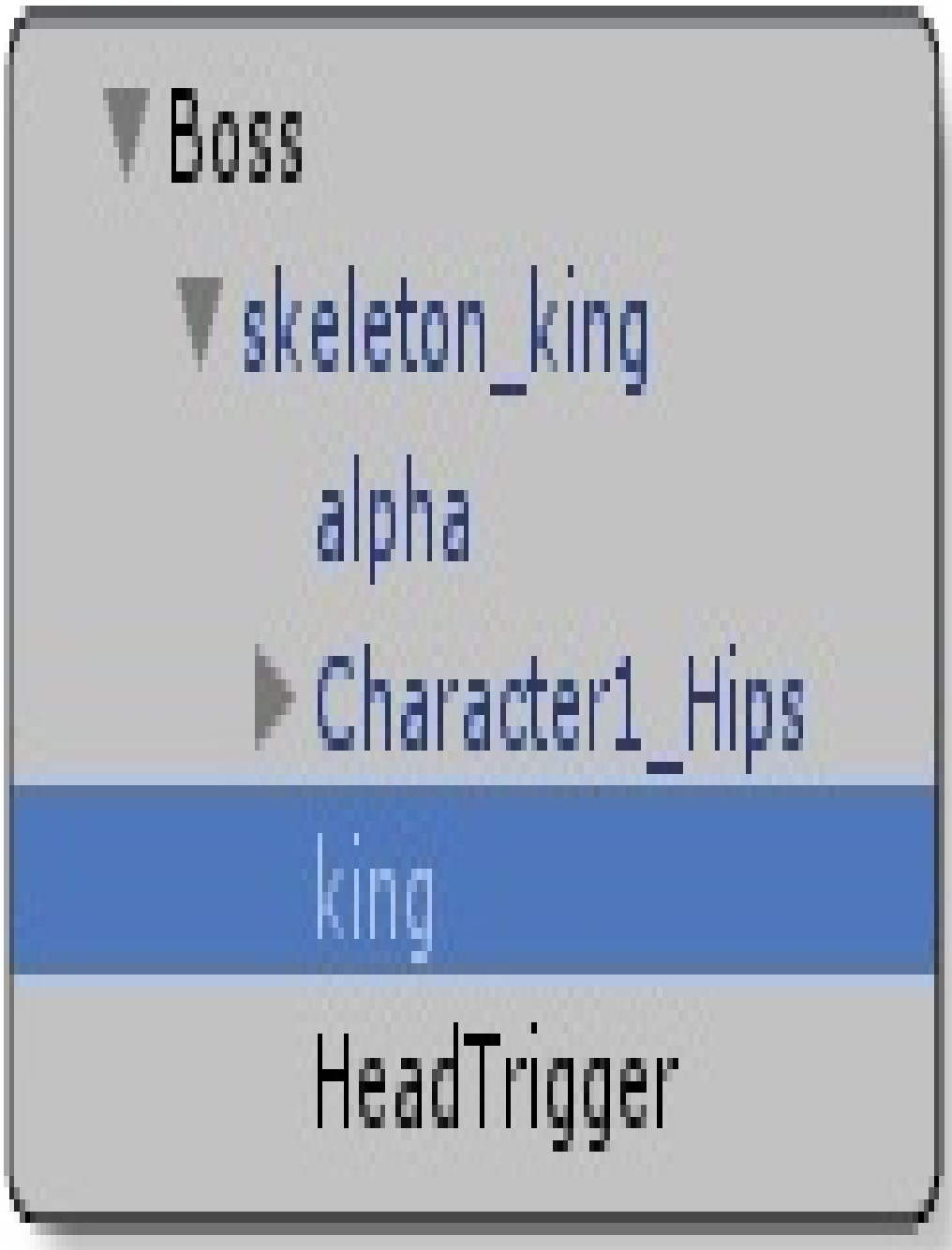
Vamos a cambiar el material del jefe cuando haya recibido daño considerable

Con los conceptos aprendidos haga un nuevo material, nómbrelo “HurtMaterial” con la textura “skelet_king_dHurt.jpg” de la carpeta “resources”.

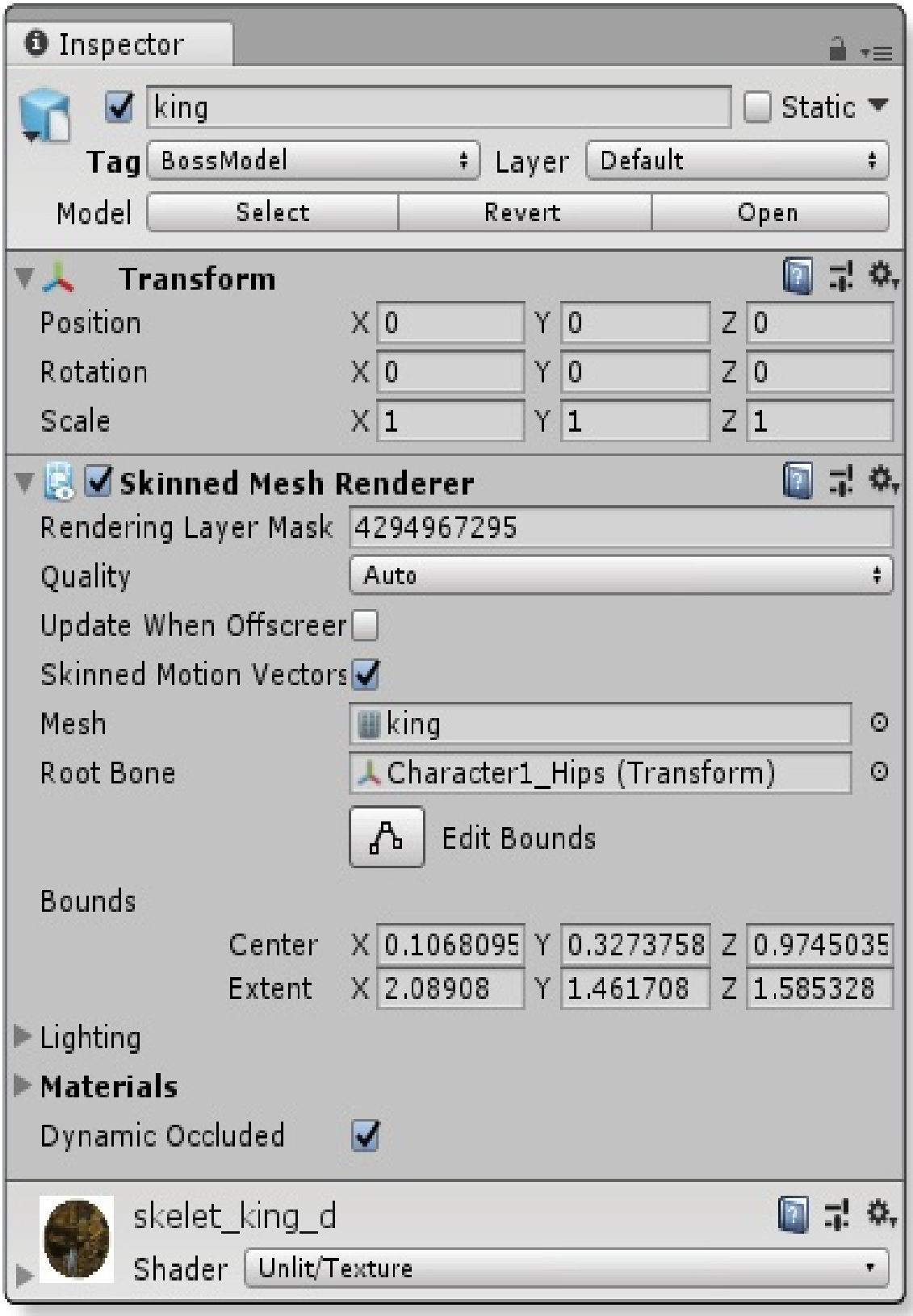

Herramientas de imagen
textures
Vista
Administrar
) 3 (udemy) › LIBRO › CAPITULO 8 - BATALLA CON EL JEFE FINAL › resources seccion 08 › skeleton_king › textures
TARGA
mayaSwatches
skelet_king_d.jpg
skelet_king_d.tga
skelet_king_d.tx
skelet_king_dHurt.jpg
ionado 1,92 MB
Create
Boss A
music
EffectExa
Materials
Models
Boss
Enemy
Enemy
Enemy
Fantas
FBX
Assets ▸ Materials
brain_tex
caveTexture
Fantasy_Demon_...
HurtMaterial
Lava

Creación de material para el daño

Haga un nuevo “Tag”, nómbrelo “BossModel” y asígnelo al objeto “king” hijo de “skeleton_king”.


Boss
skeleton_king
alpha
Character1_Hips
king
HeadTrigger

Selección de objeto “king”


Inspector
king
Static
Tag
BossModel
Layer
Default
Model
Select
Revert
Open
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Skinned Mesh Renderer
Rendering Layer Mask
4294967295
Quality
Auto
Update When Offscreer
Skinned Motion Vectors
Mesh
king
Root Bone
Character1_Hips (Transform)
Edit Bounds
Bounds
Center
X 0.1068095
Y 0.3273758
Z 0.9745035
Extent
X 2.08908
Y 1.461708
Z 1.585328
Lighting
Materials
Dynamic Occluded
skelet_king_d
Shader
Unlit/Texture

Valores de objeto “king”

Hemos asignado el “Tag” a este objeto ya que contiene el componente “SkinnedMeshRenderer” y el material que vamos a cambiar con código.

Vaya al script “BossHealth”, escriba e inicialice las variables que se muestran.

```
private BoxCollider weaponCollider;
private SphereCollider triggerCollider;

public Material hurtBossMaterial;
private GameObject bossModel;


// Use this for initialization
void Start () {

    anim = GameObject.Find ("Boss").GetComponent<Animator>();
    bossController = GameObject.Find ("Boss").GetComponent<BossCont
    capsuleCollider = GameObject. Find("Boss").GetComponent<Capsule
    weaponCollider = GameObject.Find("Boss").GetComponentInChildren
    triggerCollider =GameObject.Find("Boss").GetComponentInChildren
    bossModel = GameObject.FindGameObjectWithTag("BossModel");
```

Creación e inicialización de variables

Vaya a la función “OnTriggerEnter” y escriba la siguiente estructura condicional para cambiar el material del jefe cuando su salud sea menor de 6.

```
void OnTriggerEnter (Collider other){
    if (other.tag == "PlayerWeapon" && bossHealth > 0){
        anim.SetTrigger("isHit");
        bossHealth --;
        print ("Boss Health" + bossHealth);
        if (bossHealth < 6) {
            bossModel.GetComponent<SkinnedMeshRenderer>().material = hurtBossMaterial;
        }
    } else {
        BossDead();
    }
}
```

Código para cambiar el material del jefe

Vaya a Unity, seleccione el objeto “HeadTrigger” y asigne el material de daño en su componente “BossHealth”.

Boss Health (Script)
Script
BossHealth
Boss Health
20
Boss Dead
Boss Controller
None (Boss Controller)
Hurt Boss Material
skelet_king_hurt

Asignación de material de daño en script

Previsualice el juego para observar como cambia el material del jefe cuando su salud llega a 5.

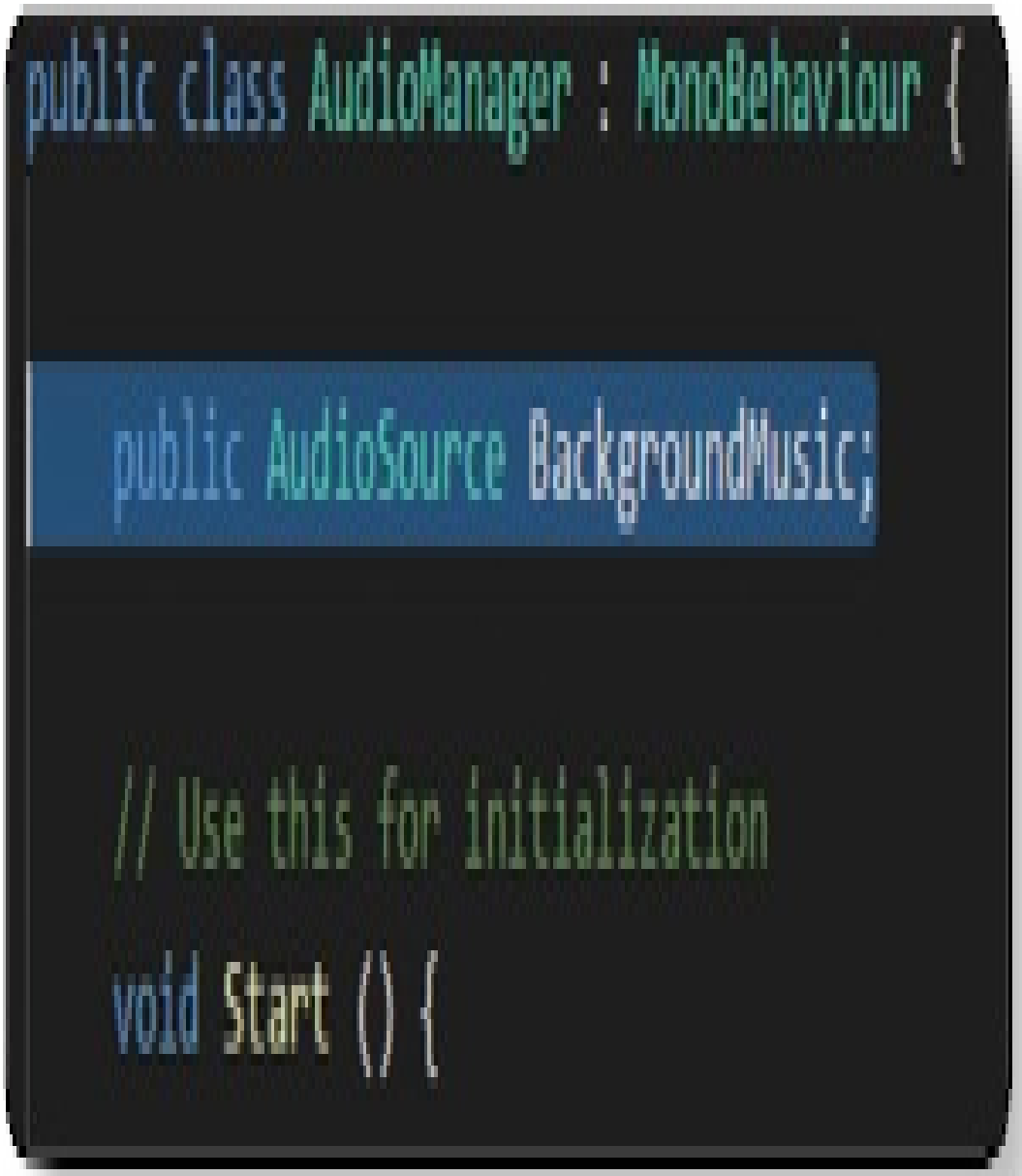
Previsualización de cambio de material

A continuación, cambiaremos la música del juego a música de la batalla final cuando nuestro “Player” cruce a través de “BossCheckPoint”.

De la carpeta de “resources” del capítulo, importe al folder de audio el archivo “bossMusic.mp3”.

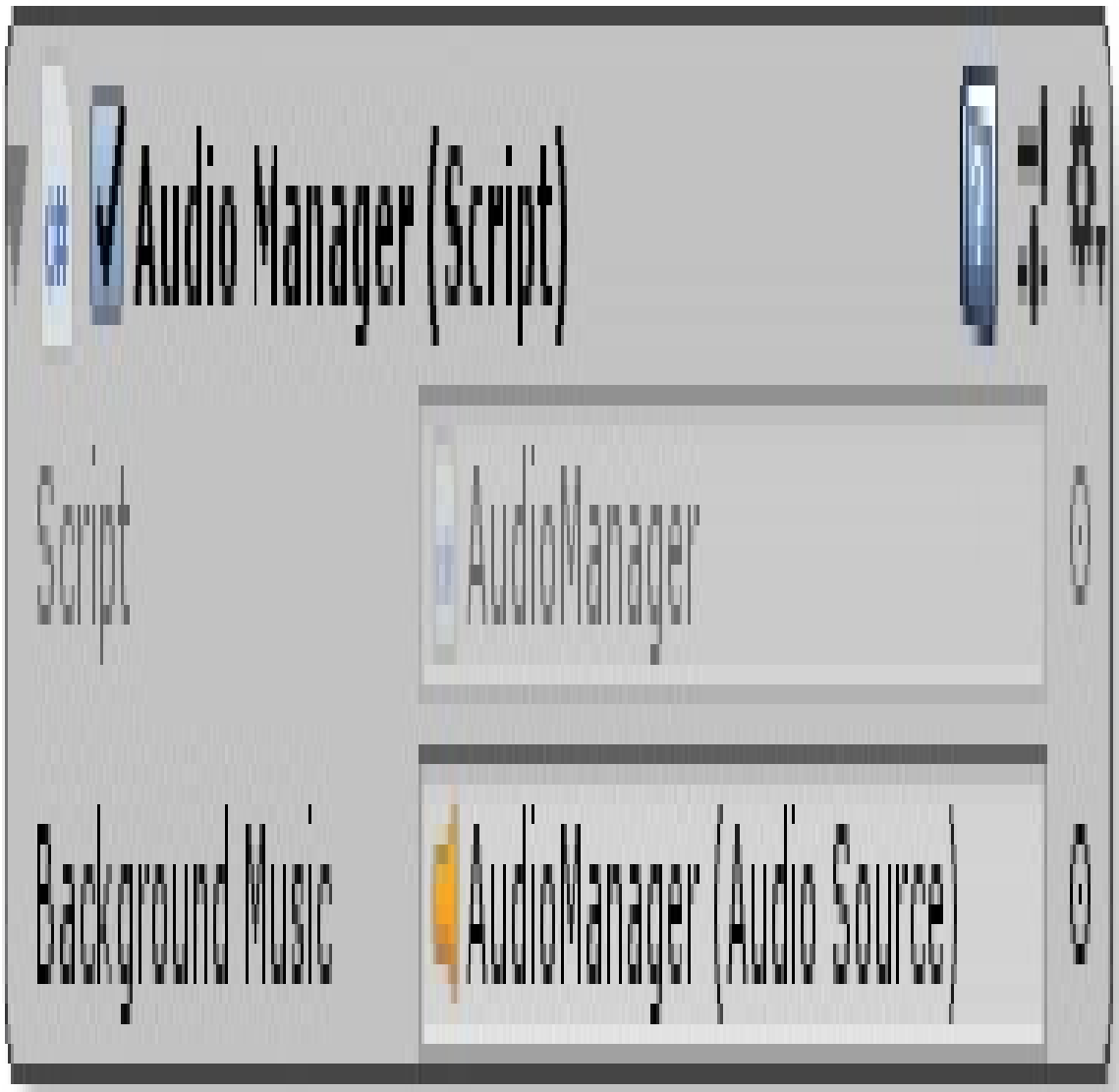
Haga un nuevo C#Script”, nómbrelo “AudioManager” y asígnelo al objeto “AudioManager”.

Abra el script “AudioManager” y escriba la variable que se muestra.


public class AudioManager : MonoBehaviour {
public AudioSource BackgroundMusic;
// Use this for initialization
void Start () {

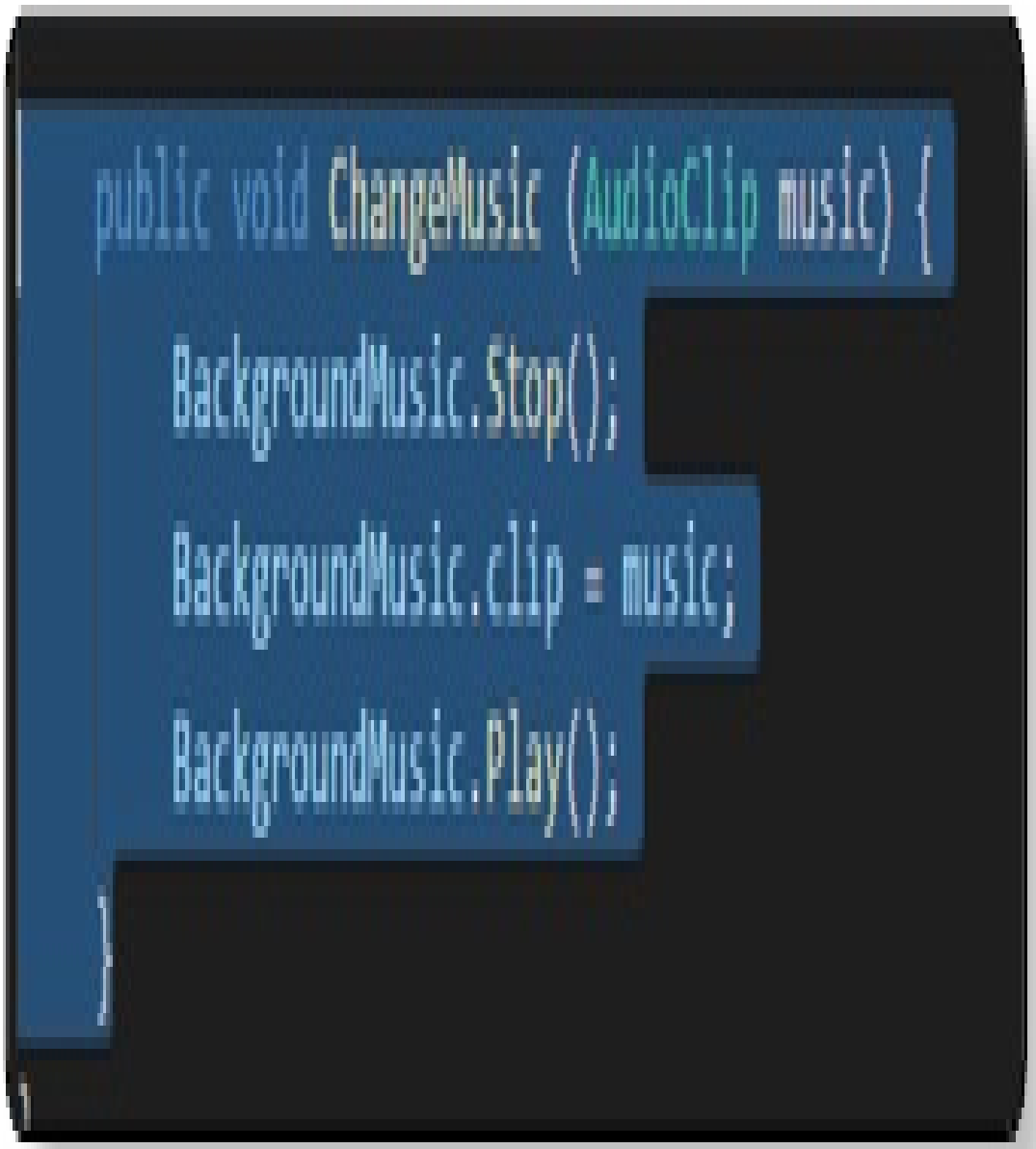
Creación de variable

Vaya a Unity, seleccione el objeto “AudioManager” y asigne el objeto “AudioManager” en el componente script.


Audio Manager (Script)
Script
AudioManager
Background Music
AudioManager (Audio Source)

Asignación de elemento “AudioManager”

Vaya al script “AudioManager” y escriba la siguiente función.


public void ChangeMusic (AudioClip music) {
BackgroundMusic.Stop();
BackgroundMusic.clip = music;
BackgroundMusic.Play();
}

Creación de función “ChangeMusic”

Vaya al script “BossCheckPoint”, escriba e inicialice las siguientes variables.

```
public AudioClip newTrack;
private AudioManager audioManager;

void Awake ()
{
    bossController = GameObject.FindGameObjectWithTag("Boss
    characterMovement = GameObject.FindGameObjectWithTag("P
    smoothFollow = GameObject.FindGameObjectWithTag ("Camer
}

// Use this for initialization
void Start () {

    collider = GetComponent<BoxCollider>();
    playerAnimator = GameObject.FindGameObjectWithTag ("Pla
    audioManager = FindObjectOfType<AudioManager>();

}
```

Creación e inicialización de variables

Vaya a la función “OnTriggerExit” y escriba la siguiente estructura condicional para cambiar la música cuando el “Player” atraviese el “BossCheckPoint”.

```
void OnTriggerExit(Collider other) {
    if(other.tag == "Player")
    {
        bossController.bossAwake = true;
        collider.isTrigger = false;
        characterMovement.enabled = false;
        playerAnimator.Play("Ghoulie_Idle");
        smoothFollow.bossCameraActive = true;

        if(newTrack != null){
            audioManager.ChangeMusic(newTrack);
        }
    }
```

Estructura condicional para cambiar la música

Vaya a Unity, seleccione el objeto “BossCheckPoint” y asigne el track de “BossMusic” al componente “BossCheckPoint”.

Boss Check Point (Script)
Script
BossCheckPoint
Collider
None (Box Collider)
New Track
BossMusic

Asignación de nuevo clip de música

RETO AL ESTUDIANTE

Hacer que se escuche la música de “Victory.mp3” cuando muera el jefe.

Añadir los audios de los comportamientos de daño y muerte del jefe contenidos en la carpeta “resources”.

Hacer que las piedras del ataque03 bajen más rápido conforme a la salud del jefe.

IMPLEMENTACIÓN EN ANDROID

INTRODUCCIÓN

En este capítulo nos enfocaremos a los detalles finales de nuestro juego con los siguientes conceptos:

Añadir un video tipo “CutScene” dentro de nuestro proyecto

Exportar nuestro juego para ordenadores Windows

Instalar y preparar los elementos necesarios para la exportación a Android

Crear y programar los controles virtuales para nuestro juego en dispositivos móviles

Implementación de “Ads” o anuncios para monetizar nuestro juego

Exportar nuestro juego como archivo APK

Insertar video en Unity

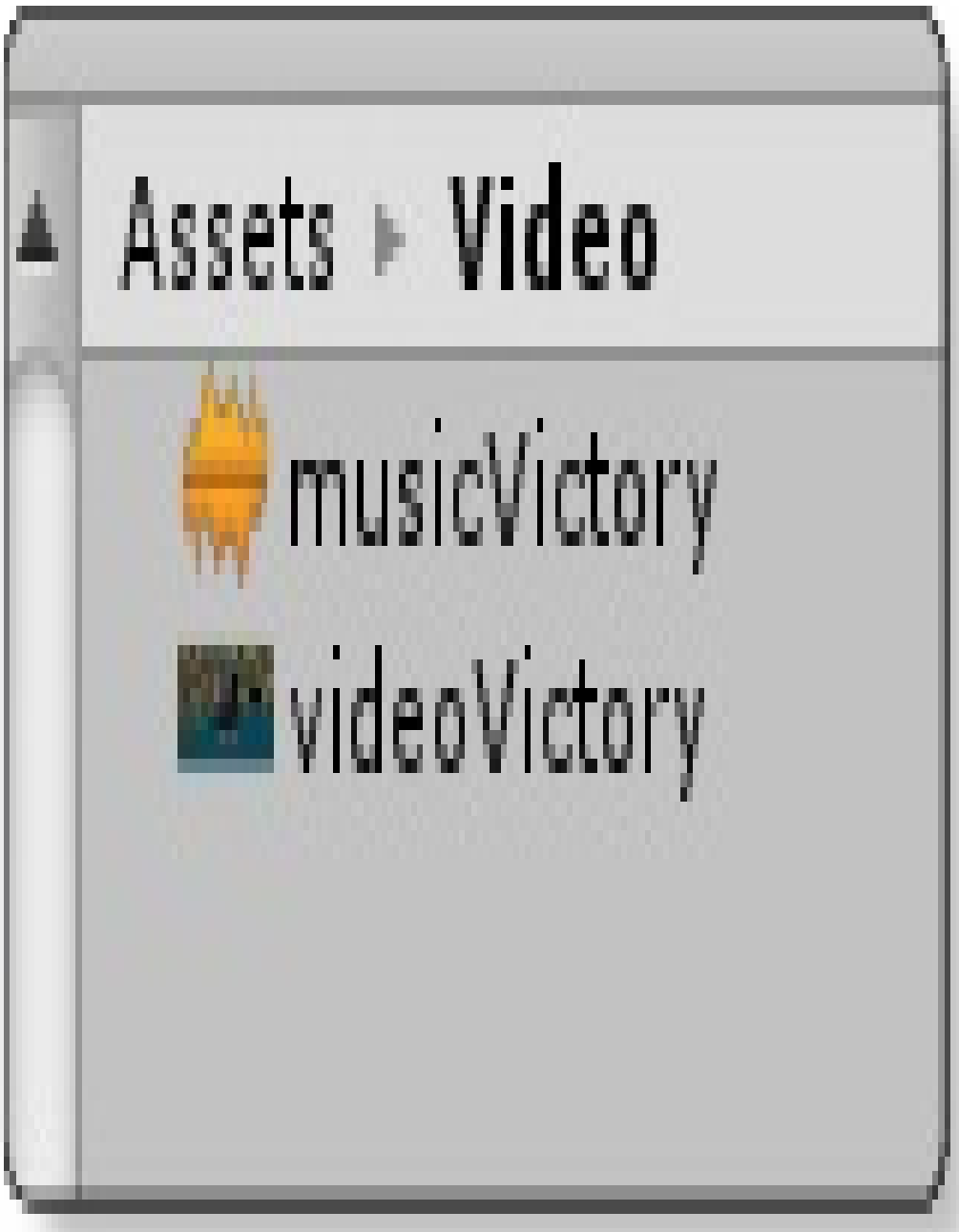
En este apartado insertaremos un video en la escena. Dentro de la carpeta de “resources” del capitulo encontraremos el video “videoVictory.mp4” y su audio aparte “musicVictory.wav”.

Insertaremos el video cuando haya acabado con el jefe y posteriormente nos regresaremos a la escena del menú inicial.

Esta técnica nos servirá para añadir un video en nuestro proyecto como “CutScene” al programarlo con algún evento específico.

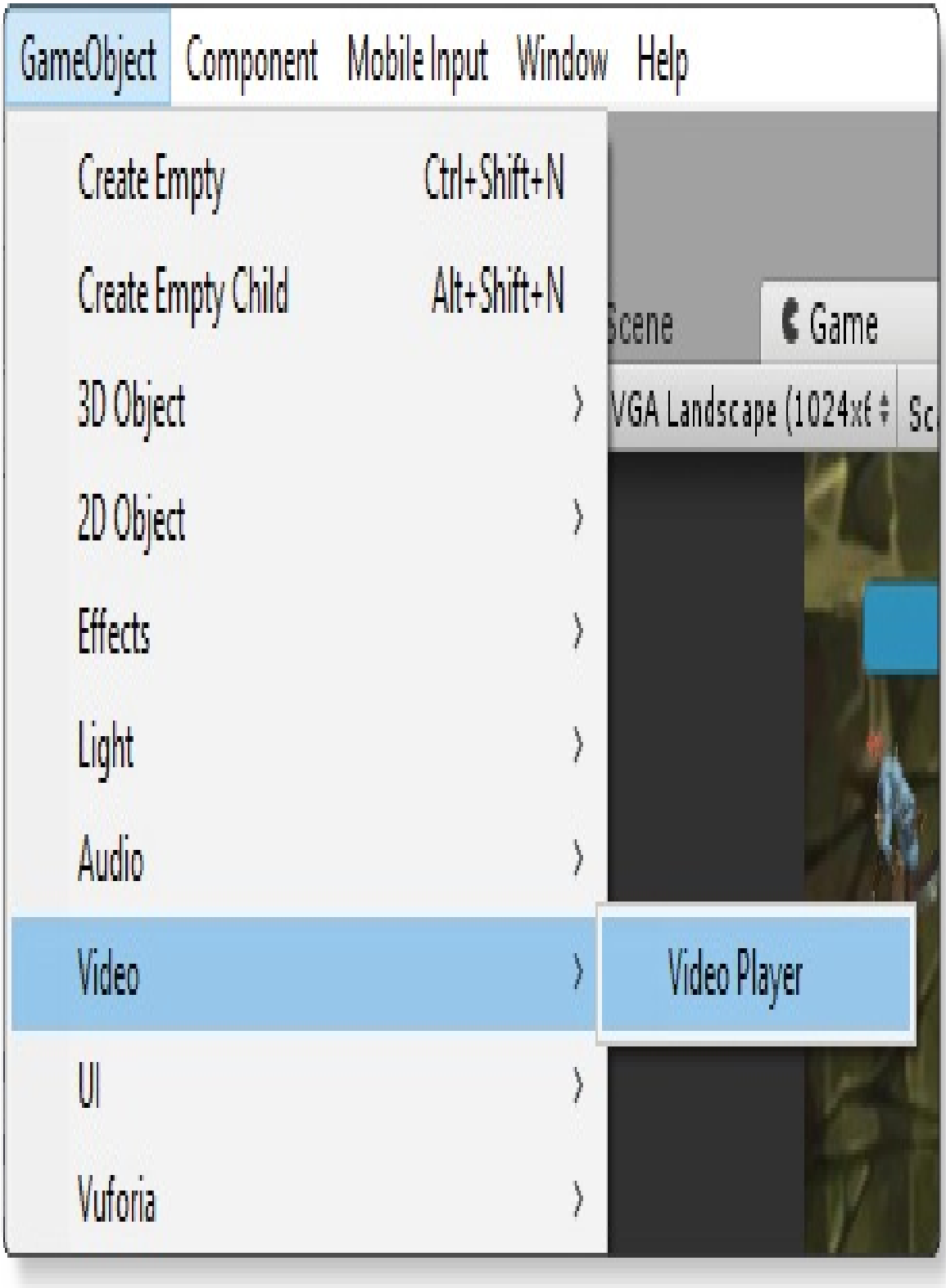
En la ventana “Project” haga un nuevo folder y nómbrelo “Video”.

Importe al nuevo folder los archivos “videoVictory” mp4 y wav.


Assets
Video
musicVictory
videoVictory

Importación de audio y video

En el menú general asigne “GameObject / Video / Videoplayer” para crear un nuevo objeto tipo “VideoPlayer”.


GameObject
Component
Mobile Input
Window
Help
Create Empty
Ctrl+Shift+N
Create Empty Child
Alt+Shift+N
3D Object
2D Object
Effects
Light
Audio
Video
UI
Vuforia
Video Player
Scene
Game
VGA Landscape (1024x

Creación de objeto “VideoPlayer”

Con el objeto “VideoPlayer” seleccionado, asigne un componente “AudioSource”.

Asigne los valores que se muestran para el componente “VideoPlayer”.

Video Player
Source
Video Clip
Video Clip
videoVictory
Play On Awake
Wait For First Frame
Loop
Playback Speed
1
Render Mode
Camera Near Plane
Camera
Main Camera (Camera)
Alpha
1
Aspect Ratio
Fit Horizontally
Audio Output Mode
Audio Source
Track 0 [2 ch]
Audio Source
Video Player (Audio Source)

Valores para componente “VideoPlayer”

NOTA:

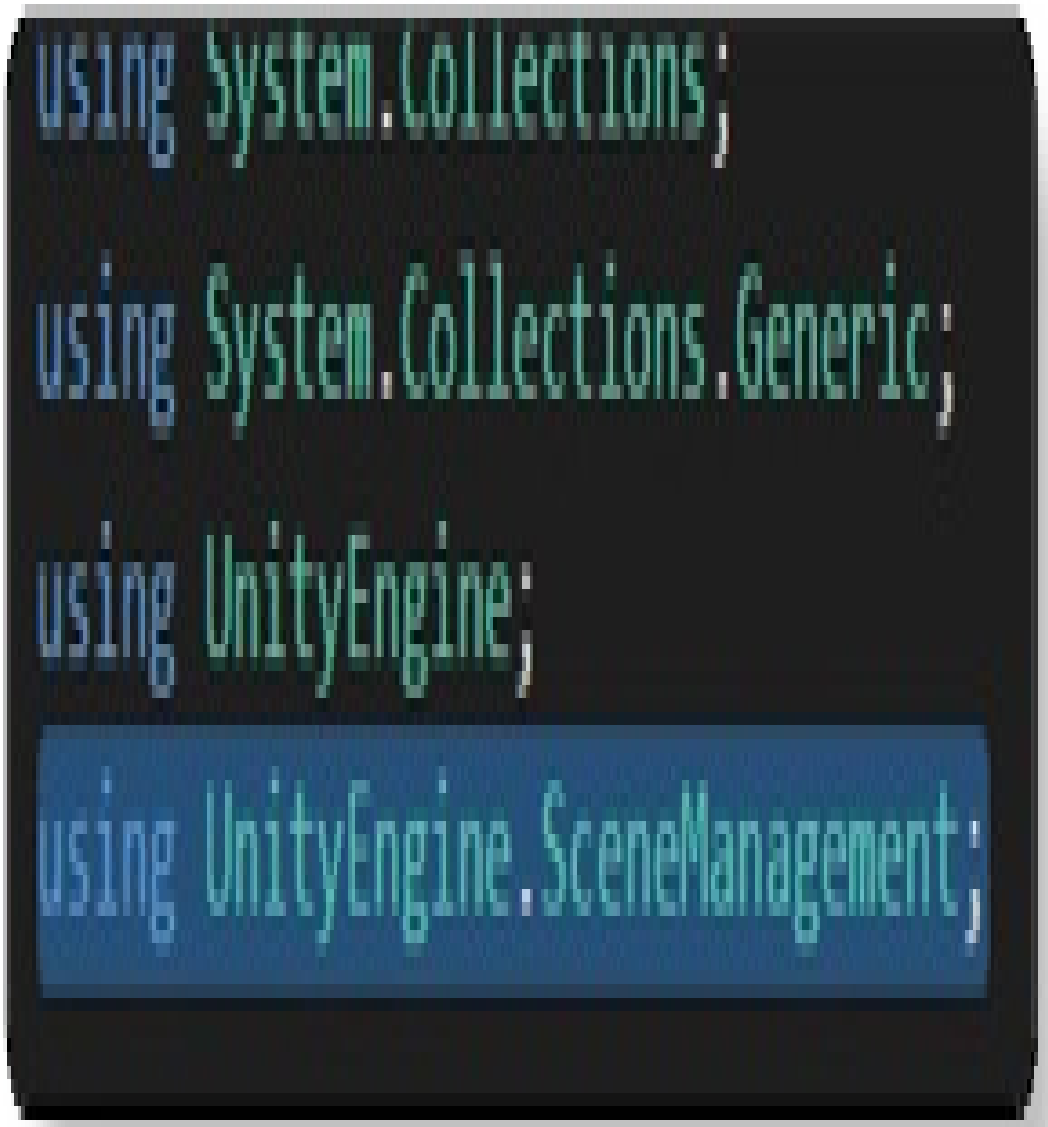
En el valor AudioSource, únicamente arrastramos el componente “AudioSource” que se encuentra bajo el componente “VideoPlayer” para asignarlo.

En este punto si previsualizamos el juego observaremos inmediatamente correr el video ya que no lo hemos programado que corra bajo ninguna condición.

En los valores de “VideoPlayer” desactive la casilla “WaitForFirstFrame”.

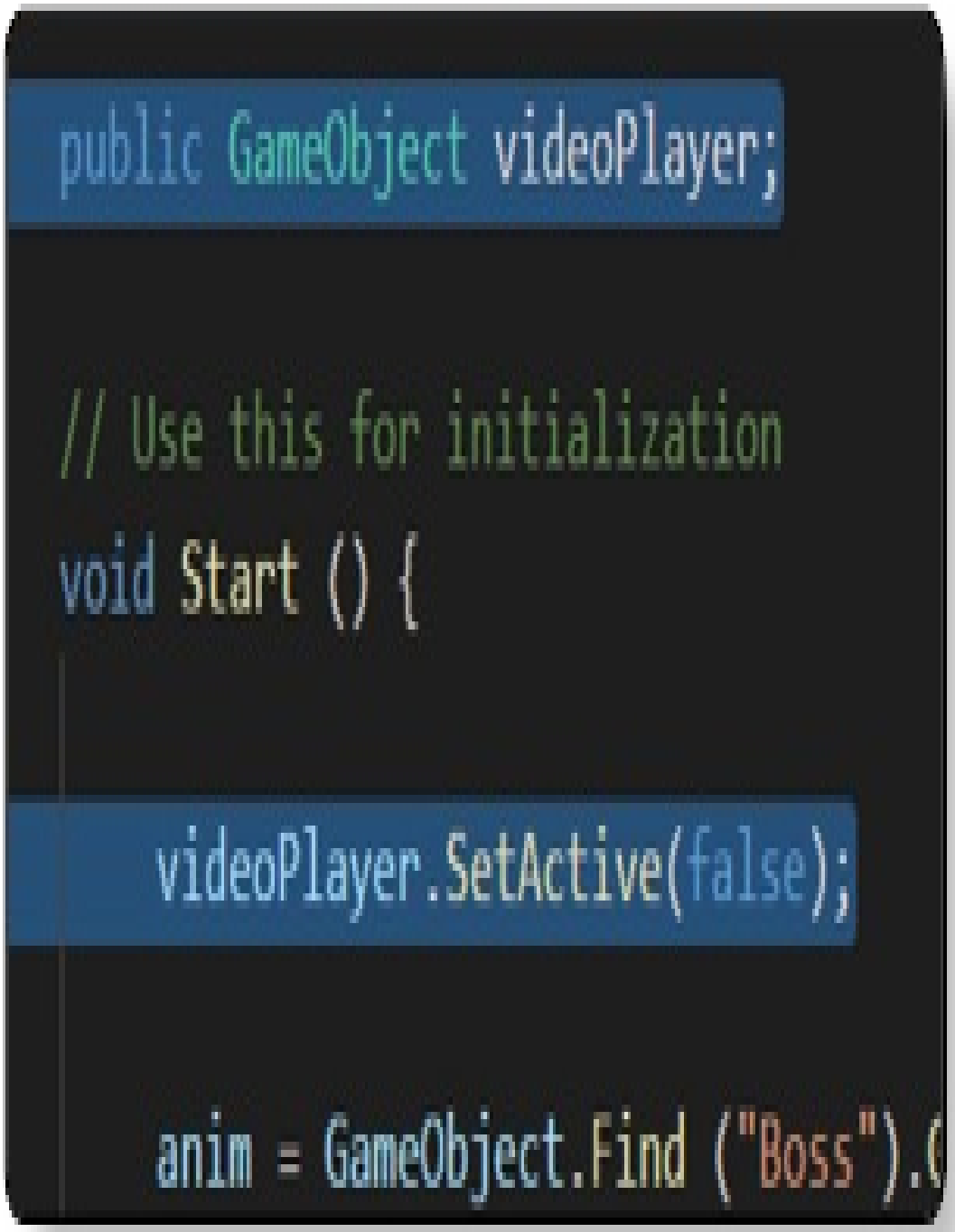
Necesitamos correr el video conforme a la muerte del jefe.

Vaya al script “BossHealth” y mande llamar la librería “SceneManagment” como se muestra.


using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

Importación de librería

Haga la variable que se indica e inicialícela como se muestra.


public GameObject videoPlayer;
// Use this for initialization
void Start () {
videoPlayer.SetActive(false);
anim = GameObject.Find ("Boss").

Creación e inicialización de variables

Haga la siguiente función “IEnumerator” y mándela llamar en la función “BossDead” como se muestra.

```
void BossDead(){
    bossDead = true;
    anim.SetTrigger ("isDead");
    BossController.bossAwake = false;
    capsuleCollider.enabled = false;
    weaponCollider.enabled = false;
    triggerCollider.enabled = false;

    StartCoroutine (playVideo ());

    if(newTrack02 != null){
        audioManager.ChangeMusic02(newTrack02);
    }
}

IEnumerator playVideo(){
    yield return new WaitForSeconds (5);
    videoPlayer.SetActive(true);
    yield return new WaitForSeconds (7);
    SceneManager.LoadScene (0);
}
```

Función IEnumerator “PlayVideo”

Previsualice el juego para observar que cuando muere el jefe tarda 5 segundos en iniciar el video, seguidamente se tarda 7 con el video y al terminar se carga la escena del menú de inicio.

Exportación del juego para pc

En este punto de nuestro proyecto ya podemos exportar nuestro juego para ser jugado en la PC con los controles del teclado y ratón que hemos programado o bien para poder conectar un mando USB al ordenador.

Puede también conectar un mando de Xbox 360 y funcionará a la perfección.

A continuación, exportaremos nuestro juego para plataforma PC.

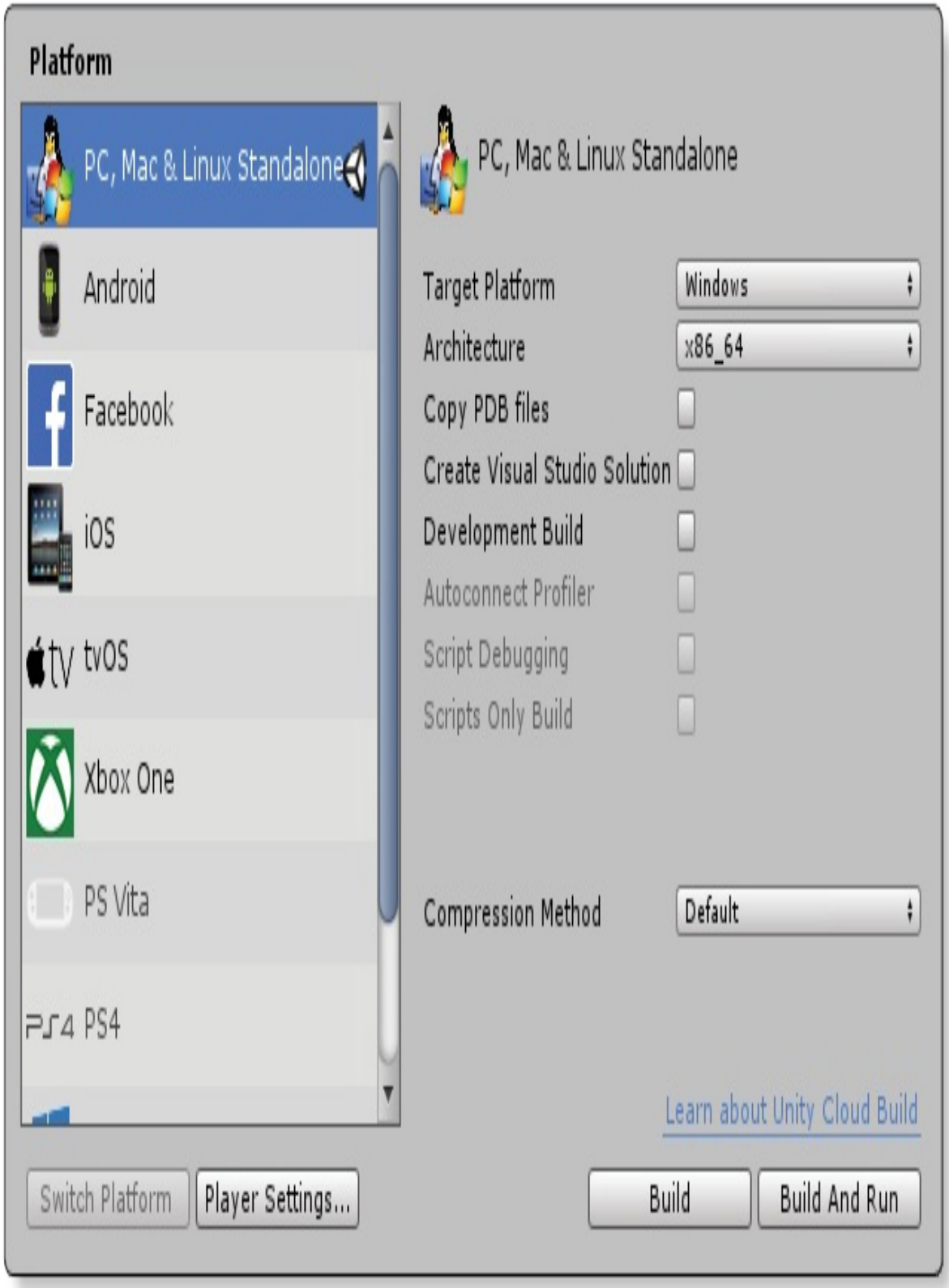
En el menú general vaya a “File / BuildSettings”.

Asegúrese de tener cargadas las escenas activas de nuestro juego.

Build Settings
Scenes In Build
Scenes/StartScreen
0
Scenes/GhoulieMan604
1

Escenas para construir nuestro juego

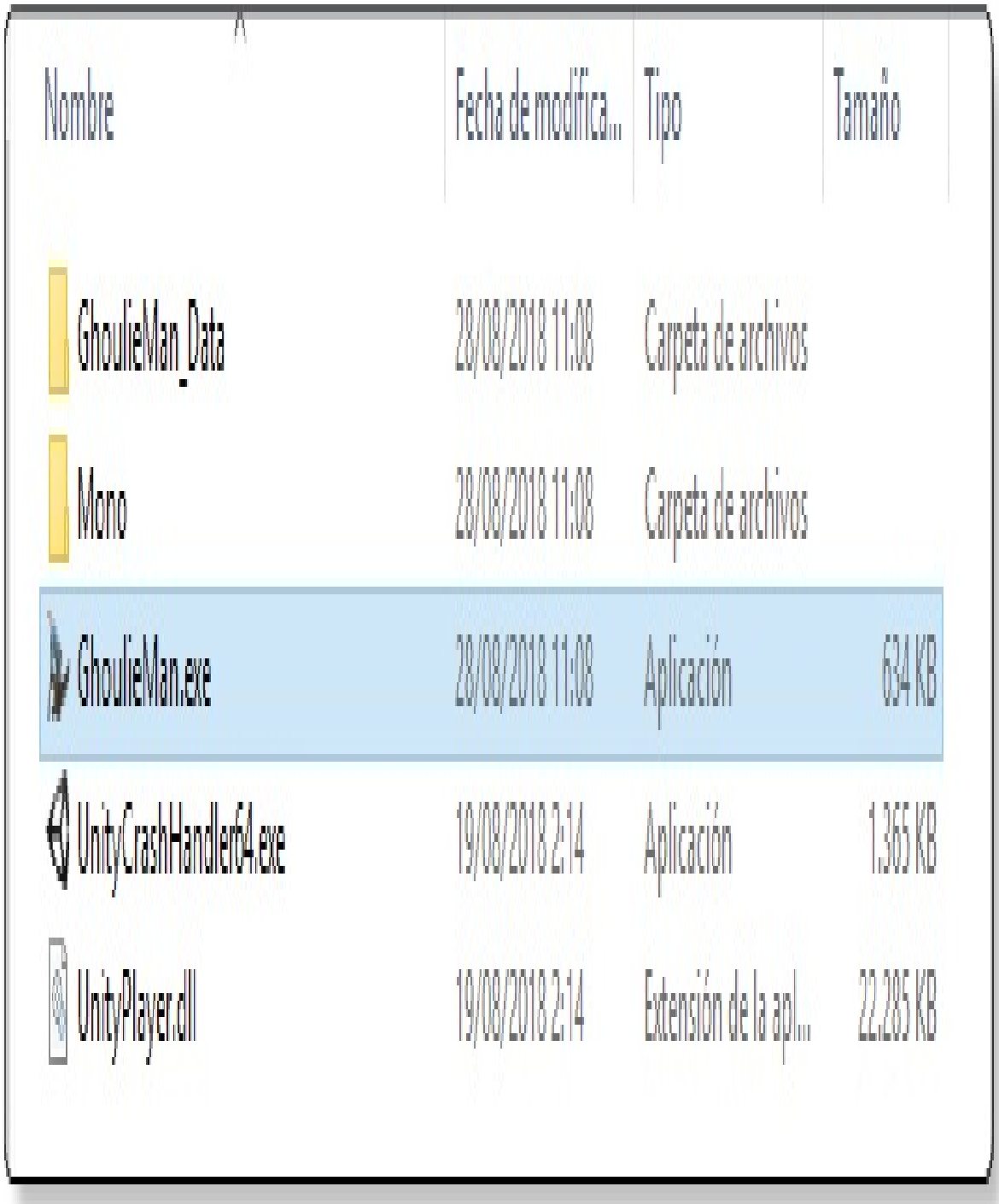
Seleccione la plataform “PC, Mac & Linux Standalone, asigne los valores que se muestran y clic en “Build”.


Platform
PC, Mac & Linux Standalone
Android
Facebook
iOS
tvOS
Xbox One
PS Vita
PS4
PC, Mac & Linux Standalone
Target Platform
Windows
Architecture
x86_64
Copy PDB files
Create Visual Studio Solution
Development Build
Autoconnect Profiler
Script Debugging
Scripts Only Build
Compression Method
Default
Learn about Unity Cloud Build
Switch Platform
Player Settings...
Build
Build And Run

Valores para plataforma PC

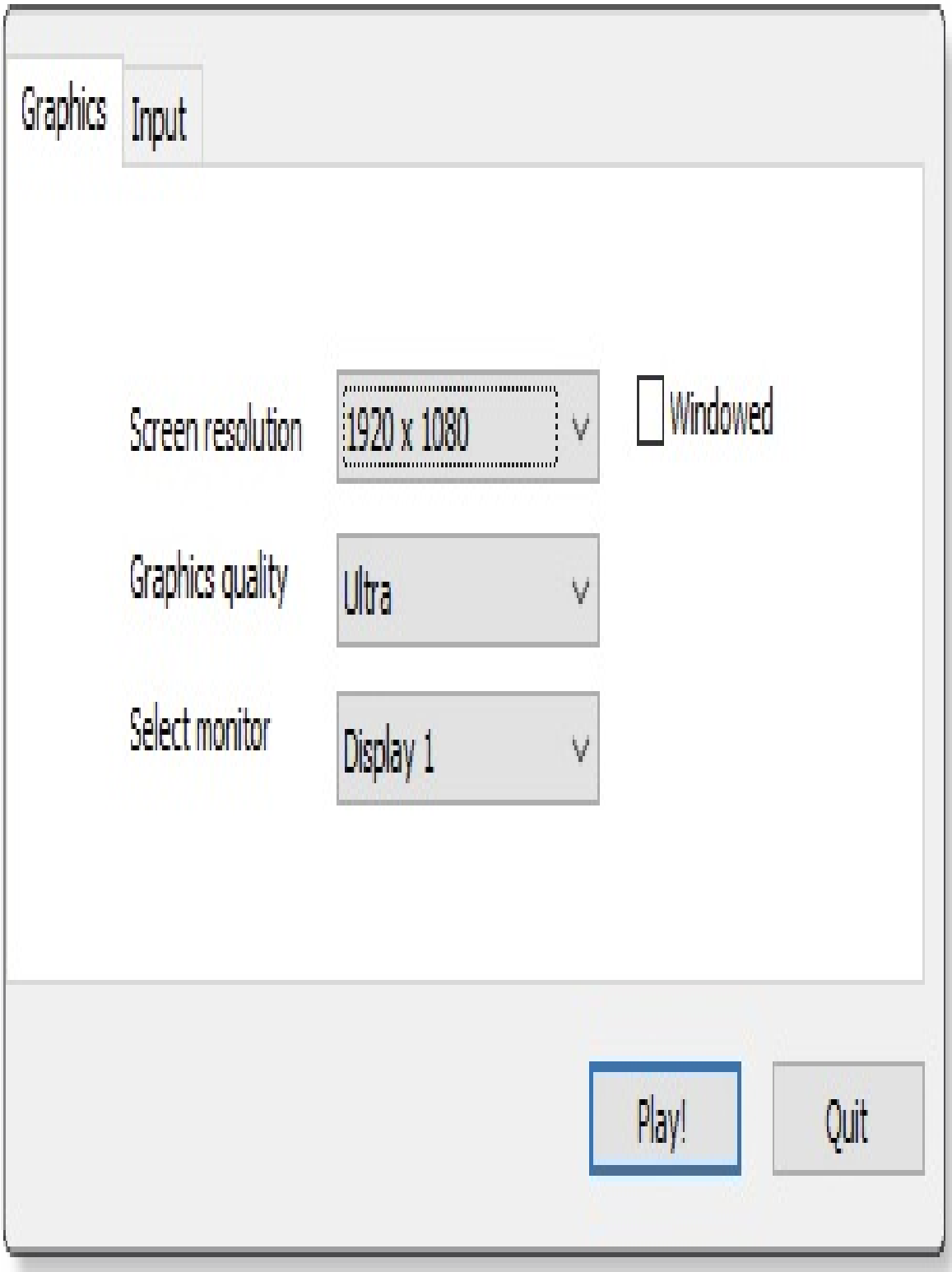
Elija la ruta donde se va a crear el archivo instalador del juego.

Al finalizar de construir podrá observar los siguientes archivos en la ruta asignada previamente.


Nombre
Fecha de modifica...
Tipo
Tamaño
GhoulieMan_Data
28/08/2018 11:08
Carpeta de archivos
Mono
28/08/2018 11:08
Carpeta de archivos
GhoulieMan.exe
28/08/2018 11:08
Aplicación
634 KB
UnityCrashHandler64.exe
19/08/2018 2:14
Aplicación
1.365 KB
UnityPlayer.dll
19/08/2018 2:14
Extensión de la apl...
22.285 KB

Archivos externos construidos

Doble clic en el archivo .exe para instalarlo en su ordenador y observar las opciones


Graphics
Input
Screen resolution
1920 x 1080
Windowed
Graphics quality
Ultra
Select monitor
Display 1
Play!
Quit

Opciones de visualización del juego

¡Elija la configuración deseado y clic en “Play!”

Preparación de elementos para plataforma Android

En este apartado descargaremos e instalaremos los elementos necesarios para poder previsualizar nuestro videojuego en nuestro dispositivo móvil Android a tiempo real y para poder exportarlo en esta misma plataforma.

En el menú general asigne “Edit / Prefences” y clic en la pestaña “External Tools”.

Primero vamos a descargar e instalar el módulo de Java “JDK”.

Vaya al apartado JDK y clic en el botón “Download”.

Este link lo llevara a la página oficial para descargar “Java SE Development Kit”, elija la versión conforme a su sistema operativo.

Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u181 checksum

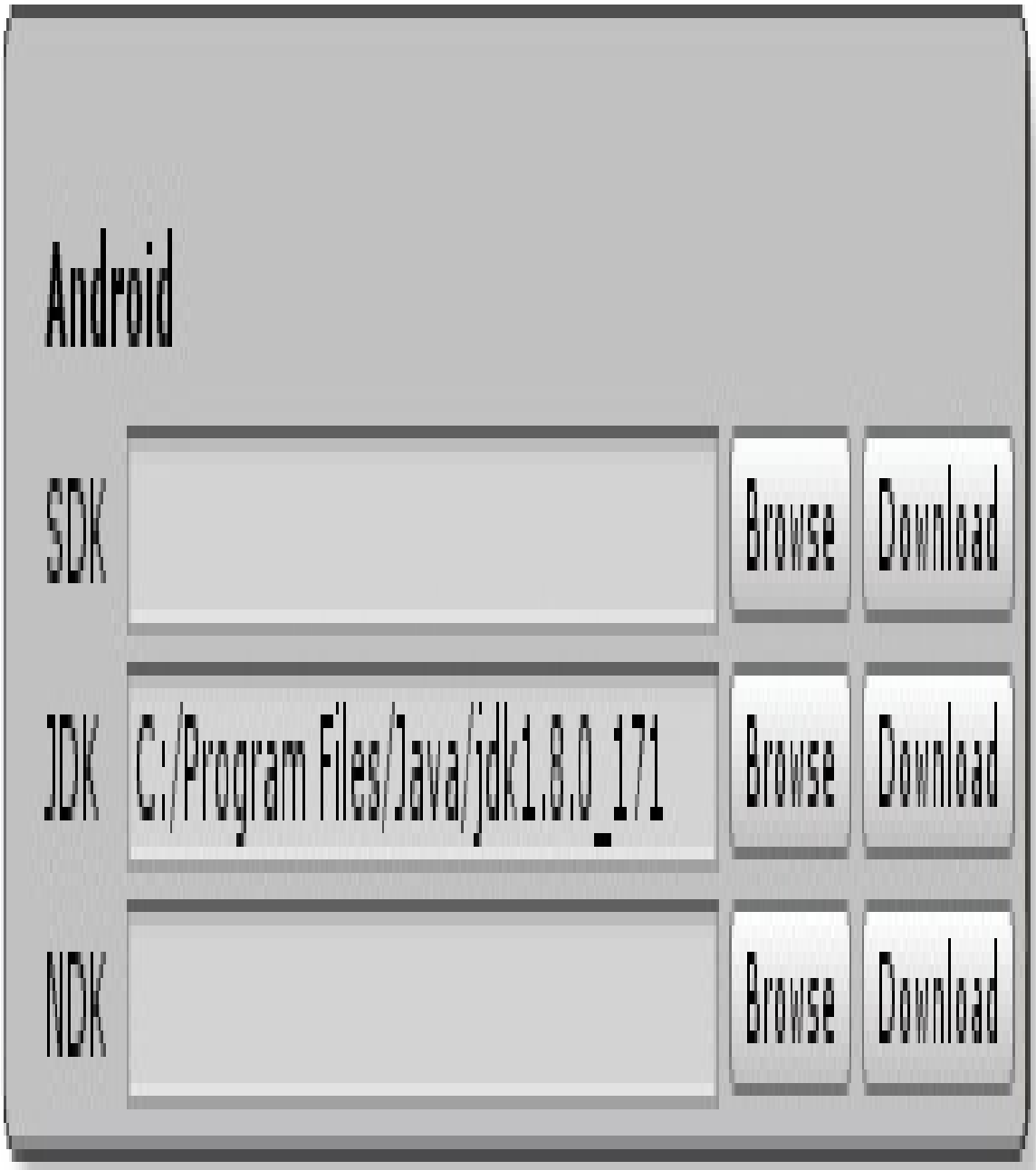
Java SE Development Kit 8u181

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.95 MB	jdk-8u181-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.89 MB	jdk-8u181-linux-arm64-vfp-hflt.tar.gz
Linux x86	165.06 MB	jdk-8u181-linux-i586.rpm
Linux x86	179.87 MB	jdk-8u181-linux-i586.tar.gz
Linux x64	162.15 MB	jdk-8u181-linux-x64.rpm
Linux x64	177.05 MB	jdk-8u181-linux-x64.tar.gz
Mac OS X x64	242.83 MB	jdk-8u181-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	133.17 MB	jdk-8u181-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.34 MB	jdk-8u181-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	133.83 MB	jdk-8u181-solaris-x64.tar.Z
Solaris x64	92.11 MB	jdk-8u181-solaris-x64.tar.gz
Windows x86	194.41 MB	jdk-8u181-windows-i586.exe
Windows x64	202.73 MB	jdk-8u181-windows-x64.exe

Descarga de JDK

Tome en cuenta en donde se instala para que pueda asignar la ruta correspondiente en Unity.

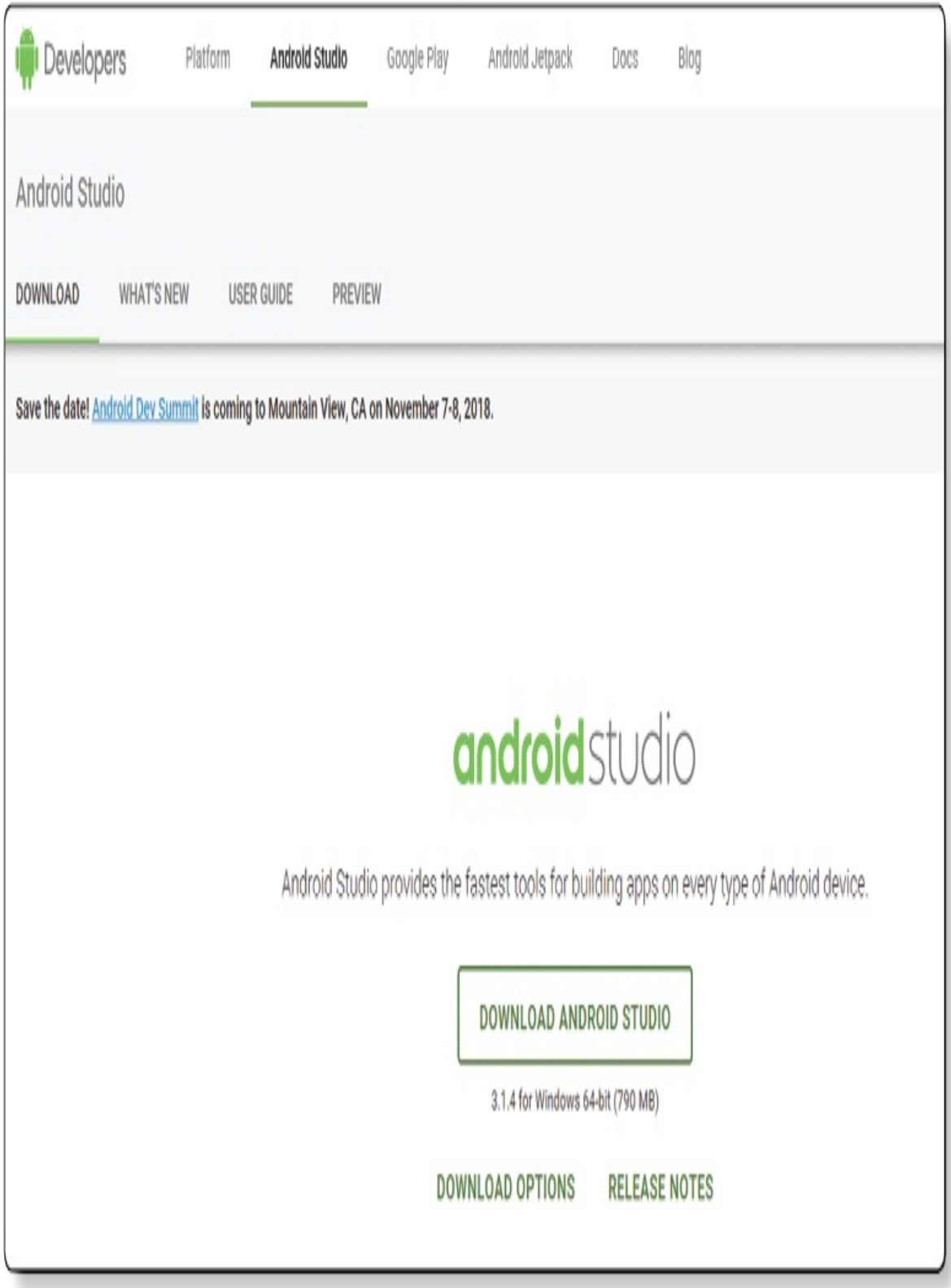

Android
SDK
Browse
Download
JDK
C:/Program Files/Java/jdk1.8.0_171
Browse
Download
NDK
Browse
Download

Asignación en Unity de JDK

A continuación, se descargará y se instalará el módulo SKD para Android.

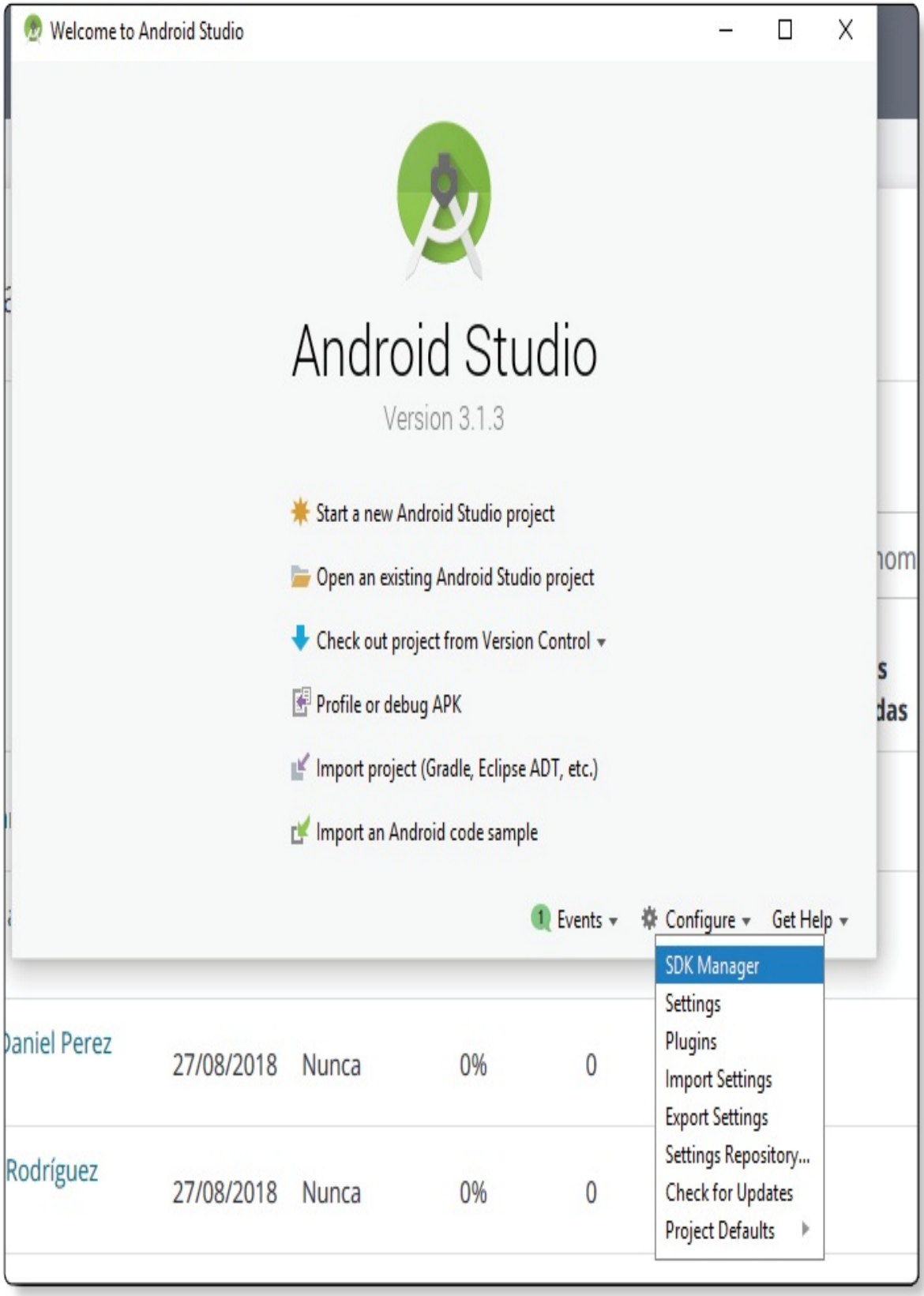
Clic en el botón “Download” relativo a SDK.

En el enlace clic en “Download Android Studio”.


Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
Blog
Android Studio
DOWNLOAD
WHAT'S NEW
USER GUIDE
PREVIEW
Save the date! Android Dev Summit is coming to Mountain View, CA on November 7-8, 2018.
android studio
Android Studio provides the fastest tools for building apps on every type of Android device.
DOWNLOAD ANDROID STUDIO
3.1.4 for Windows 64-bit (790 MB)
DOWNLOAD OPTIONS
RELEASE NOTES

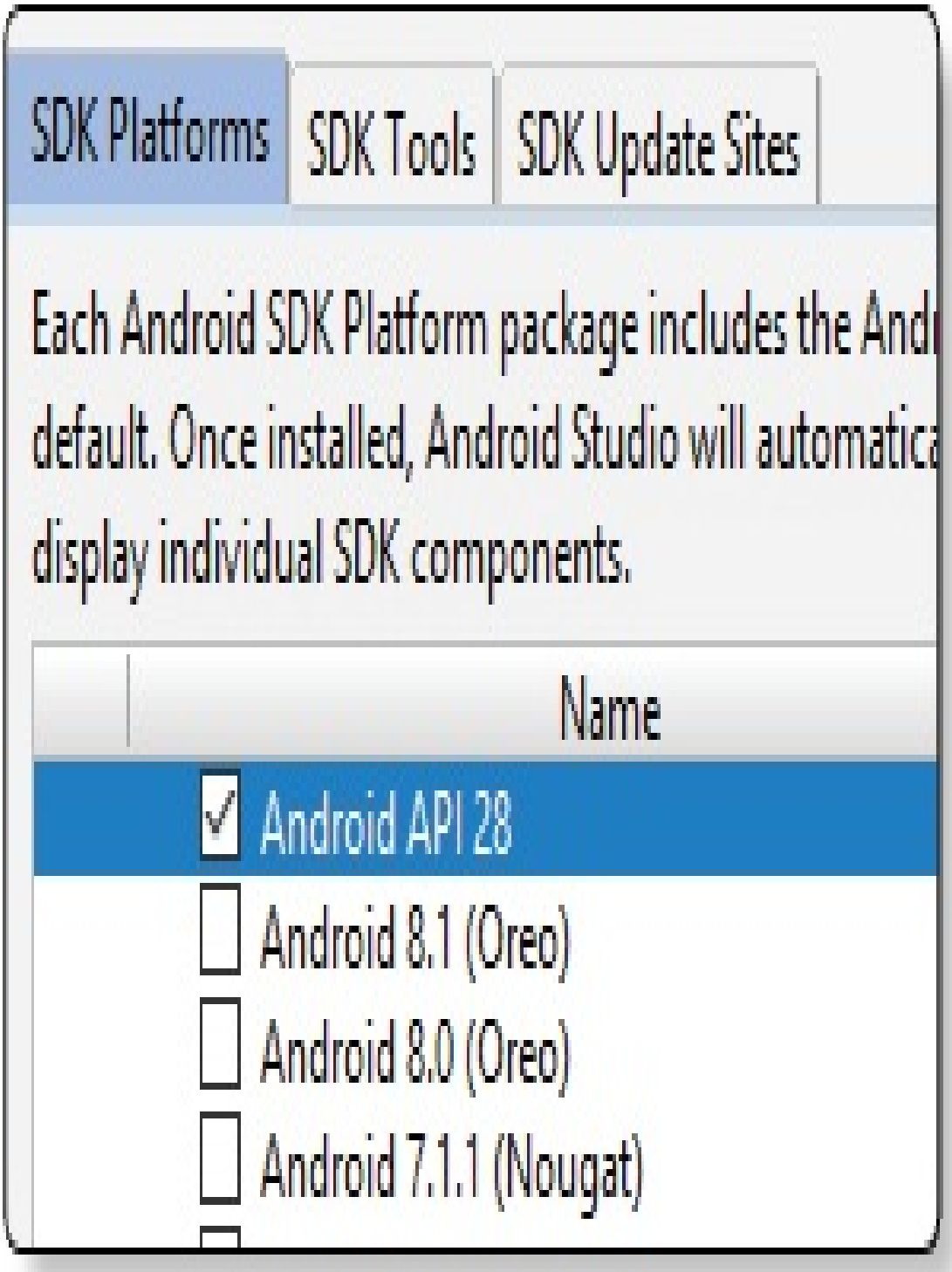
Descarga de Android Studio

Después de instalarlo abra Android Studio y vaya a SKD Manager como se muestra.


Welcome to Android Studio
Android Studio
Version 3.1.3
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Events
Configure
Get Help
SDK Manager
Settings
Plugins
Import Settings
Export Settings
Settings Repository...
Check for Updates
Project Defaults
Daniel Perez
27/08/2018
Nunca
0%
0
Rodríguez
27/08/2018
Nunca
0%
0

SDK Manager

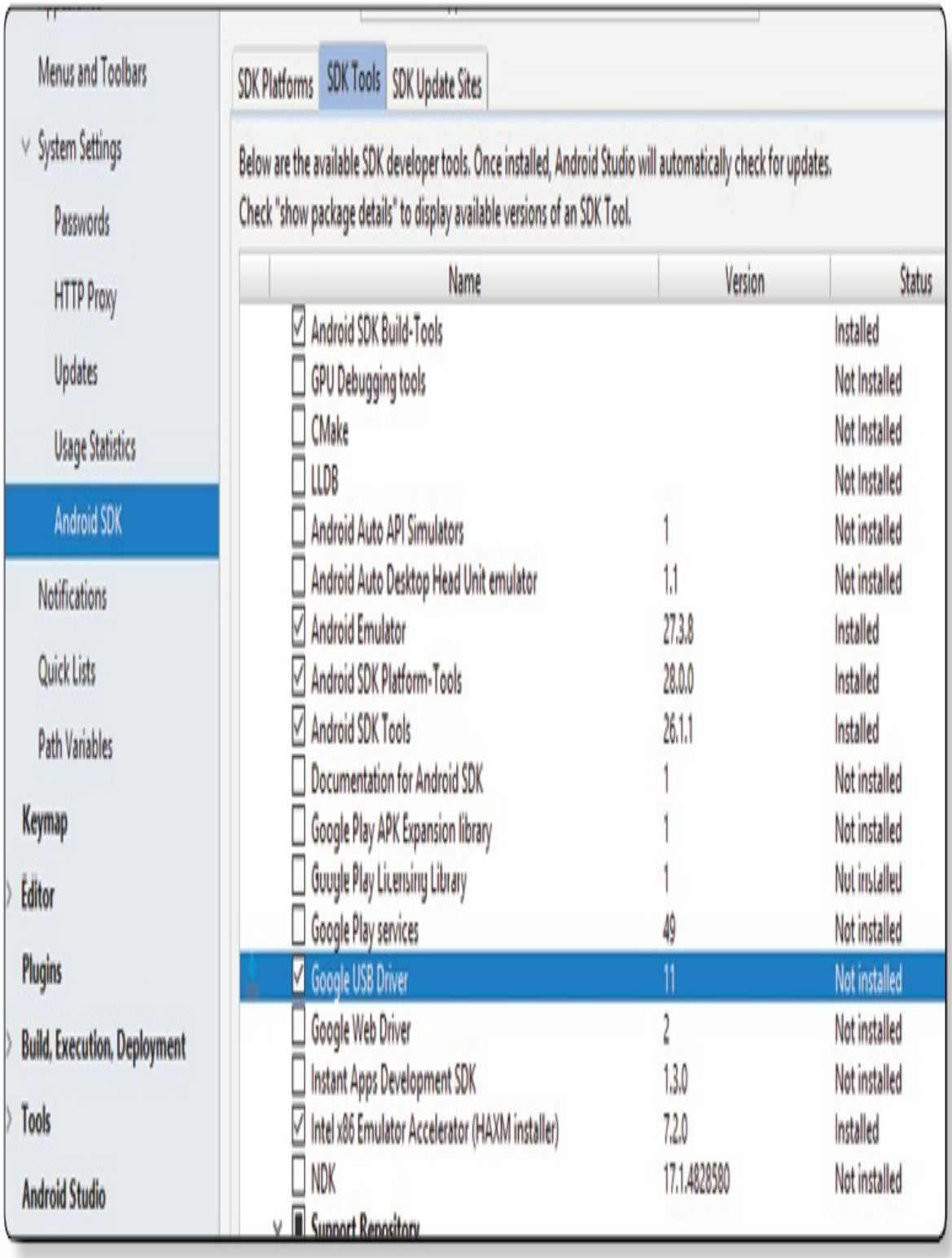
Seleccione los siguientes paquetes y descárguelos.


SDK Platforms
SDK Tools
SDK Update Sites
Each Android SDK Platform package includes the And
default. Once installed, Android Studio will automatica
display individual SDK components.
Name
Android API 28
Android 8.1 (Oreo)
Android 8.0 (Oreo)
Android 7.1.1 (Nougat)

Descarga de paquete

NOTA:

Debe de tener en cuenta que, para la fecha de impresión de este libro, esta era la versión más reciente, debe de descargar la versión más reciente conforme el tiempo de realización del ejercicio.


Menus and Toolbars
System Settings
Passwords
HTTP Proxy
Updates
Usage Statistics
Android SDK
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Build, Execution, Deployment
Tools
Android Studio
SDK Platforms
SDK Tools
SDK Update Sites
Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates.
Check "show package details" to display available versions of an SDK Tool.
Name
Version
Status
Android SDK Build-Tools
Installed
GPU Debugging tools
Not Installed
CMake
Not Installed
LLDB
Not Installed
Android Auto API Simulators
1
Not installed
Android Auto Desktop Head Unit emulator
1.1
Not installed
Android Emulator
27.3.8
Installed
Android SDK Platform-Tools
28.0.0
Installed
Android SDK Tools
26.1.1
Installed
Documentation for Android SDK
1
Not installed
Google Play APK Expansion library
1
Not installed
Google Play Licensing Library
1
Not installed
Google Play services
49
Not installed
Google USB Driver
11
Not installed
Google Web Driver
2
Not installed
Instant Apps Development SDK
1.3.0
Not installed
Intel x86 Emulator Accelerator (HAXM installer)
7.2.0
Installed
NDK
17.1.4828580
Not installed
Support Repository

Descarga de Google USB Driver

Esta herramienta permitirá la conectividad entre el módulo Android y la conexión de nuestro dispositivo vía USB.

Al instalar “GoobleUSBDriver” tome nota de su ruta de instalación, la requerirá en pasos posteriores.

Installing Requested Components

SDK Path: C:\Users\Luis\AppData\Local\Android\Sdk

To install:

- Google USB Driver (extras;google;usb_driver)

Preparing "Install Google USB Driver (revision: 11)".

Downloading https://dl.google.com/android/repository/usb_driver_r11-windows.zip

"Install Google USB Driver (revision: 11)" ready.

Installing Google USB Driver in C:\Users\Luis\AppData\Local\Android\Sdk\extras\google\usb_driver

"Install Google USB Driver (revision: 11)" complete.

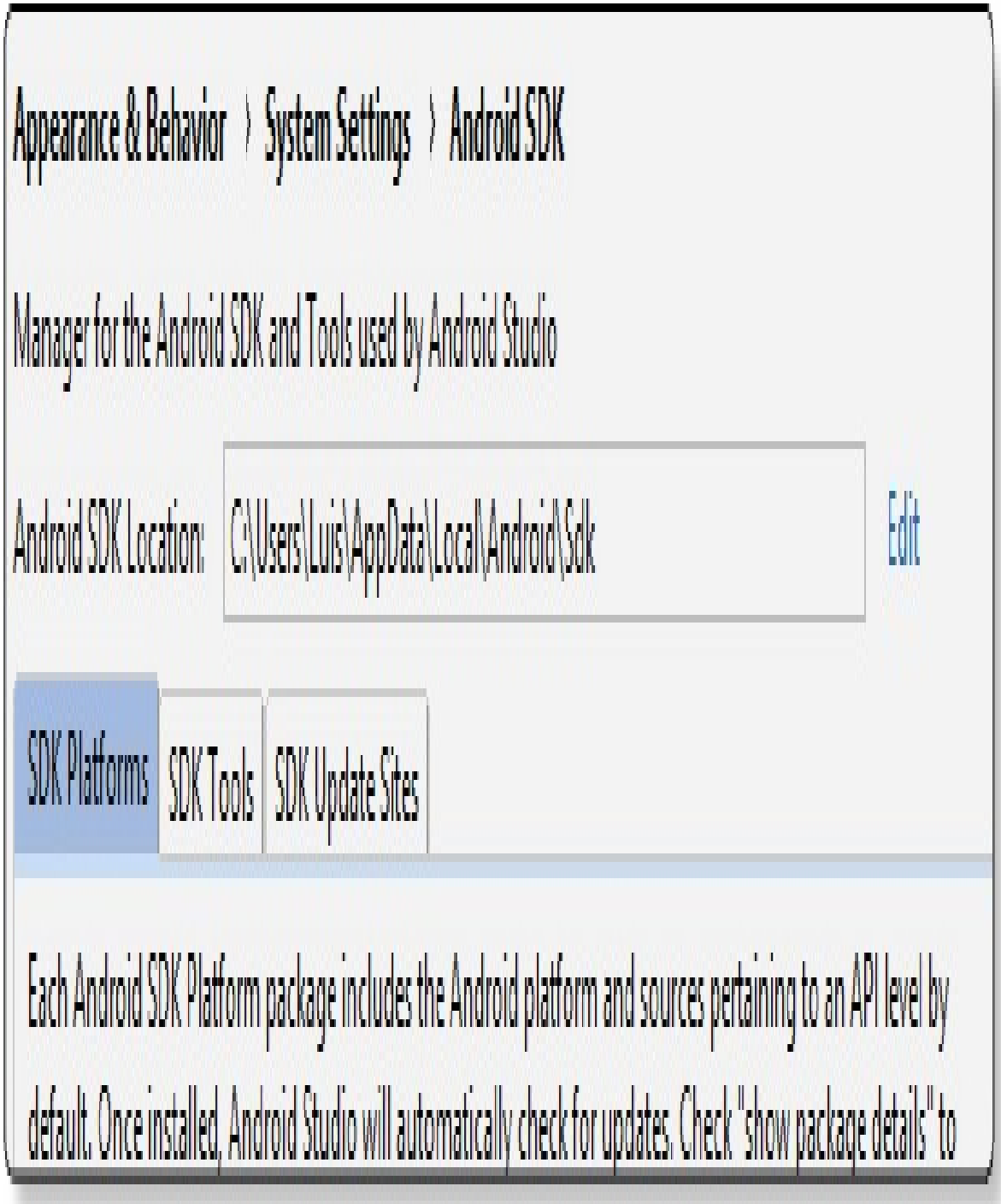
"Install Google USB Driver (revision: 11)" finished.

Ruta de instalación Google USB Driver

NOTA:

La ruta es “SDK Path”.

Tome nota de la ruta donde se instala el módulo SKD.


Appearance & Behavior › System Settings › Android SDK
Manager for the Android SDK and Tools used by Android Studio
Android SDK Location:
C:\Users\Luis\AppData\Local\Android\Sdk
Edit
SDK Platforms
SDK Tools
SDK Update Sites
Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by
default. Once installed, Android Studio will automatically check for updates. Check "show package details" to

Ruta de instalación de SDK

En Unity asigne la ruta para el módulo SDK.

Android
SDK
C:/Users/Luis/AppData/Local/Android/S
Browse
Download
JDK
C:/Program Files/Java/jdk1.8.0_171
Browse
Download
NDK
Browse
Download
IL2CPP requires that you have Android NDK r13b installed.
If you are not targeting IL2CPP you can leave this field empty.

Asignación en Unity de ruta SDK

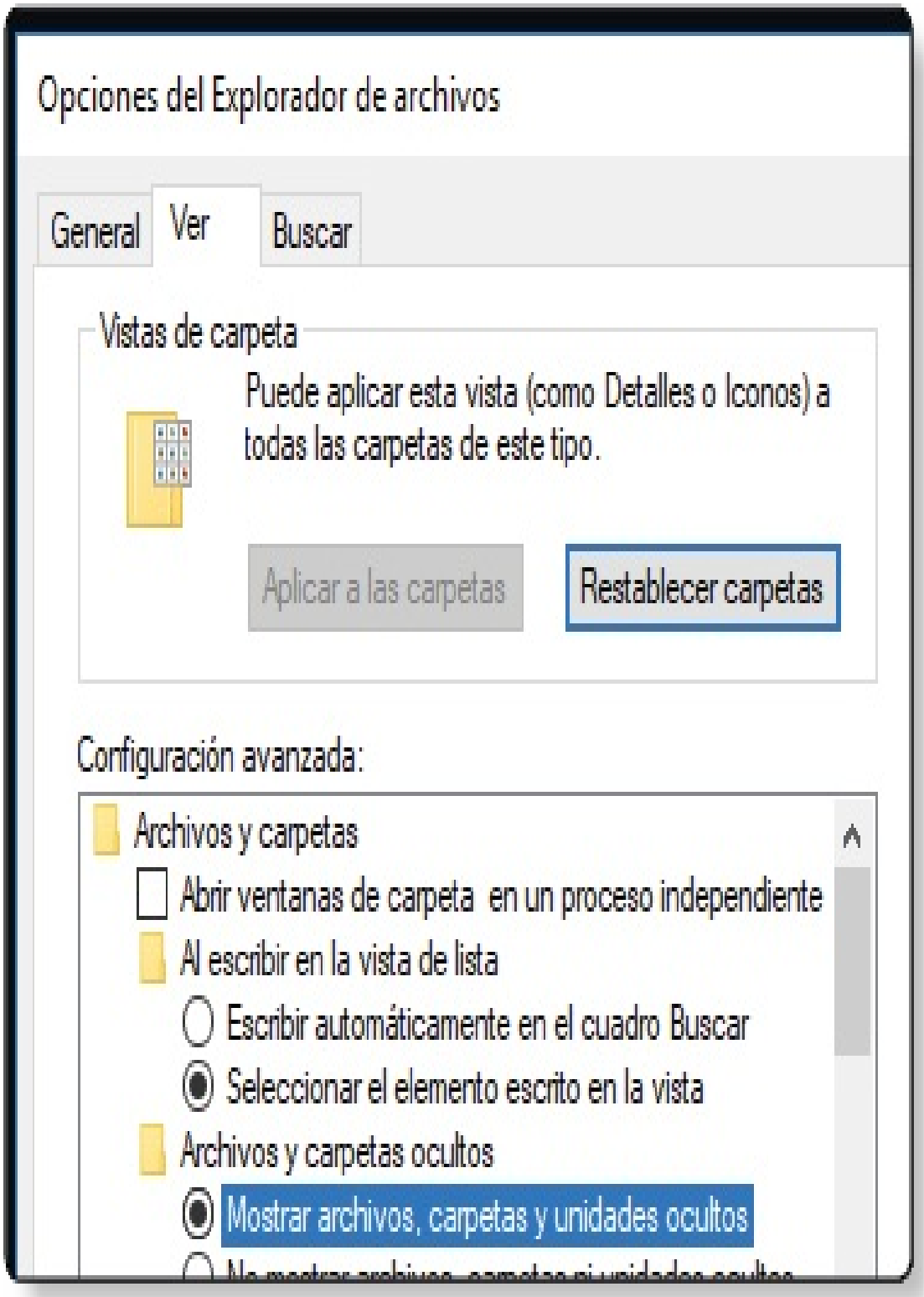
NOTA:

Muy probablemente no pueda visualizar la carpeta “APPData” ya que en Windows se encuentra escondida, para poderla visualizar debe de ir a las opciones de folder.

Filtros
Mejor coincidencia
Opciones del Explorador de archivos
Panel de control
Configuración

Opciones de folder en Windows

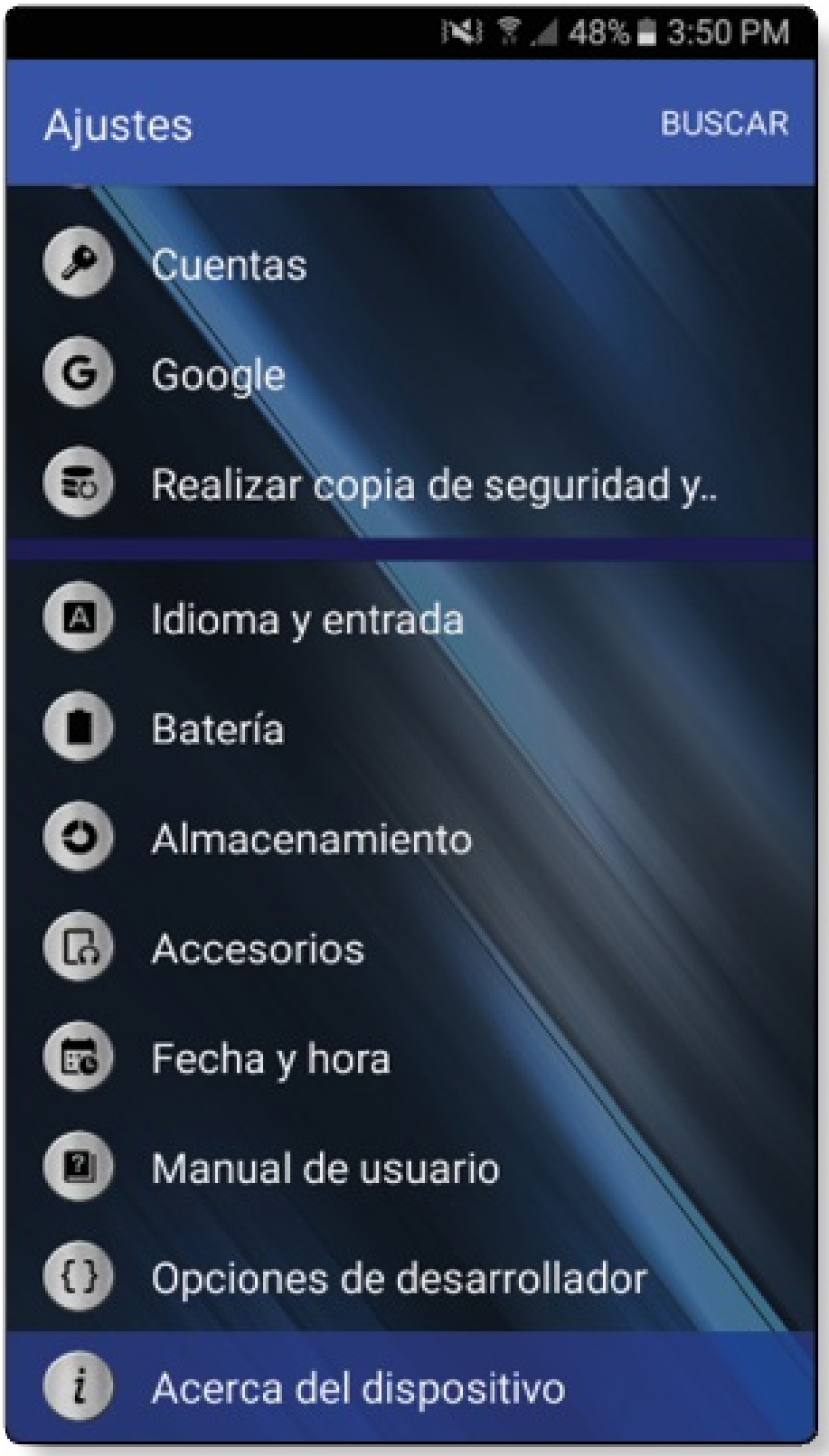
Configurar que muestre las carpetas ocultas como se muestra.


Opciones del Explorador de archivos
General
Ver
Buscar
Vistas de carpeta
Puede aplicar esta vista (como Detalles o Iconos) a todas las carpetas de este tipo.
Aplicar a las carpetas
Restablecer carpetas
Configuración avanzada:
Archivos y carpetas
Abrir ventanas de carpeta en un proceso independiente
Al escribir en la vista de lista
Escribir automáticamente en el cuadro Buscar
Seleccionar el elemento escrito en la vista
Archivos y carpetas ocultos
Mostrar archivos, carpetas y unidades ocultos

Activación para mostrar archivos y carpetas ocultas en Windows

A continuación, debemos de abrir las opciones de desarrollador en el dispositivo Android que por defecto se encuentran escondidas.

En su dispositivo móvil vaya a Ajustes y seleccione la opción “Acerca del dispositivo”.


48% 3:50 PM
Ajustes
BUSCAR
Cuentas
Google
Realizar copia de seguridad y..
Idioma y entrada
Batería
Almacenamiento
Accesorios
Fecha y hora
Manual de usuario
Opciones de desarrollador
Acerca del dispositivo

Ajustes en dispositivo móvil

NOTA:

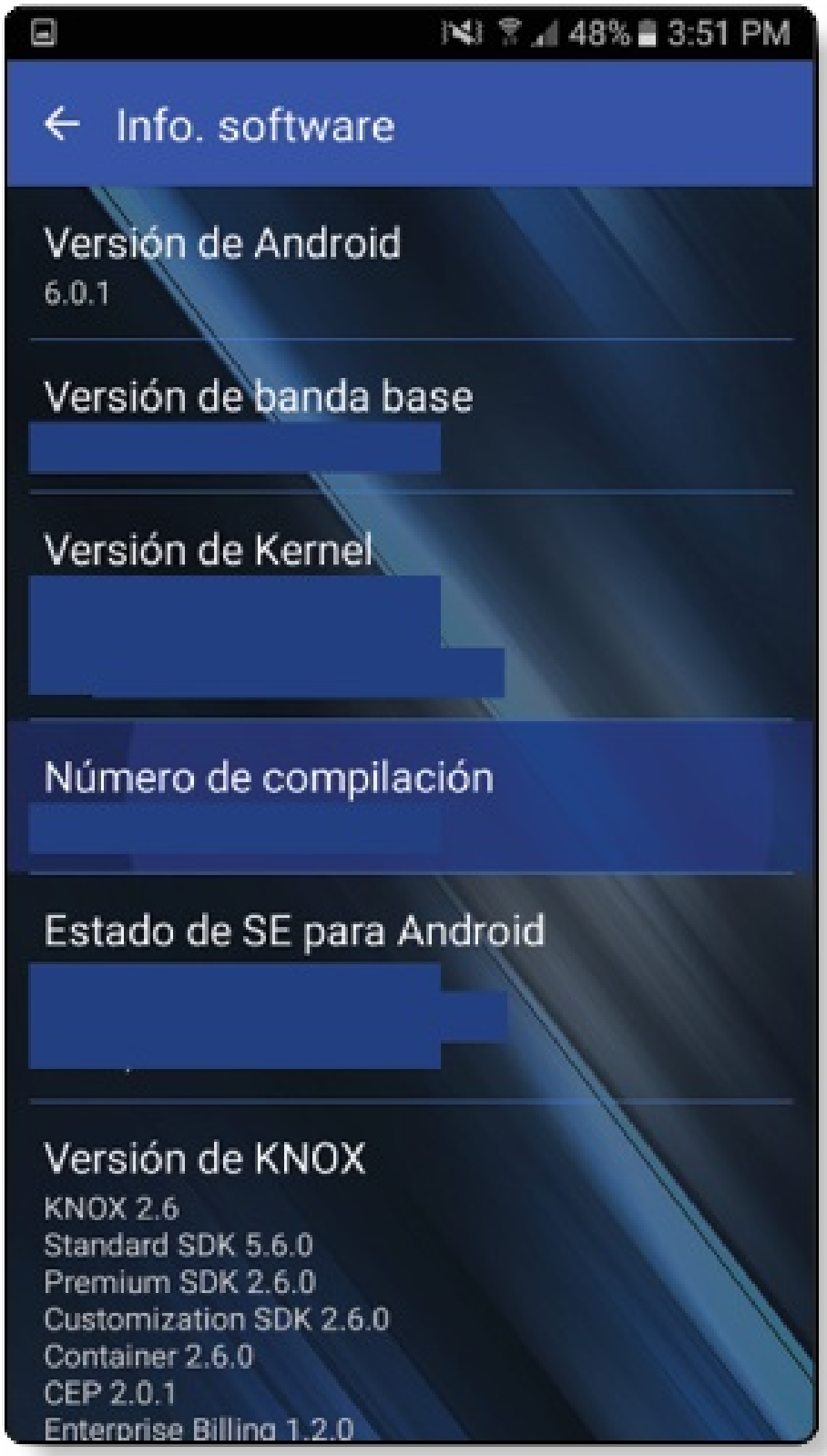
Las opciones pueden variar dependiendo la versión de su sistema operativo, puede también buscar en la web “Activación de opciones de desarrollador” conforme a su sistema.

Clic en la opción “Info. Software”.

48% 3:50 PM
Acerca del dispositivo
descarga.
Actualización programada
Desactivado
Estado
Ver el estado de la tarjeta SIM, el IMEI del dispositivo y demás información.
Información legal
Nombre del dispositivo
Número de modelo
Info. software
Ver la versión actual instalada de Android, la versión de banda base, la versión de kernel, el número de compilación, etc.
Información de la batería
Ver el estado de la batería del dispositivo, la carga restante y demás información.

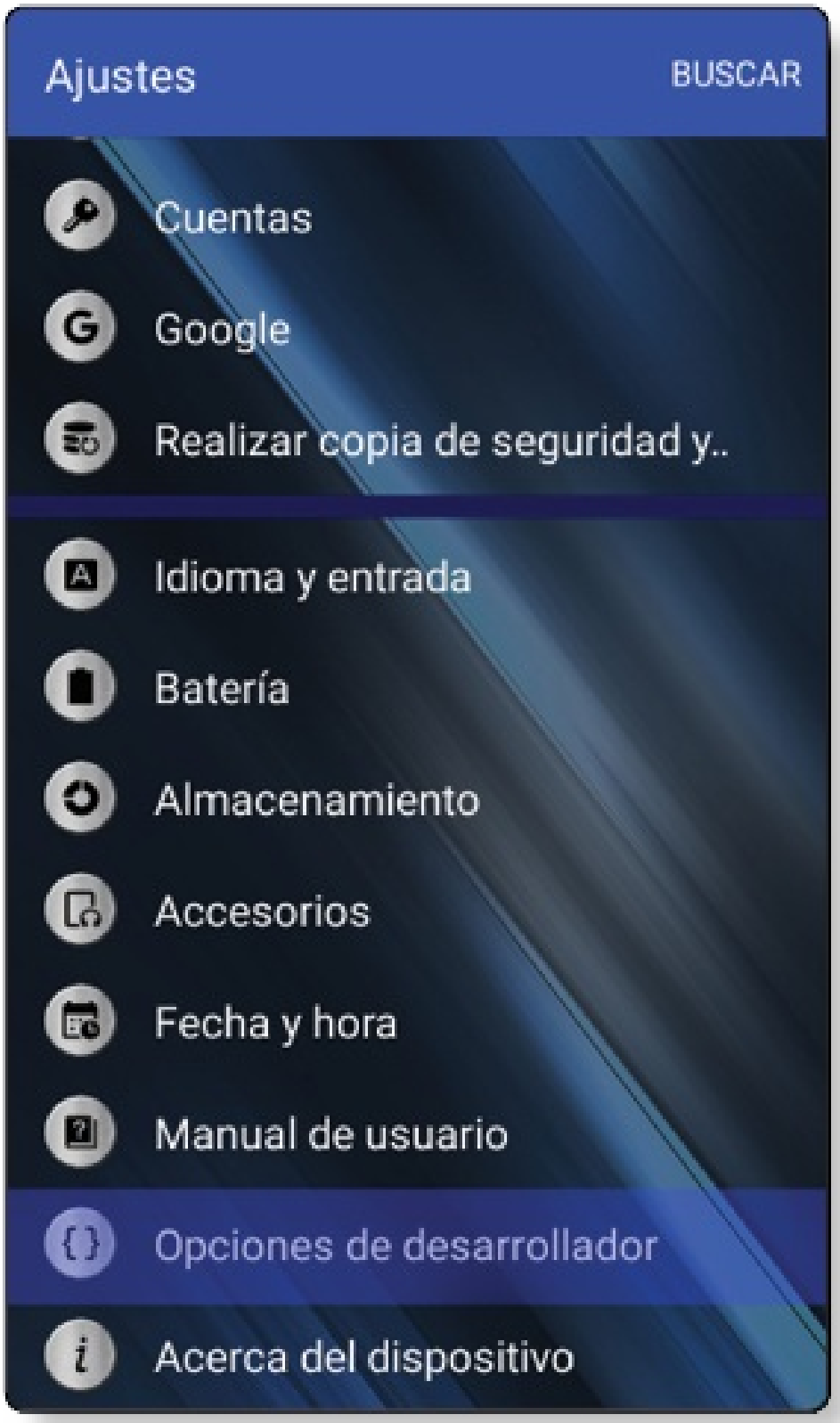
Acerca del dispositivo en dispositivo móvil

Clic 7 veces en la opción “Número de compilación” para activar y mostrar las “Opciones de desarrollador”.


48% 3:51 PM
Info. software
Versión de Android
6.0.1
Versión de banda base
Versión de Kernel
Número de compilación
Estado de SE para Android
Versión de KNOX
KNOX 2.6
Standard SDK 5.6.0
Premium SDK 2.6.0
Customization SDK 2.6.0
Container 2.6.0
CEP 2.0.1
Enterprise Billing 1.2.0

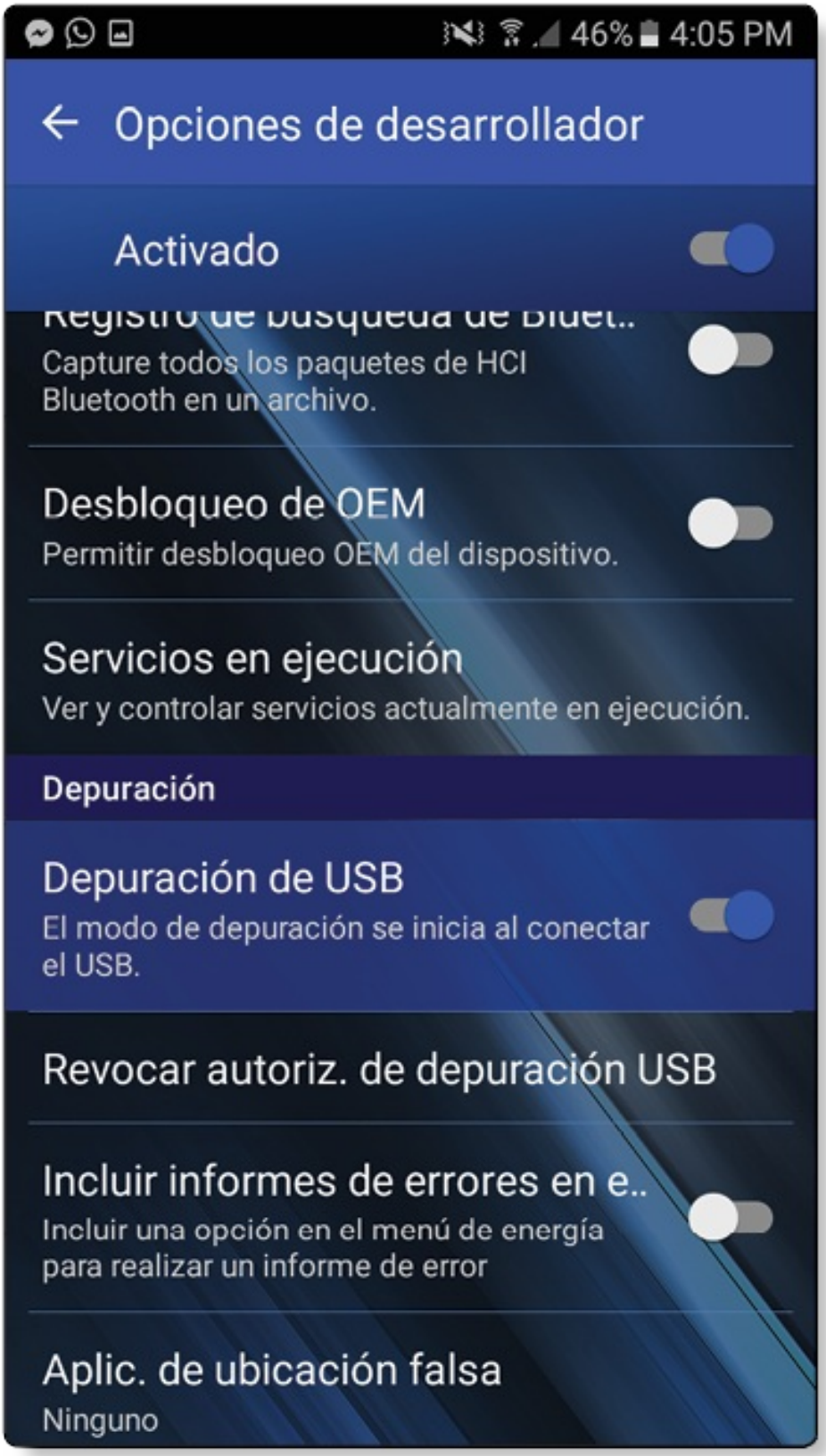
Info. software en dispositivo móvil

Regrese a los Ajustes para observar las “Opciones de desarrollador”.


Ajustes
BUSCAR
Cuentas
Google
Realizar copia de seguridad y..
Idioma y entrada
Batería
Almacenamiento
Accesorios
Fecha y hora
Manual de usuario
Opciones de desarrollador
Acerca del dispositivo

Opciones de desarrollador activas en dispositivo móvil

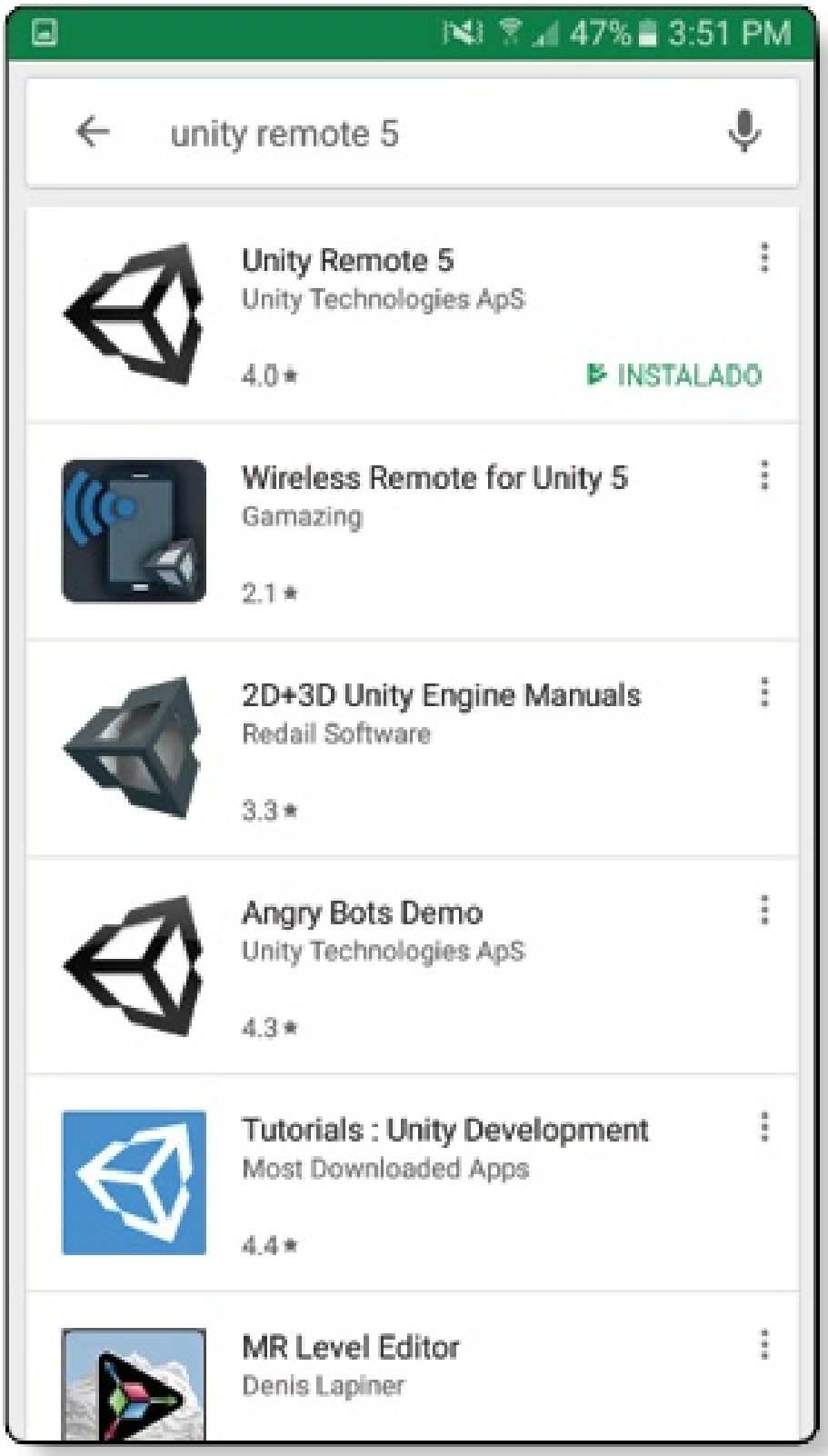
Vaya a “Opciones de desarrollador” y active su opción “Depuración de USB”.


46% 4:05 PM
Opciones de desarrollador
Activado
Registro de búsqueda de Bluet..
Capture todos los paquetes de HCI Bluetooth en un archivo.
Desbloqueo de OEM
Permitir desbloqueo OEM del dispositivo.
Servicios en ejecución
Ver y controlar servicios actualmente en ejecución.
Depuración
Depuración de USB
El modo de depuración se inicia al conectar el USB.
Revocar autoriz. de depuración USB
Incluir informes de errores en e..
Incluir una opción en el menú de energía para realizar un informe de error
Aplic. de ubicación falsa
Ninguno

Activación de “Depuración de USB” en dispositivo móvil

A continuación, descargará e instalará la aplicación móvil para poder previsualizar el proyecto de Unity a tiempo real en su dispositivo móvil

En su dispositivo vaya a GooglePlay, busque y descargue la aplicación “UnityRemote”.


47% 3:51 PM
unity remote 5
Unity Remote 5
Unity Technologies ApS
4.0★
INSTALADO
Wireless Remote for Unity 5
Gamazing
2.1★
2D+3D Unity Engine Manuals
Redail Software
3.3★
Angry Bots Demo
Unity Technologies ApS
4.3★
Tutorials : Unity Development
Most Downloaded Apps
4.4★
MR Level Editor
Denis Lapiner

Búsqueda de Unity Remote en GooglePlay

unity Remote 5
47% 3:51 PM
Connect this device with a USB Cable to your computer. Press PLAY in the Unity Editor to test.
Select a device to use in:
Edit > Project Settings > Editor
Unity Remote 5
Unity Technologies ApS
Todos
DESINSTALAR
ABRIR
500
MIL
Descargas
4.0
9,994
Herramientas
Similares
Unidad Remota 5 en su dispositivo Android para probar su juego en vivo en el Editor de la unidad
NOVEDADES
Added support for gamepads connected to the remote device (requires Unity 5.4)

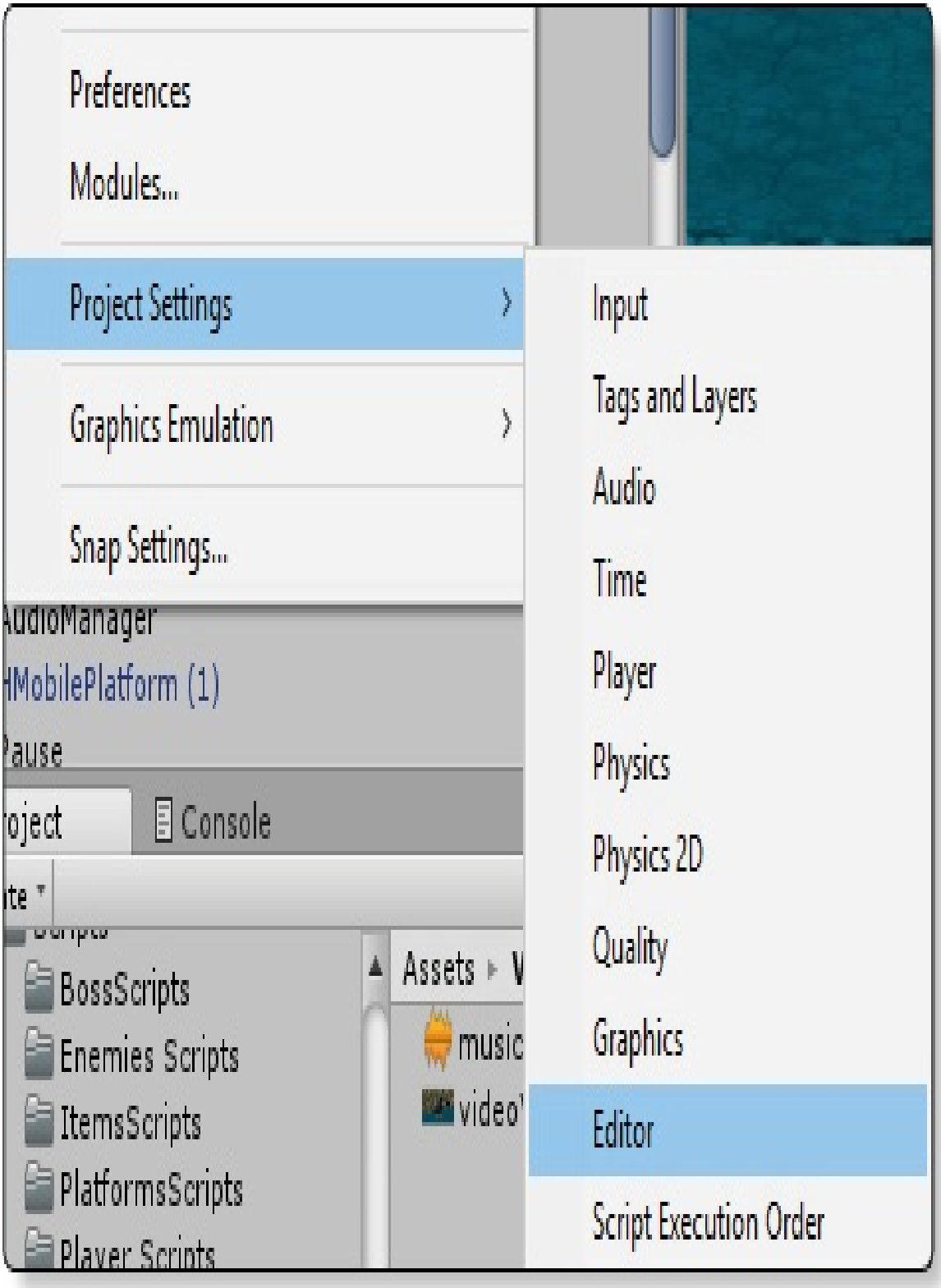
Unity Remote en GooglePlay

NOTA:

Esta versión de la aplicación es la más actualizada al momento de escribir este libro, si existe una más nueva descárguela.

A continuación, configurará Unity para que pueda visualizar su juego en su dispositivo móvil.

Vaya a Unity y asigne en el menú general “Edit/ Project Setting / Editor”.


Preferences
Modules...
Project Settings
Graphics Emulation
Snap Settings...
Input
Tags and Layers
Audio
Time
Player
Physics
Physics 2D
Quality
Graphics
Editor
Script Execution Order
AudioManager
HMobilePlatform (1)
Pause
Console
Assets
BossScripts
Enemies Scripts
ItemsScripts
PlatformsScripts
Player Scripts
music
video

Editor

En Inspector asigne los siguientes valores.

Inspector
Editor Settings
Unity Remote
Device
Any Android Device
Compression
JPEG
Resolution
Downsize
Joystick Source
Remote

Valores del editor

Conecte el dispositivo móvil con su cable USB a la computadora.

NOTA:

Para que funcione la aplicación, es necesario que cierre Unity, abrir la aplicación en el dispositivo y en seguida abrir Unity.

Abra la aplicación Unity Remote en su dispositivo.

Abra Unity.

Previsualice el juego para observar que corre en su dispositivo móvil.

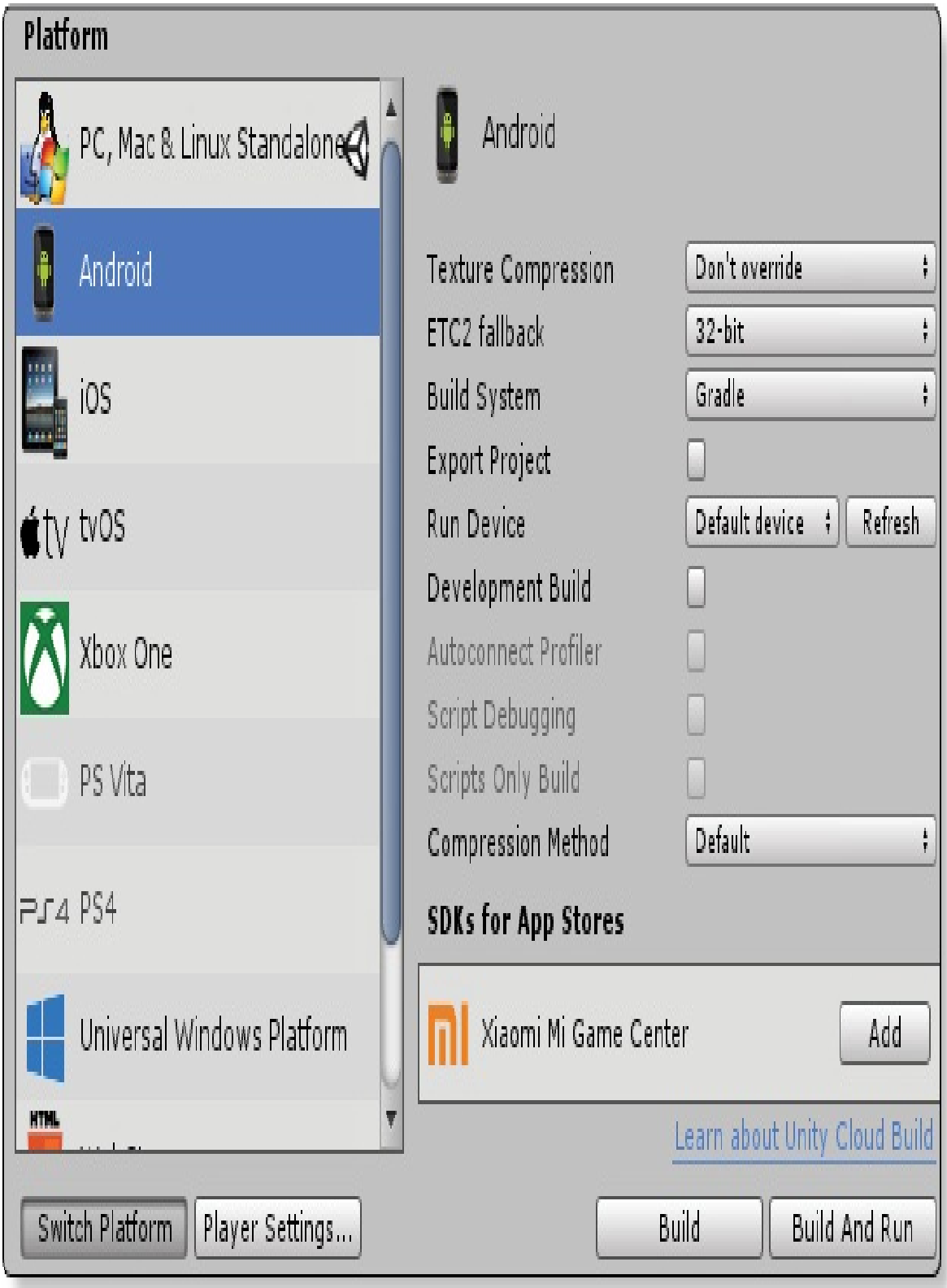
Programación de controles para dispositivos móviles

En este apartado crearemos y programaremos los controles de nuestro videojuego para los dispositivos móviles.

Nótese que cuando previsualizamos el juego en nuestro dispositivo móvil, los controles siguen siendo relativos a los que previamente programamos, es decir, su entrada es a través del teclado o de un controlador externo conectado vía USB.

En primera instancia debemos de cambiar la plataforma a Android en Unity.

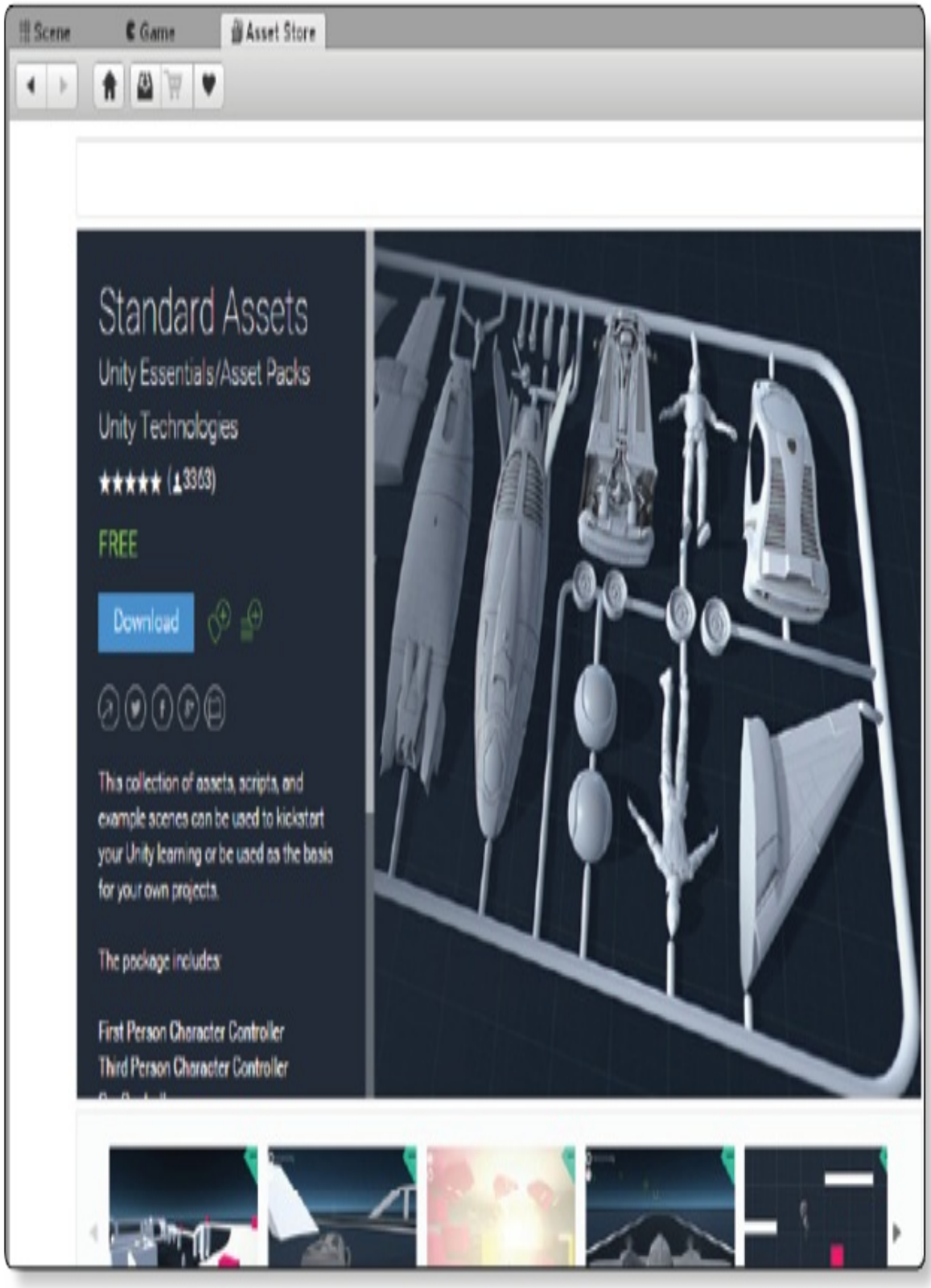
En Unity vaya a “BuildSettings”, seleccione como plataforma “Android” y clic en el botón “Switch Platform”.


Platform
PC, Mac & Linux Standalone
Android
iOS
tvOS
Xbox One
PS Vita
PS4
Universal Windows Platform
Android
Texture Compression
Don't override
ETC2 fallback
32-bit
Build System
Gradle
Export Project
Run Device
Default device
Refresh
Development Build
Autoconnect Profiler
Script Debugging
Scripts Only Build
Compression Method
Default
SDKs for App Stores
Xiaomi Mi Game Center
Add
Learn about Unity Cloud Build
Switch Platform
Player Settings...
Build
Build And Run

Cambio de plataforma a Android

A continuación, descargaremos una librería de “Assets” desde el “Asset Store” de Unity para asignar elementos de entrada para diversas plataformas.

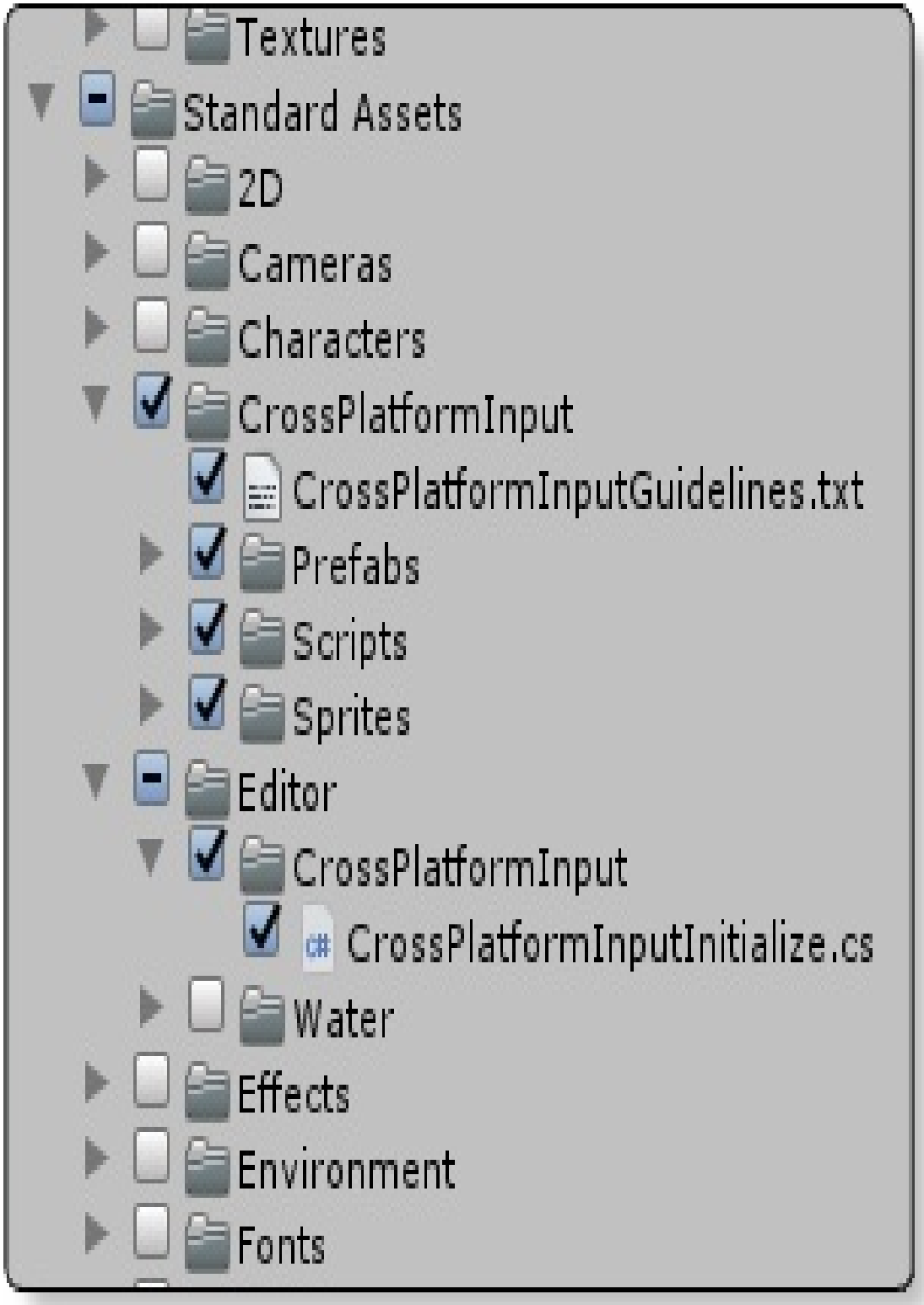
Vaya a “Asset Store”, busque el paquete “Standard Assets” y descárguelo al proyecto.


Scene
Game
Asset Store
Standard Assets
Unity Essentials/Asset Packs
Unity Technologies
(3303)
FREE
Download
This collection of assets, scripts, and example scenes can be used to kickstart your Unity learning or be used as the basis for your own projects.
The package includes
First Person Character Controller
Third Person Character Controller

Paquetería de Standard Assets en Asset Store

Al asignar clic en “Dowload” se abrirá una ventana con los paquetes que podemos importar al proyecto, únicamente elegirá los relativos a “CrssPlatformInput”.

Seleccione los objetos que se muestran y clic en “Import”.


Textures
Standard Assets
2D
Cameras
Characters
CrossPlatformInput
CrossPlatformInputGuidelines.txt
Prefabs
Scripts
Sprites
Editor
CrossPlatformInput
CrossPlatformInputInitialize.cs
Water
Effects
Environment
Fonts

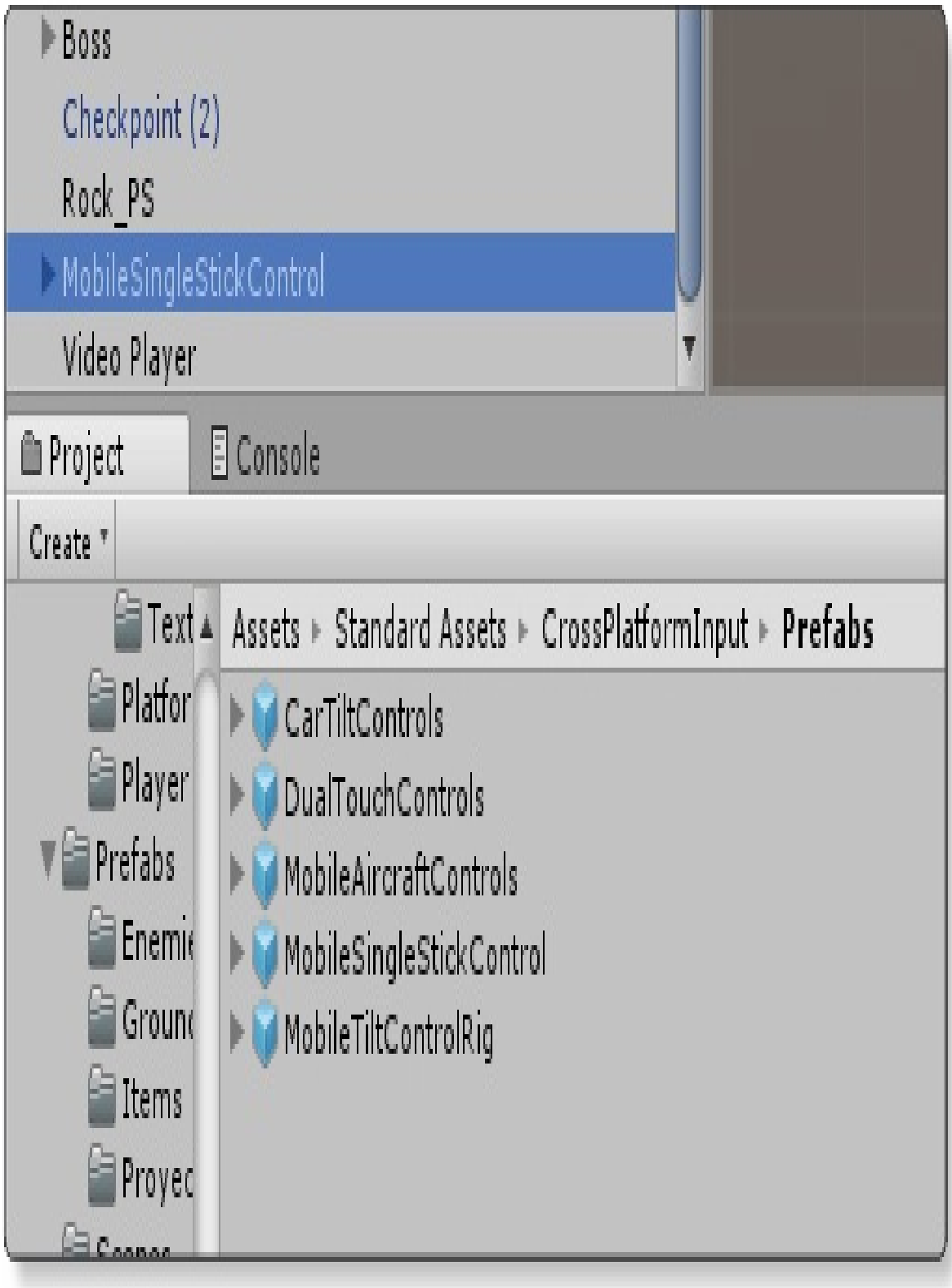
Selección de paquetería “CrossPlatformInput”

Vaya a la ventana “Project” para observar que los elementos se han importado satisfactoriamente.

Project
Console
Create
Fantas
FBX
Text
Platfor
Player
Prefabs
Enemi
Groun
Items
Assets
Standard Assets
CrossPlatfo...
Editor

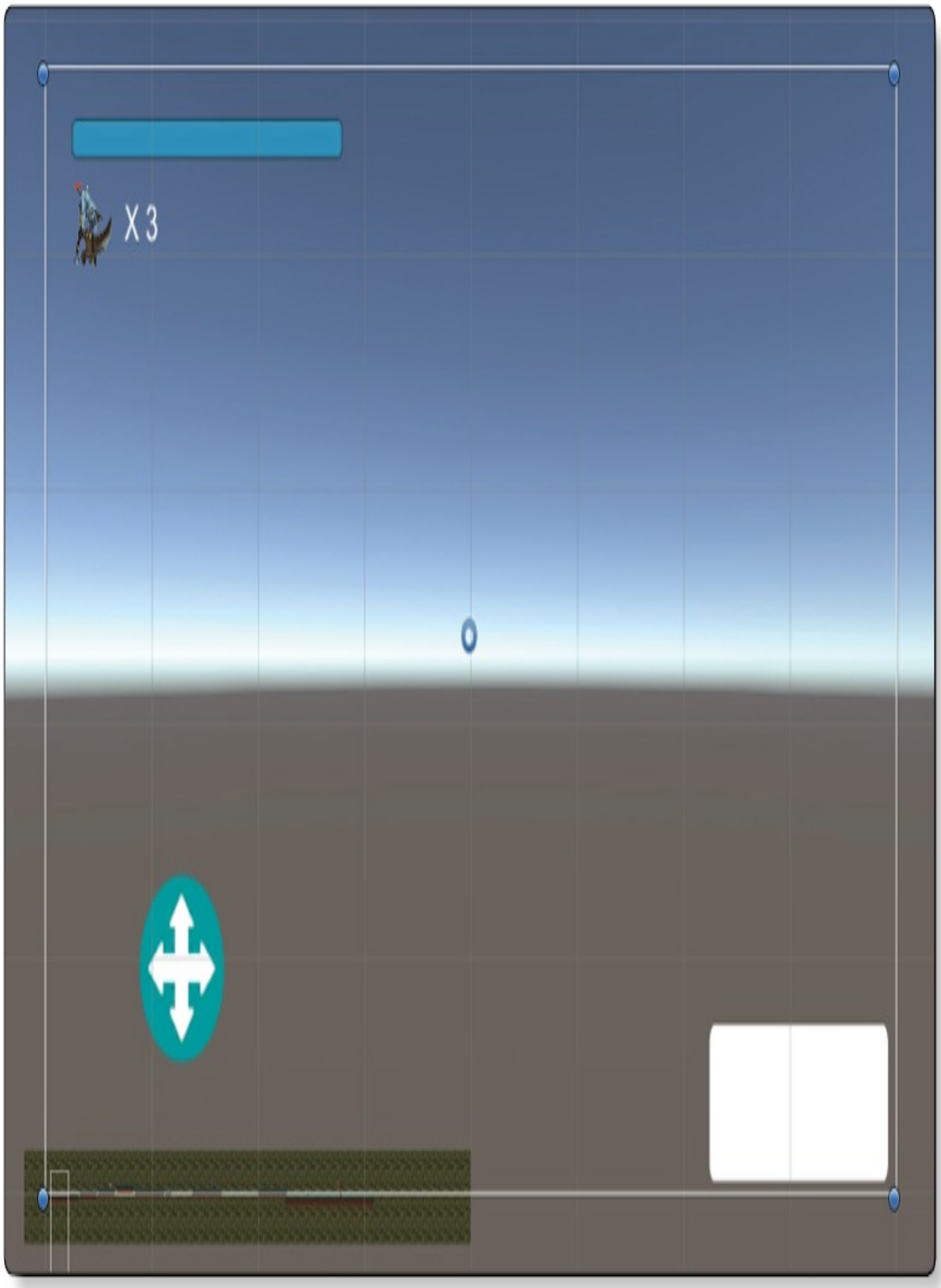
Paquetería correctamente importada

Vaya a la carpeta “Prefabs” dentro de la carpeta “CrossPlatfomrInput” y arrastre a la ventana de “Hierarchy” el objeto “MobileSingleStickControl” como se muestra.


Boss
Checkpoint (2)
Rock_PS
MobileSingleStickControl
Video Player
Project
Console
Create
Assets ▸ Standard Assets ▸ CrossPlatformInput ▸ Prefabs
CarTiltControls
DualTouchControls
MobileAircraftControls
MobileSingleStickControl
MobileTiltControlRig
Prefabs

Añadir “MobileSingleStickControl” a “Hierarchy”

Reacomode lo objetos en la vista “Scene” como se muestra y previsualice para observarlos en el juego.


X 3

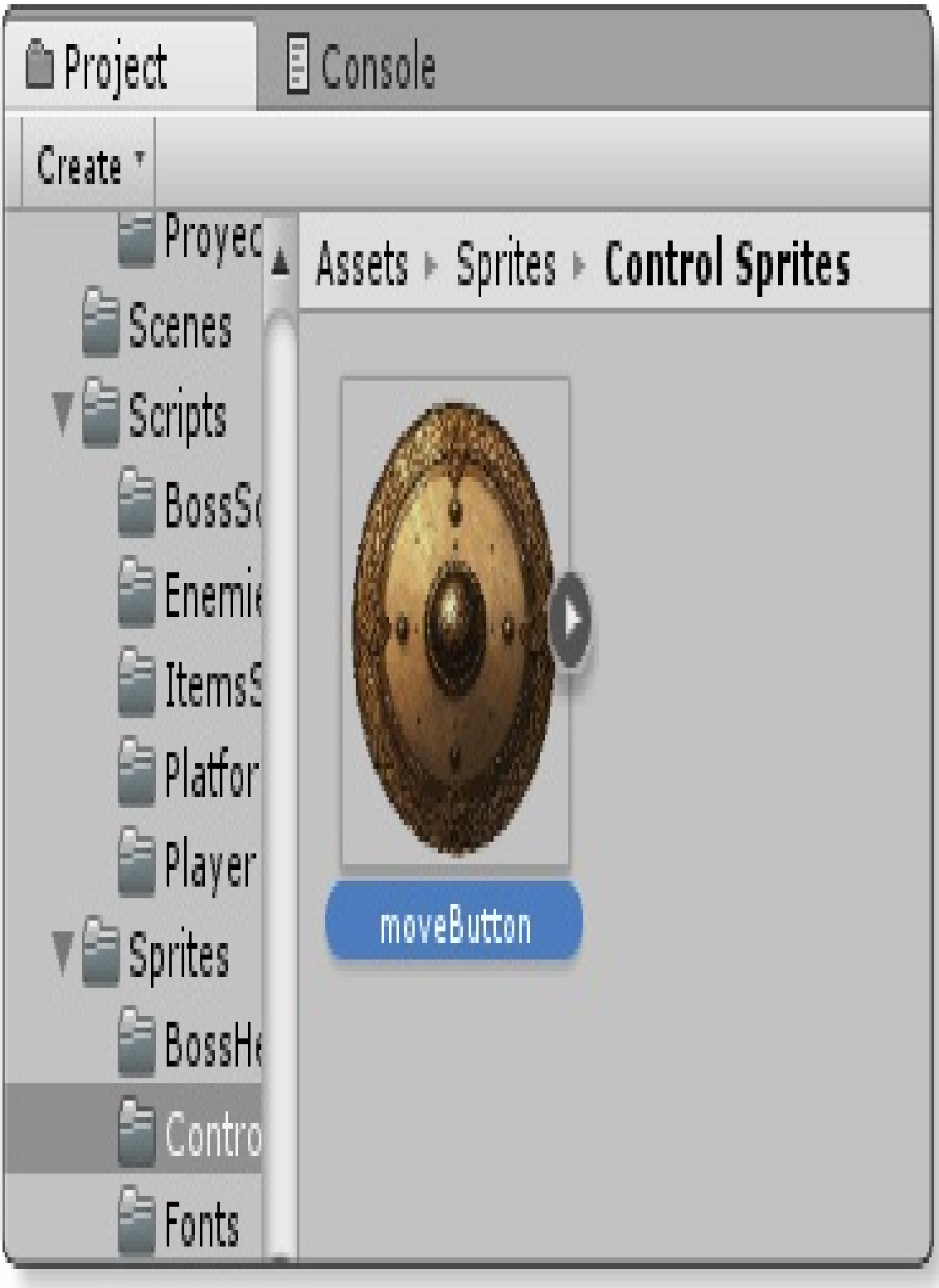
Colocación de elementos

X2

Previsualización de controles virtuales standard

A continuación, queremos cambiar el icono del joystick por un gráfico más adecuado a nuestro juego.

Importe de la carpeta de “resources” del capítulo el sprite “moveButton”, colóquelo en la carpeta “Sprites” de “Assets” y conviértalo en una textura tipo “UI Sprite”.

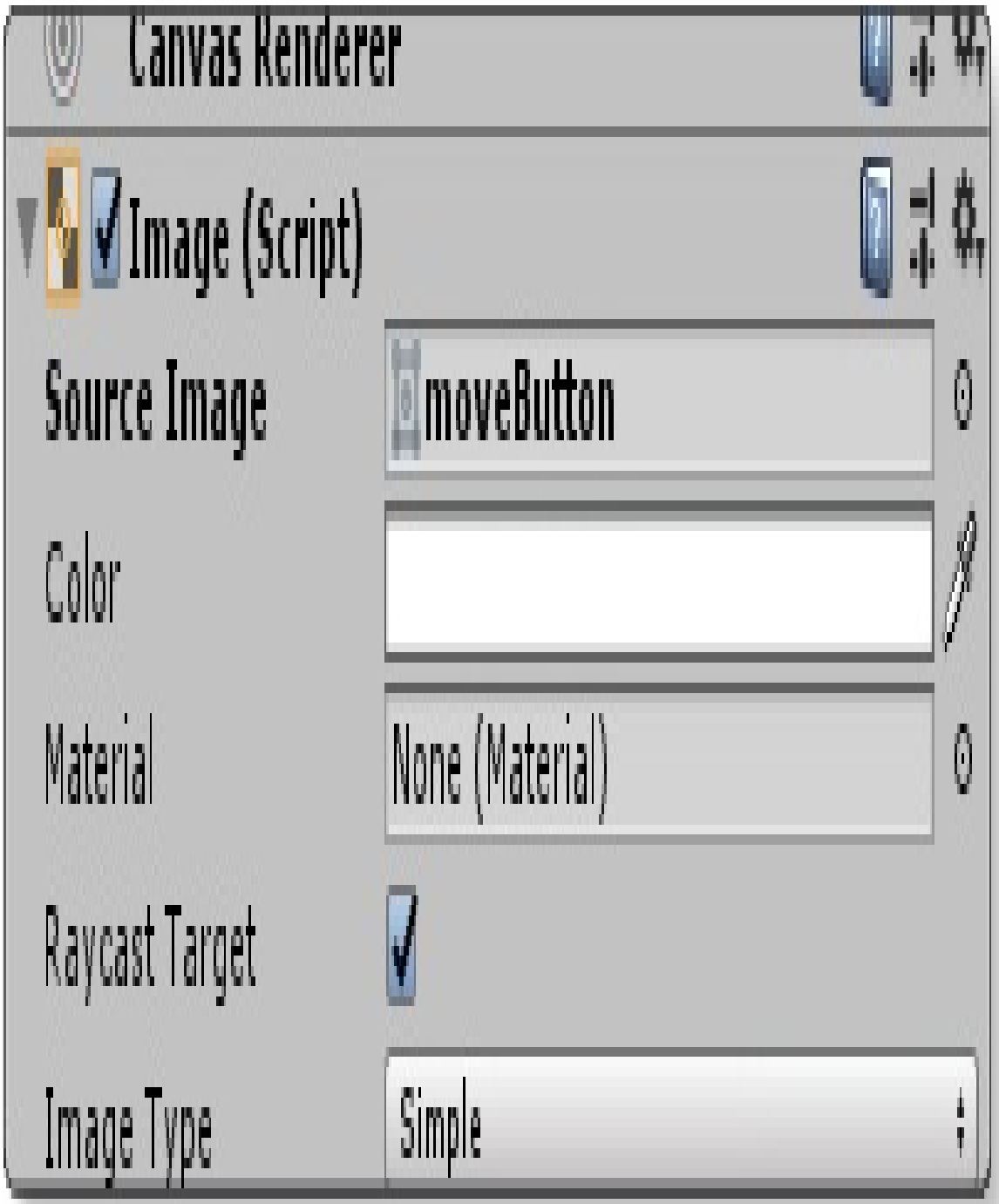

Project
Console
Create
Proyec
Scenes
Scripts
BossSc
Enemie
ItemsS
Platfor
Player
Sprites
BossHe
Contro
Fonts
Assets ▸ Sprites ▸ Control Sprites
moveButton

Creación de sprite “moveButton”

En “Hierarchy” expanda el objeto “MobileSingleStickControl”, seleccione “MobileJoystick” y en “Inspector” cambie el grafico como se muestra.

Rock_PS
MobileSingleStickControl
JumpButton
MobileJoystick
Video Player

Selección de objeto “MobileJoystick”

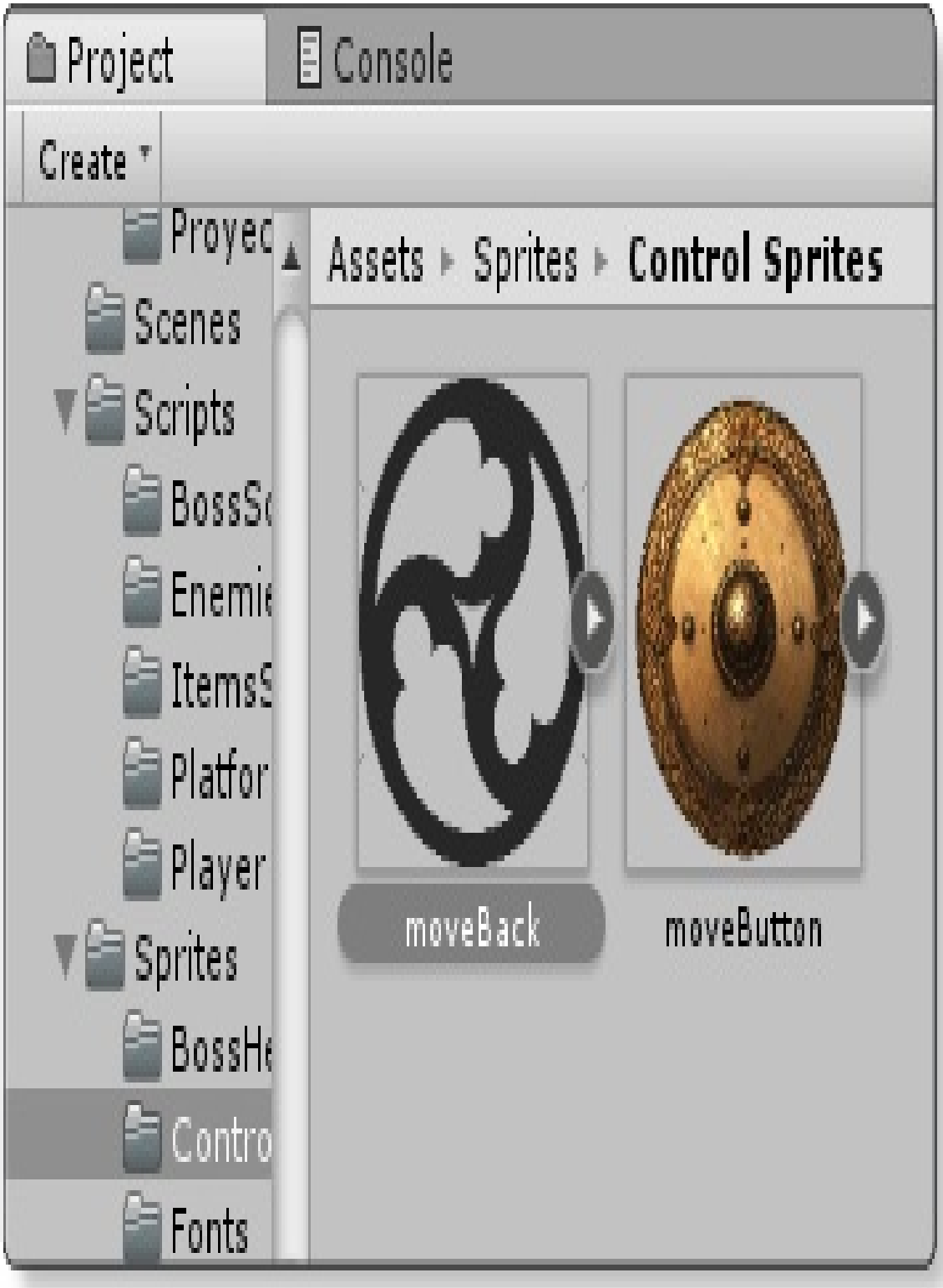

Canvas Renderer
Image (Script)
Source Image
moveButton
Color
Material
None (Material)
Raycast Target
Image Type
Simple

Asignación de gráfico nuevo

Previsualización de nuevo gráfico

A continuación, vamos a crear un gráfico “Background” para el joystick de movimiento.

Importe al proyecto el sprite “moveBack” de la carpeta de “resources” a la carpeta de “Sprites” y conviértalo en una textura tipo “UI” como se muestra.


Project
Console
Create
Assets ▸ Sprites ▸ Control Sprites
Scenes
Scripts
Sprites
Fonts
moveBack
moveButton

Sprite moveBack

En “Hierarchy” seleccione el objeto “MobileSingleStickControl”, asigne clic derecho para crear un nuevo “UI / Image” y renómbrela como “BackgroundMove”.

Seleccione la “BackgroundMove” y en “Inspector” asigne el sprite “moveBack”.

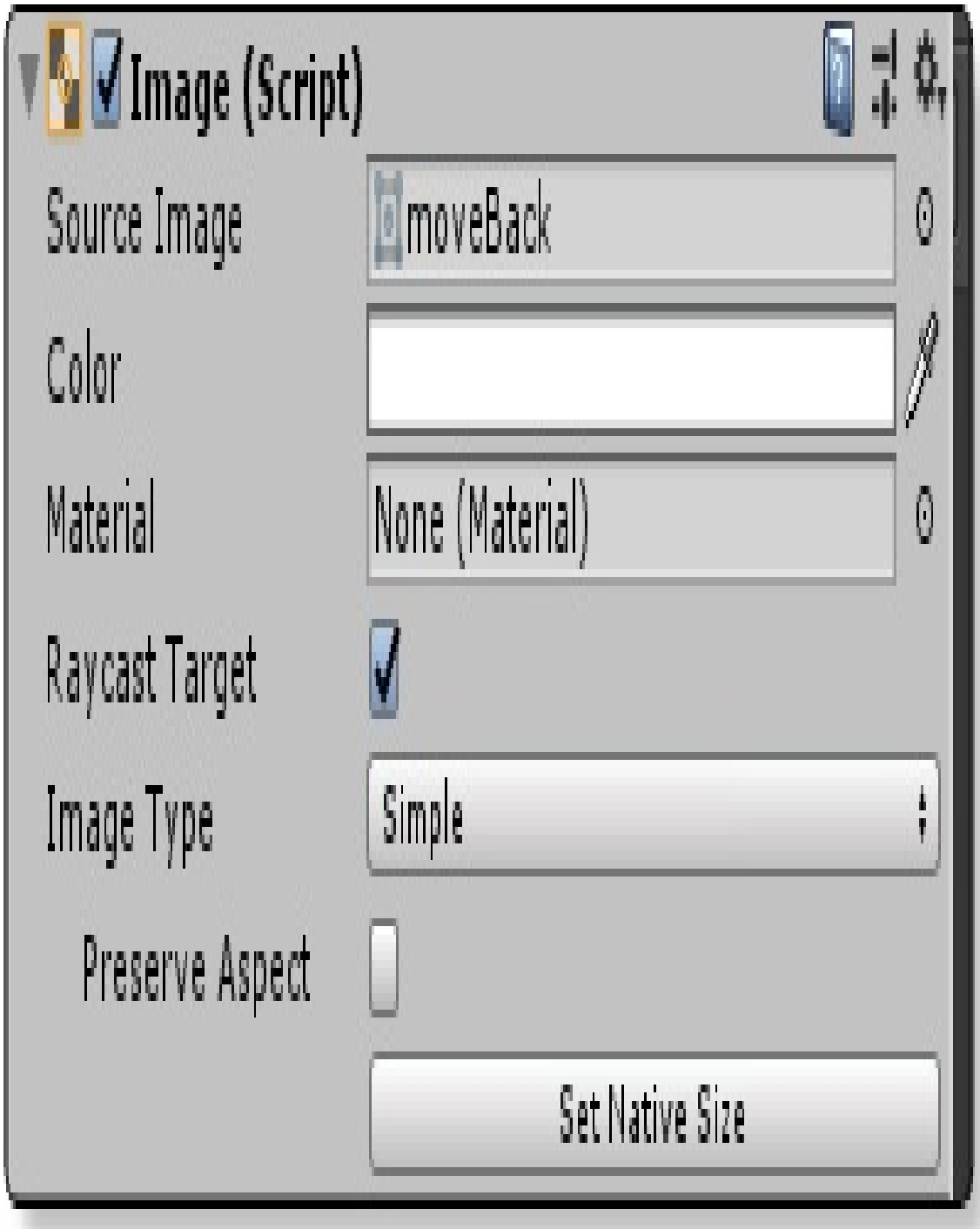
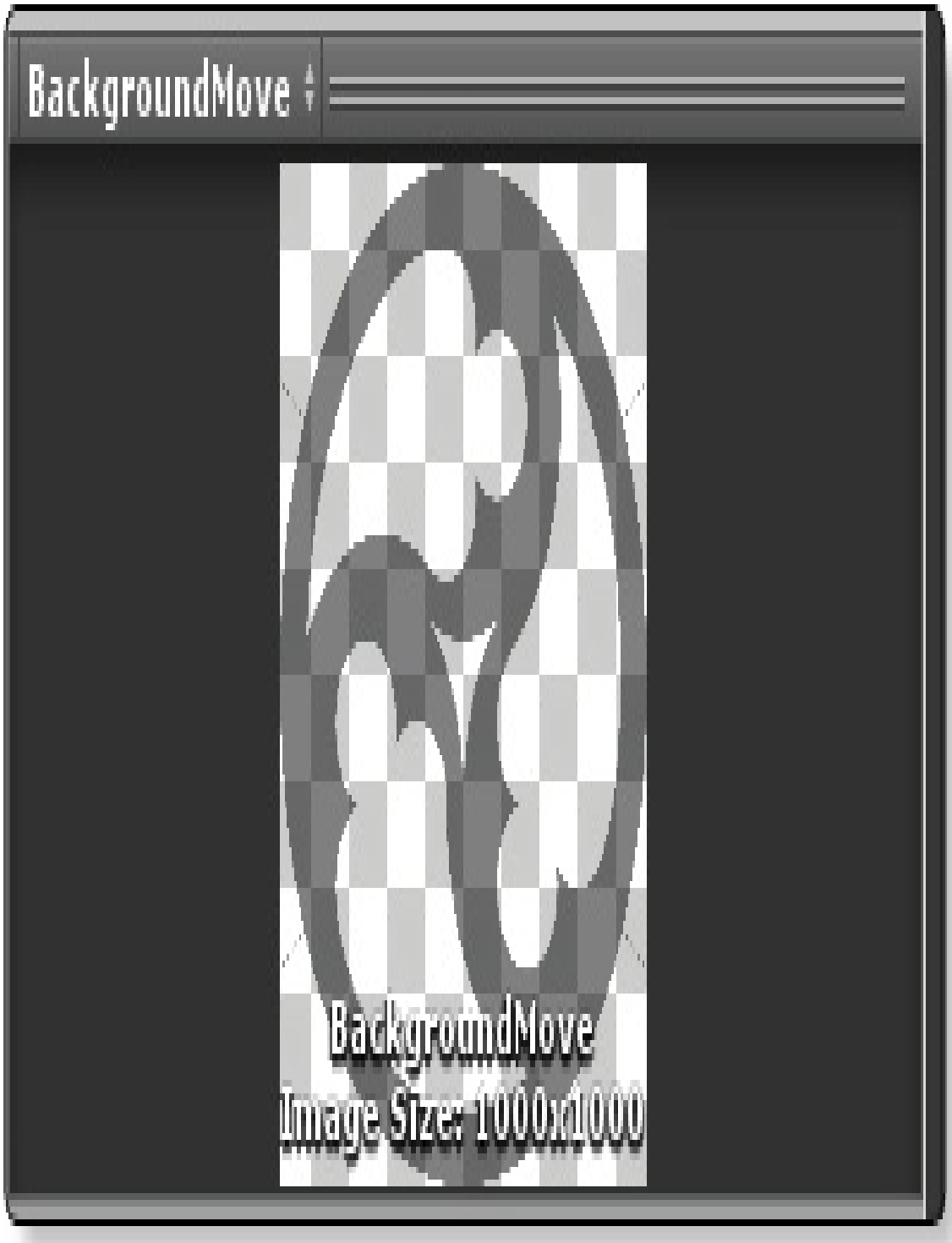


Image (Script)
Source Image
moveBack
Color
Material
None (Material)
Raycast Target
Image Type
Simple
Preserve Aspect
Set Native Size

Reemplazo de sprite

NOTA:

El grafico “moveBack” es un archivo .png el cual tiene una baja opacidad para que se logre ver un poco transparente.


BackgroundMove
BackgroundMove
Image Size: 1000x1000

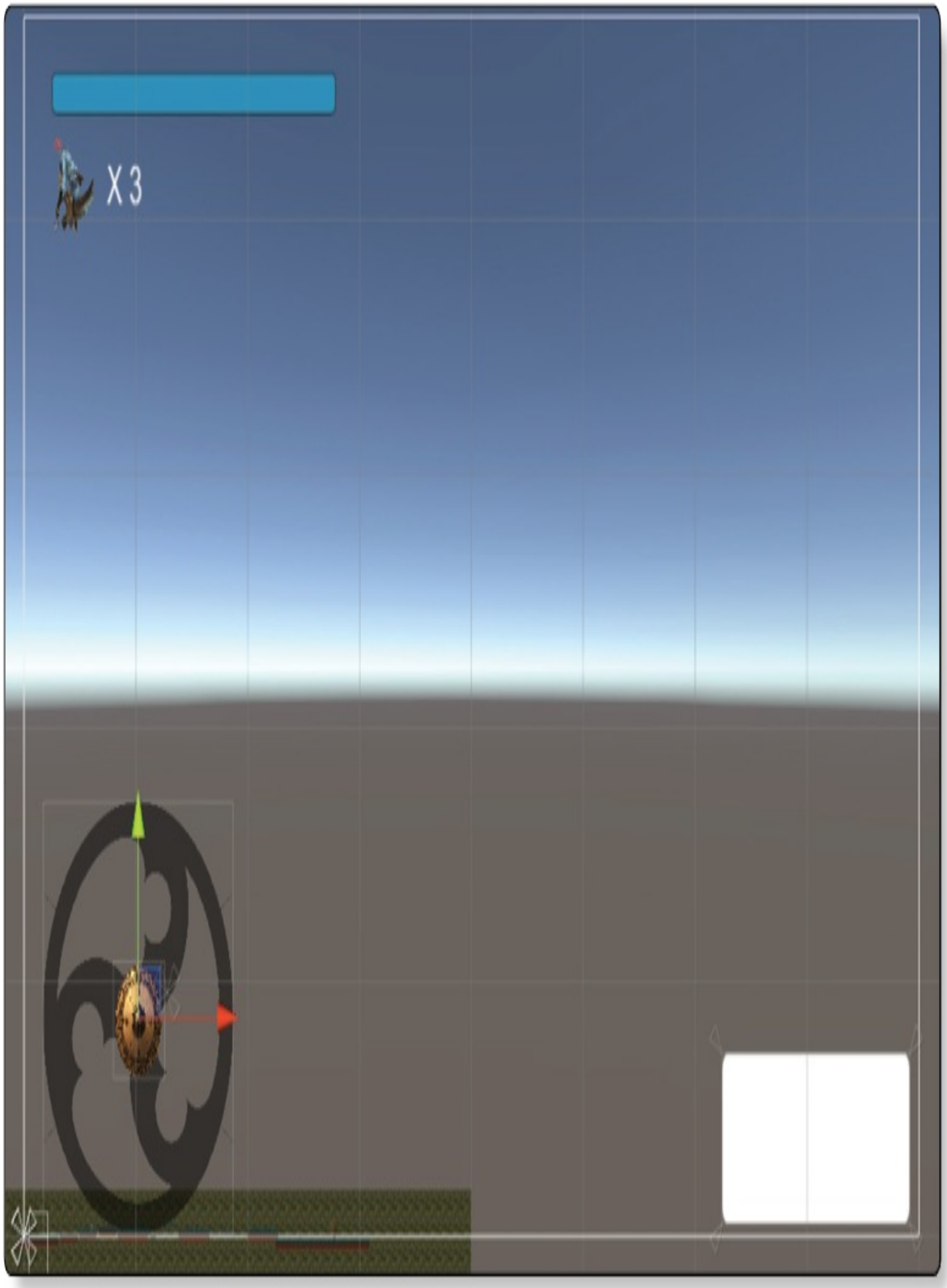
Sprite con transparencia

En la ventana de “Hierarchy” coloque más arriba el objeto “BackgroundMove” para que aparezca detrás del objeto “MobileJoystick”.

MobileSingleStickControl
BackgroundMove
MobileJoystick
JumpButton

Cambio de jerarquía visual

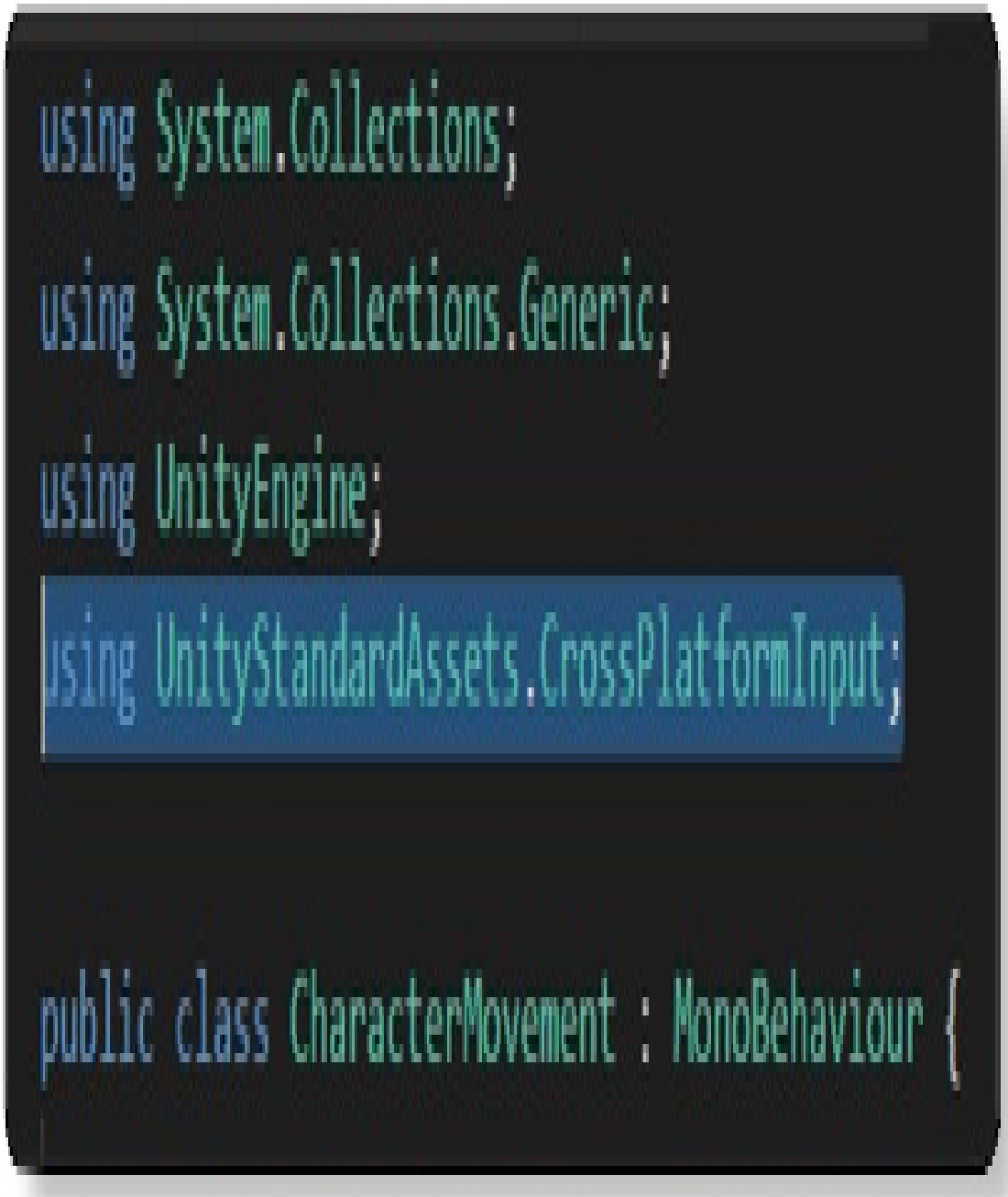
Ancle “MobileJoystick” en la esquina inferior izquierda y reacomode los elementos como se muestra.


X3

Posicionamiento de controles

A continuación, debemos de cambiar el código de programación de movimiento para que obedezca a los nuevos controles virtuales.

Vaya al script “CharacterMovement” y mande llamar la nueva librería como se muestra.

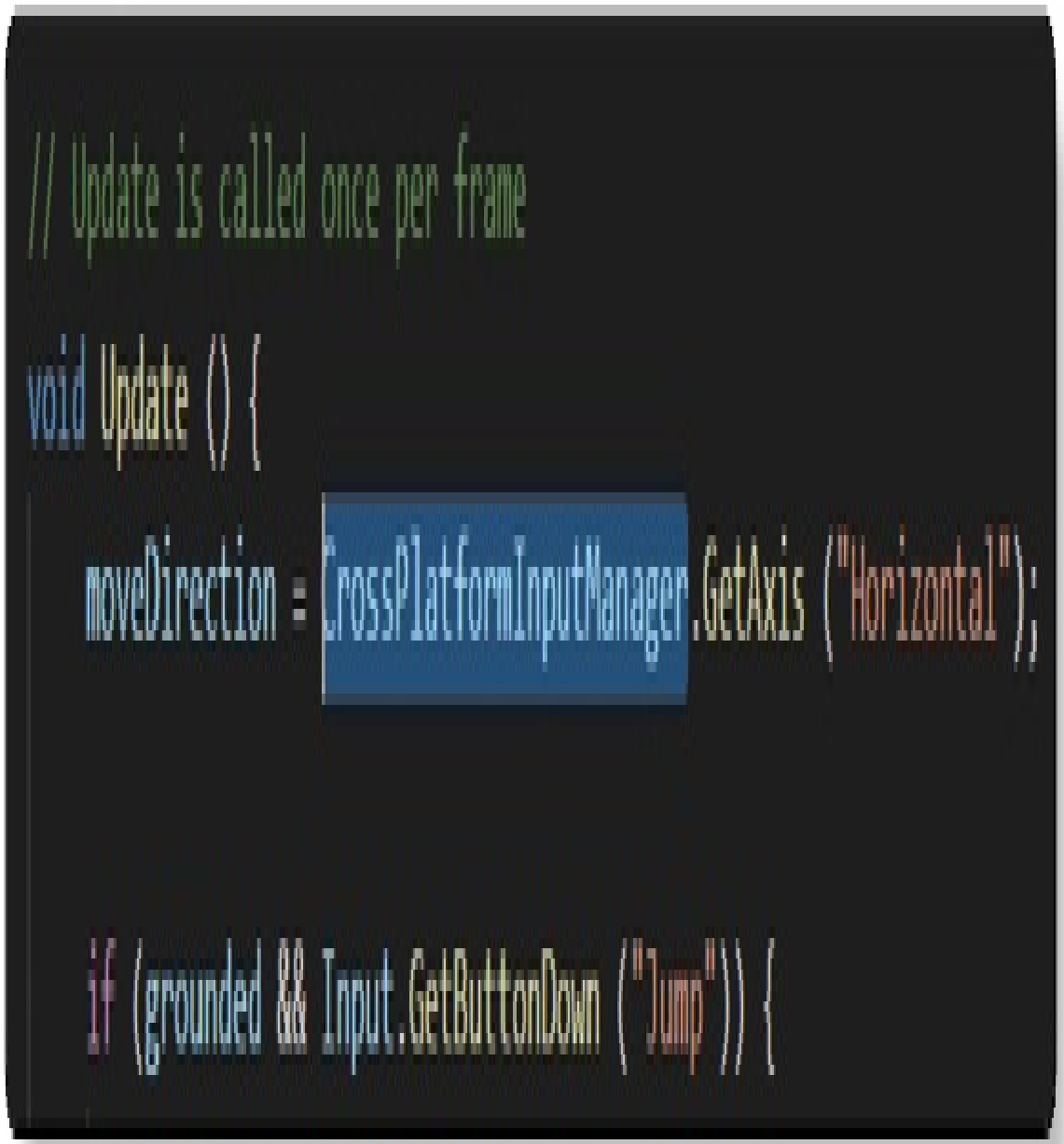

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;
public class CharacterMovement : MonoBehaviour {

Mandar llamar librería

NOTA:

Cuando utilicemos una librería externa, debemos de invocarla de esta manera.

Vaya a la función “Update” y edite el código como se muestra.


// Update is called once per frame
void Update () {
moveDirection = CrossPlatformInputManager.GetAxis ("Horizontal");
if (grounded && Input.GetButtonDown ("Jump")) {

Edición de código

Previsualice el juego para observar que el “Player” se mueve con el joystick virtual.

X2

Previsualización de control de movimiento

NOTA:

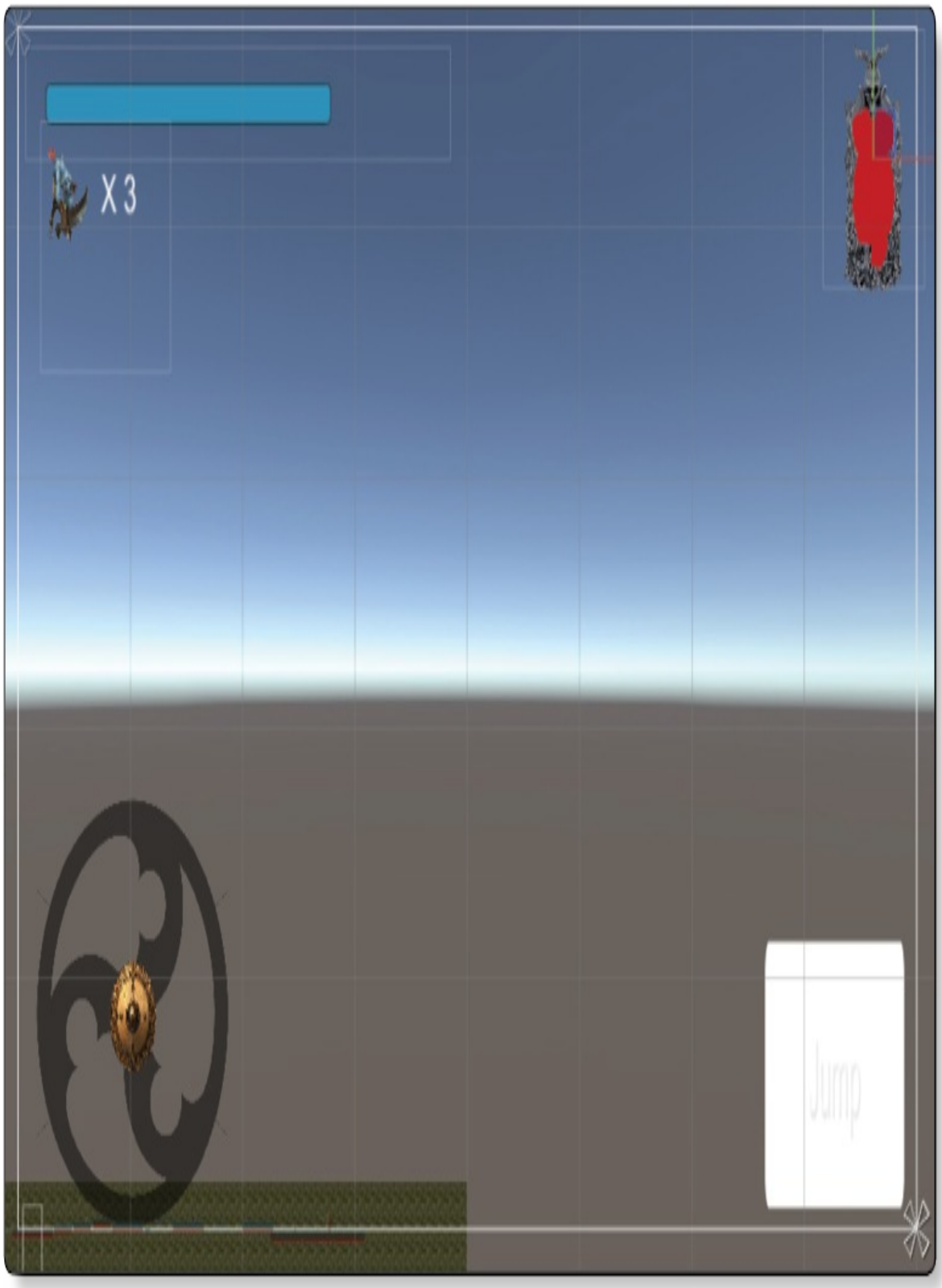
En este punto también puede previsualizarlo en su dispositivo móvil, es importante anotar que la previsualización realizada en su dispositivo a tiempo real, los gráficos no alcanzarán una calidad óptima hasta que el juego sea debidamente construido y exportado.

NOTA:

Una de las ventajas del Joystick digital es que puede controlar la velocidad del caminado adaptándose también en velocidad a la animación, es decir que, si mantenemos el joystick a la mitad del límite, podremos ver a nuestro “Player” correr más lento.

A continuación, realizaremos los cambios para la funcionalidad de ataque y de salto con botones virtuales.

Seleccione el objeto “JumpButton”, escálelo y reposicione como se muestra.

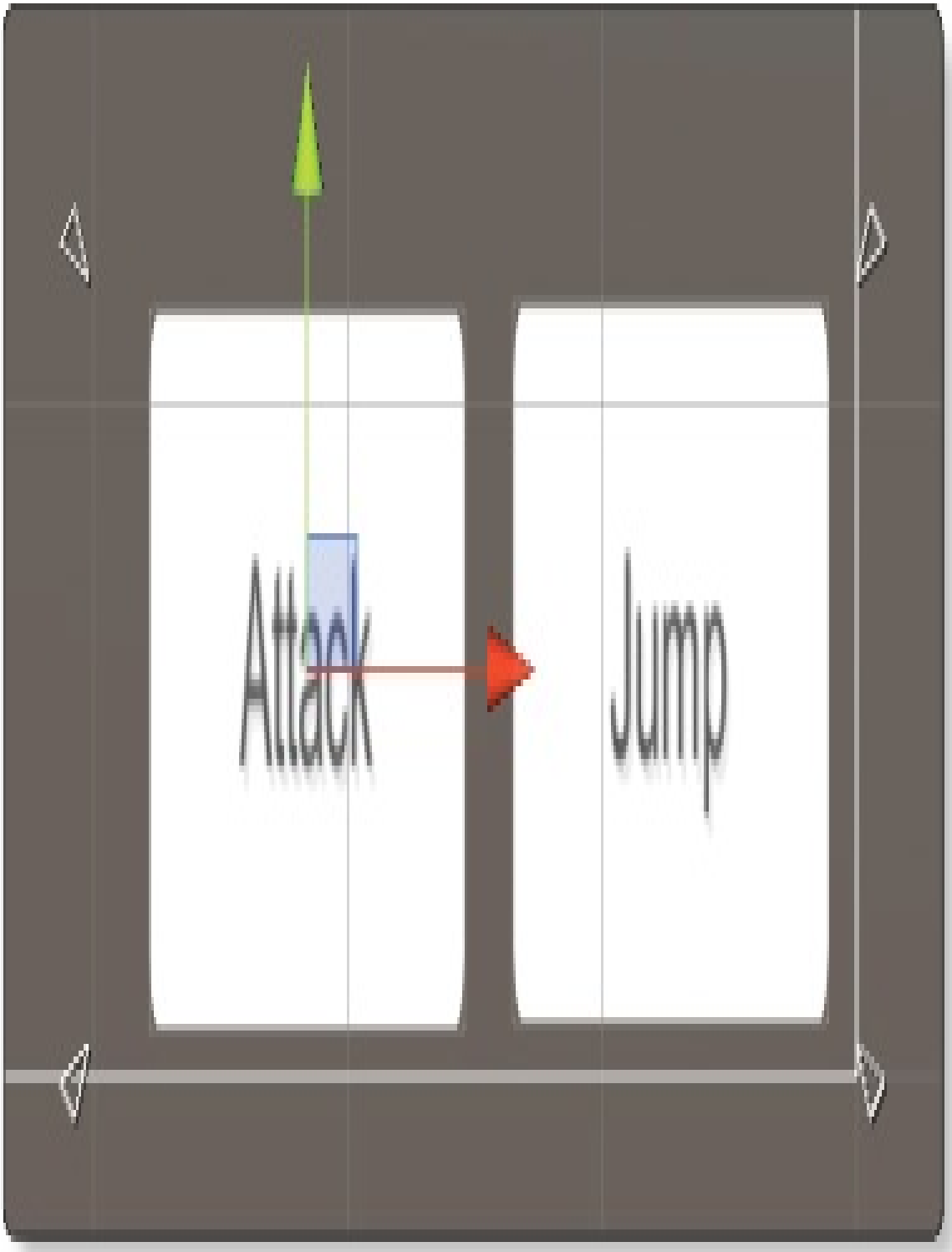

X3
Jump

Reescala de botón

NOTA:

Reposicione el indicador de vida del jefe en la parte superior derecha como se muestra.

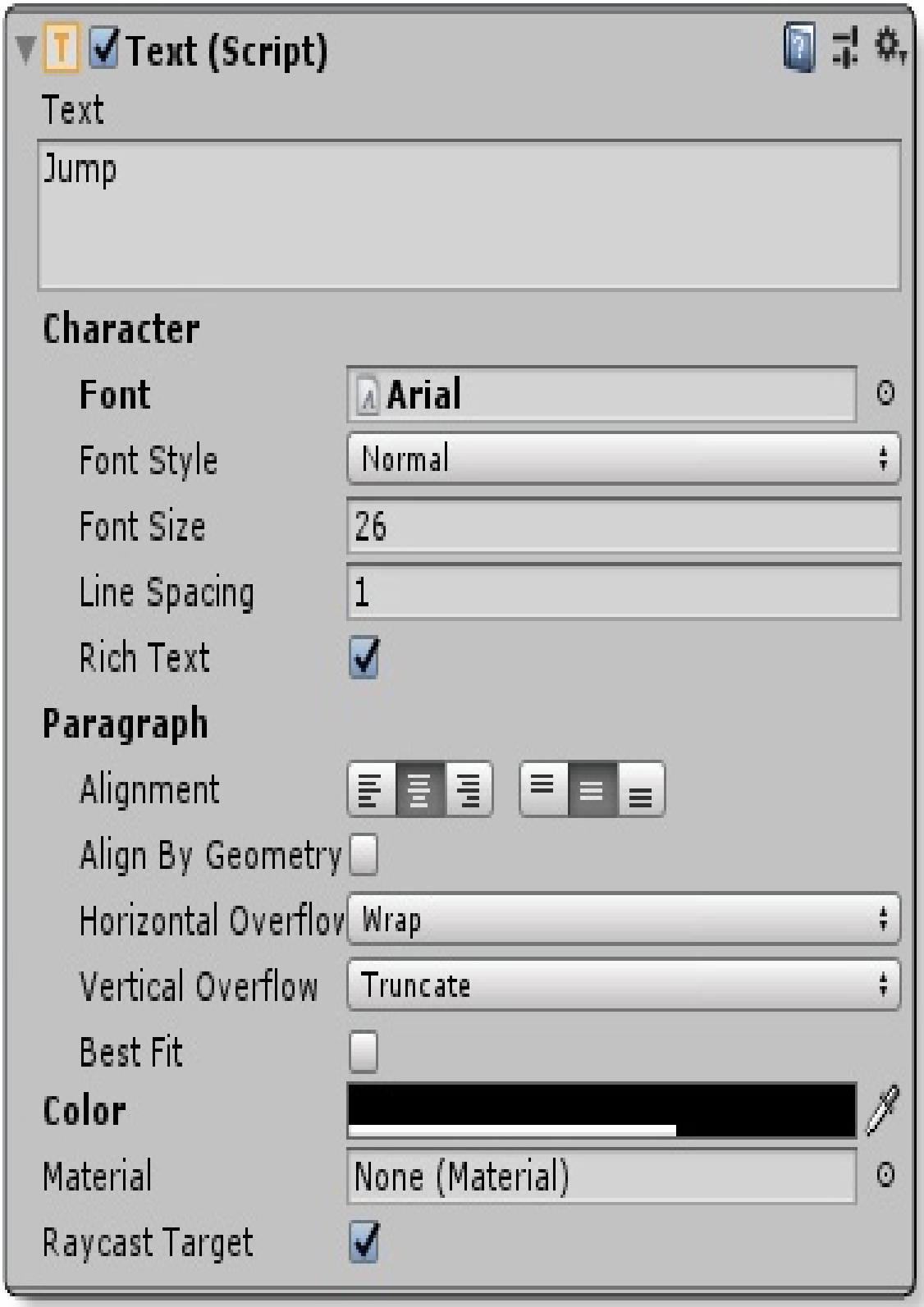
Haga un duplicado de “JumpButton”, renómbrelo “AttackButton”, colóquelos como se muestran.


Attack
Jump

Creación de segundo botón

NOTA:

Para cambiar la fuente, el tamaño y el color del texto, seleccione el subelemento del botón “Text” y en “Inspector” realice los cambios.

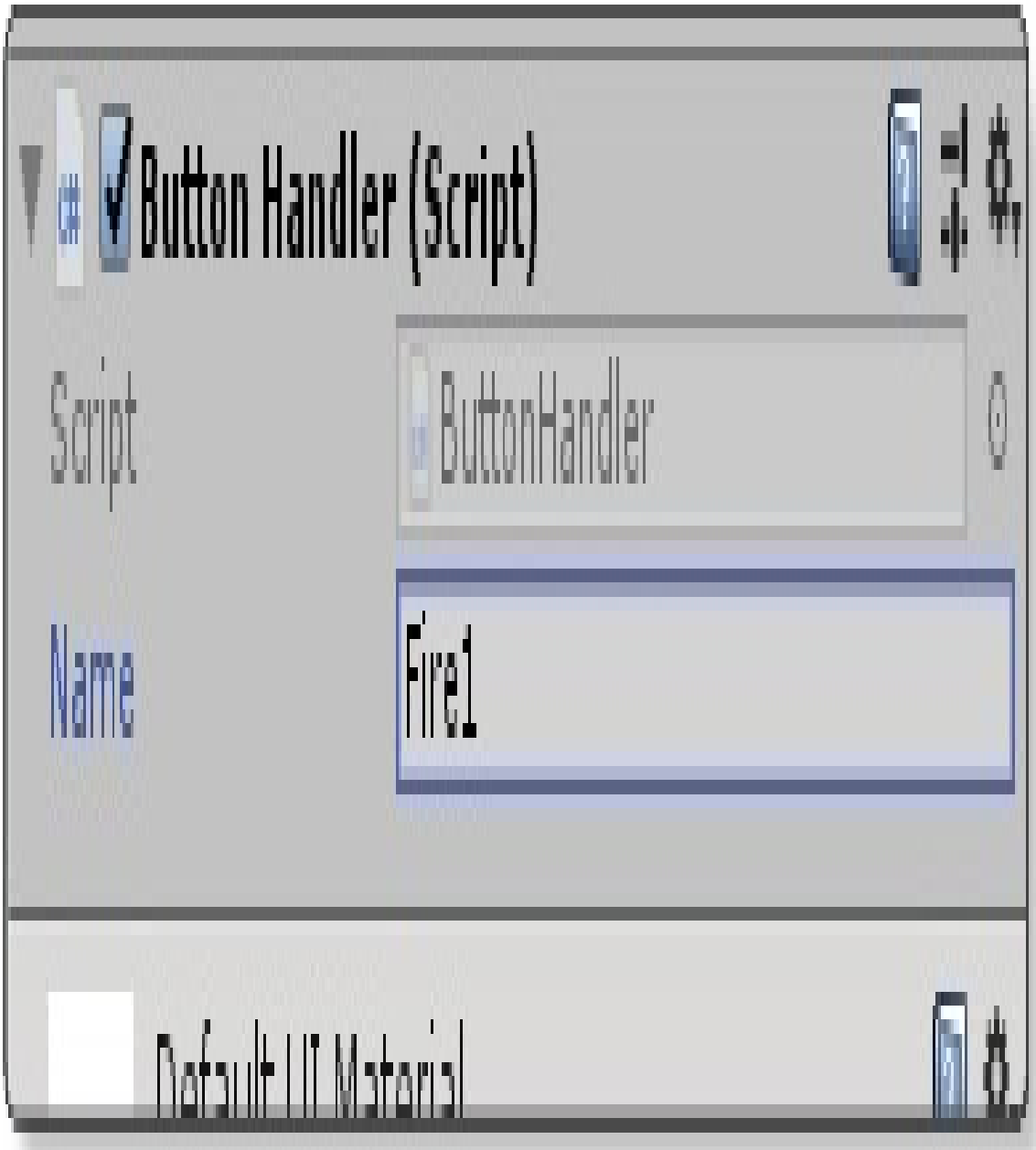

Text (Script)
Text
Jump
Character
Font
Arial
Font Style
Normal
Font Size
26
Line Spacing
1
Rich Text
Paragraph
Alignment
Align By Geometry
Horizontal Overflow
Wrap
Vertical Overflow
Truncate
Best Fit
Color
Material
None (Material)
Raycast Target

Cambio de valores en el texto del botón

Seleccione el nuevo objeto “AttackButton” y en “Inspector” renómbrelo en su componente script como “Fire1”.

MobileSingleStickControl
BackgroundMove
MobileJoystick
JumpButton
Text
AttackButton
Text
Video Player

Selección de “AttackButton”

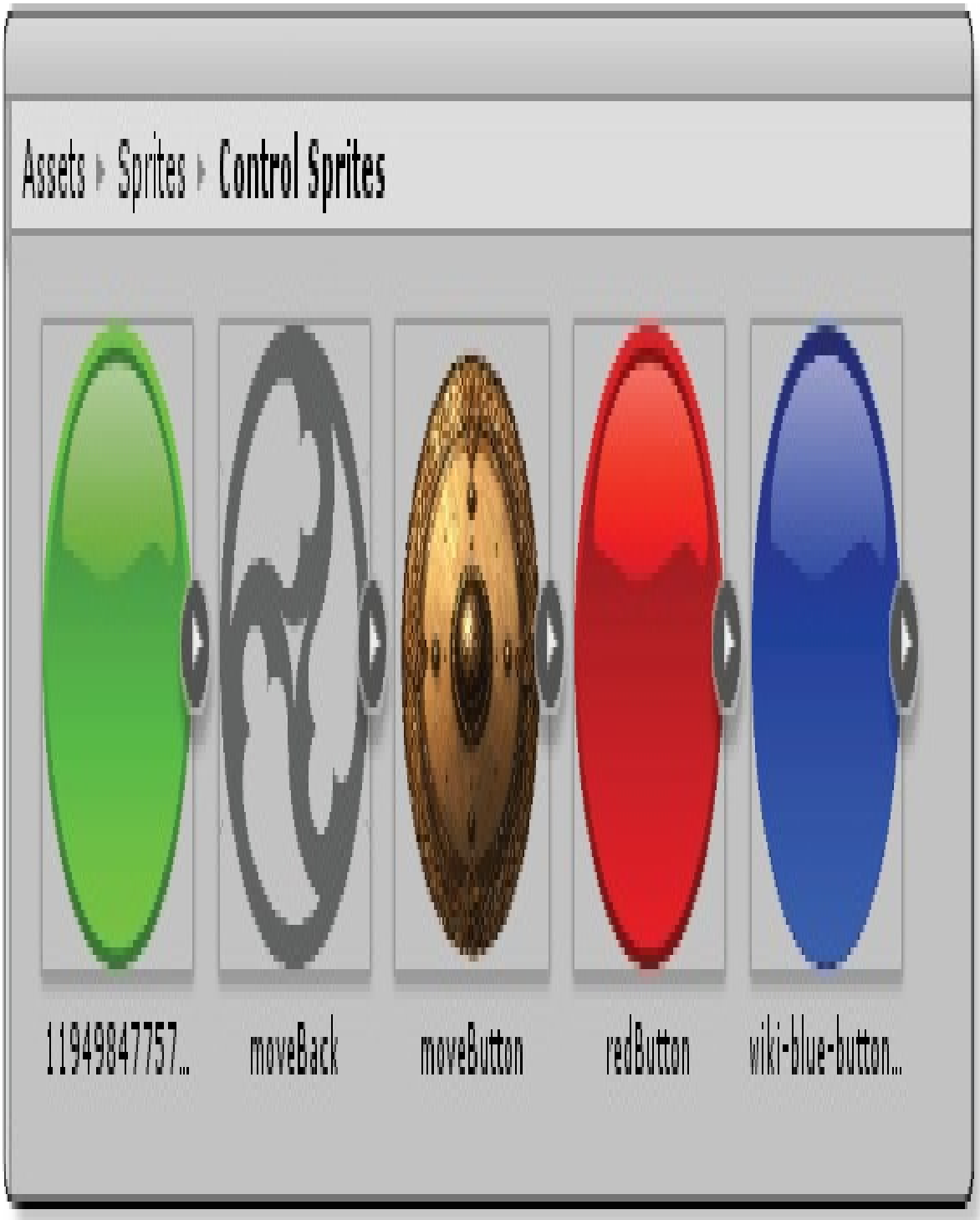

Button Handler (Script)
Script
ButtonHandler
Name
Fired
Default Material

Cambio de nombre en componente script

NOTA:

Es importante renombrarlo como se encuentra denominada la función por defecto, recordando que cuando mandamos llamar el comando preestablecido para el disparo del personaje, lo invocamos como “Fire1”.

Importe de la carpeta “resources” los gráficos referentes a los botones azul, verde y rojo, colóquelos en su carpeta correspondiente y conviértalos en sprites con textura UI como se muestra.


Assets ▸ Sprites ▸ Control Sprites
11949847757...
moveBack
moveButton
redButton
wiki-blue-button...

Importación de botones de colores

A continuación, vamos a programar eventos “OnClick” a los botones para que cambien de color cuando sean oprimidos virtualmente.

Reasigne el sprite del botón rojo a los dos botones en “Inspector” como se muestra y ánclelos en la esquina inferior derecha.

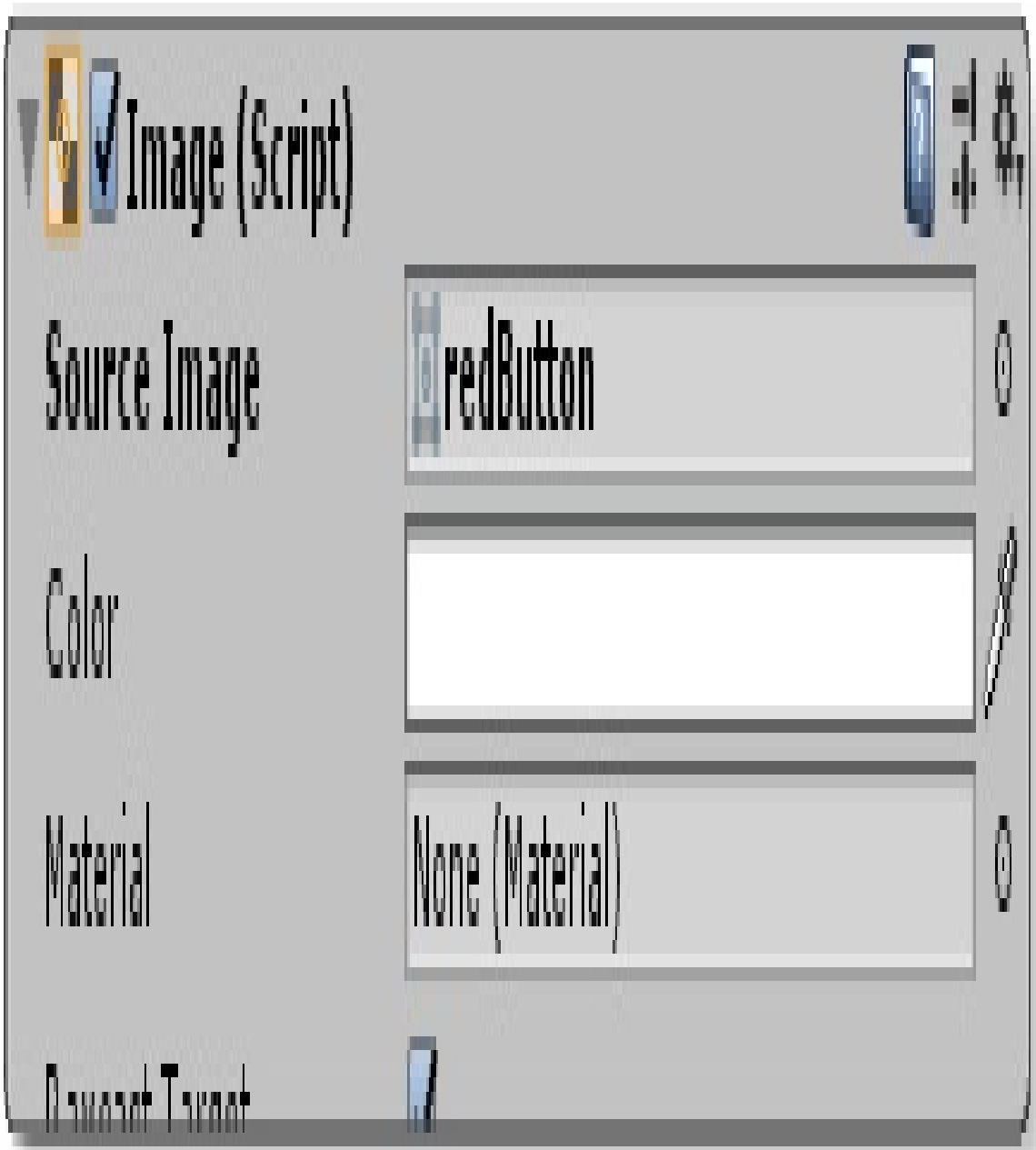


Image (Script)
Source Image
redButton
Color
Material
None (Material)
Raycast Target

Reasignación de sprite

Attack
Jump

Estados de botones conforme a su sprite

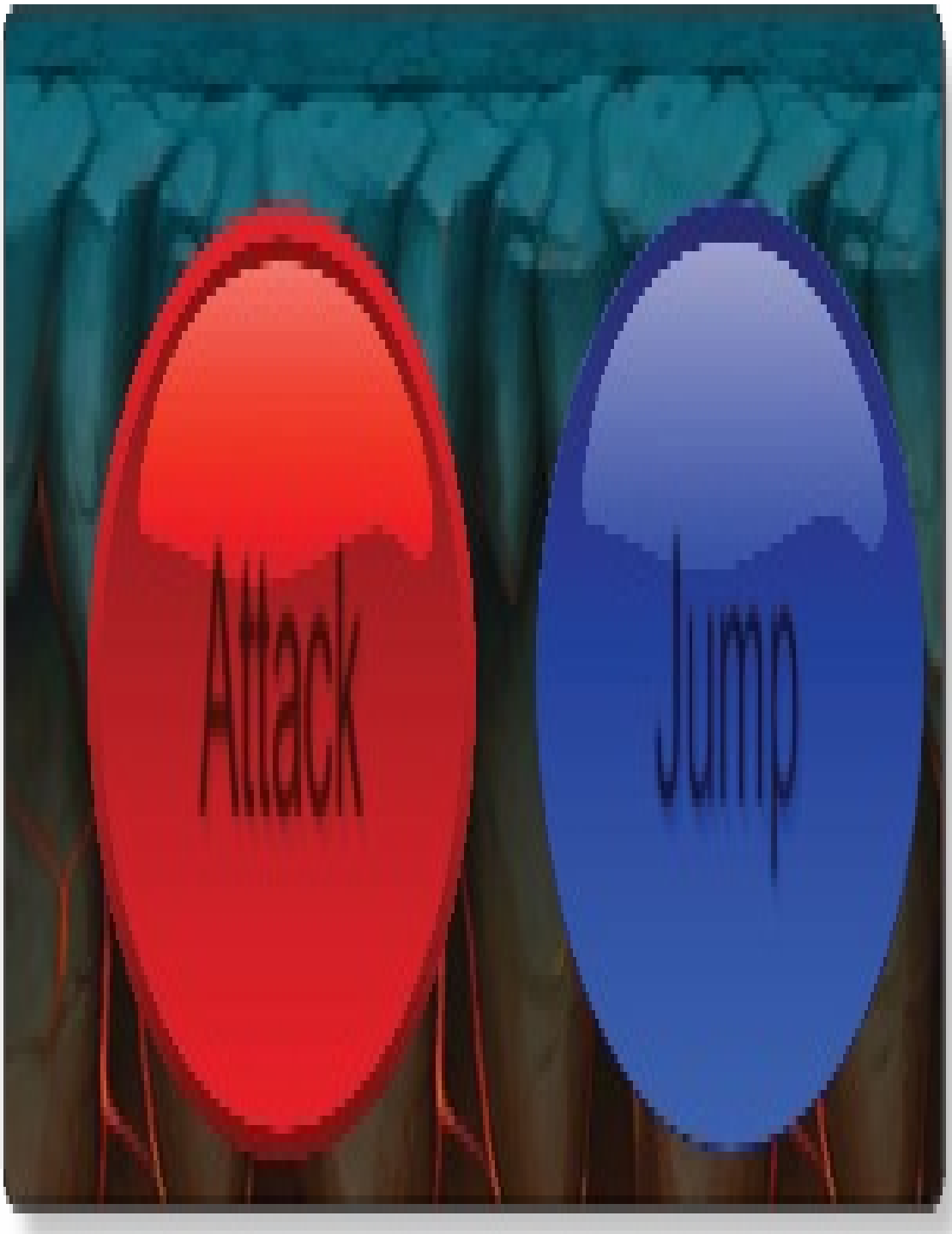
Para aplicar un estado gráfico en el botón cuando se oprima:

En “Hierarchy” seleccione “JumpButton”, asigne clic derecho para crear un nuevo elemento “UI / Image” y renómbrelo “BlueButton”. Coloquelo en su jerarquía arriba del elemento “Text” como se muestra.

MobileSingleStickControl
BackgroundMove
MobileJoystick
JumpButton
BlueButton
Text
AttackButton

Reacomodo en jerarquía

Con “BlueButton” seleccionado, reemplace su grafico de botón rojo por el grafico del botón color azul en “Inspector” y asegúrese de que tenga las mismas dimensiones de transformación que la imagen de “JumpButton” como se muestra.


Attack
Jump

Previsualización de estado de botón “Jump”

Seleccione el objeto “JumpButton”, en Inspector vaya a su componente “EventTrigger / Pointer Down”, clic en el botón (+) para asignar un nuevo comportamiento, asigne arrastrando desde “Hierarchy” el objeto “BlueButton” y asigne los parámetros que se muestran.

Event Trigger (Script)
Pointer Down (BaseEventData)
Editor And Runtir
ButtonHandler.SetDownState
JumpButton
Runtime Only
GameObject.SetActive
BlueButtor

Evento de botón “PointerDown”

El evento “PointerDown” se refiere a cuando el botón es pulsado, y en el apartado “RunTimeOnly” debemos de establecer un objeto de juego activo, el cual será en este caso el botón azul.

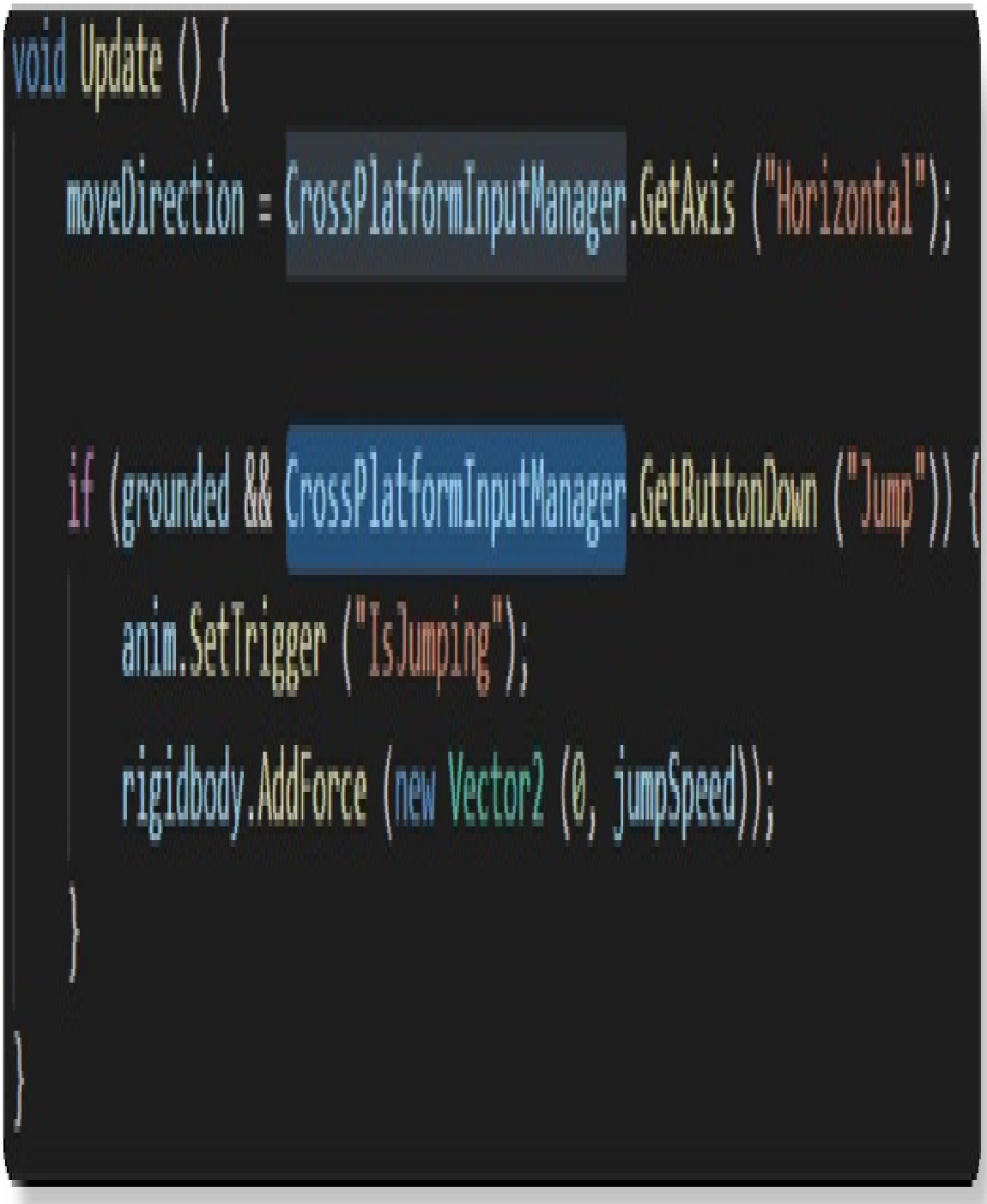
Repita el proceso para el evento “Pointer Up” y activando su casilla como se muestra.

Pointer Up (BaseEventData)
Editor And Runtir
ButtonHandler.SetUpState
JumpButton
Runtime Only
GameObject.SetActive
BlueButtor

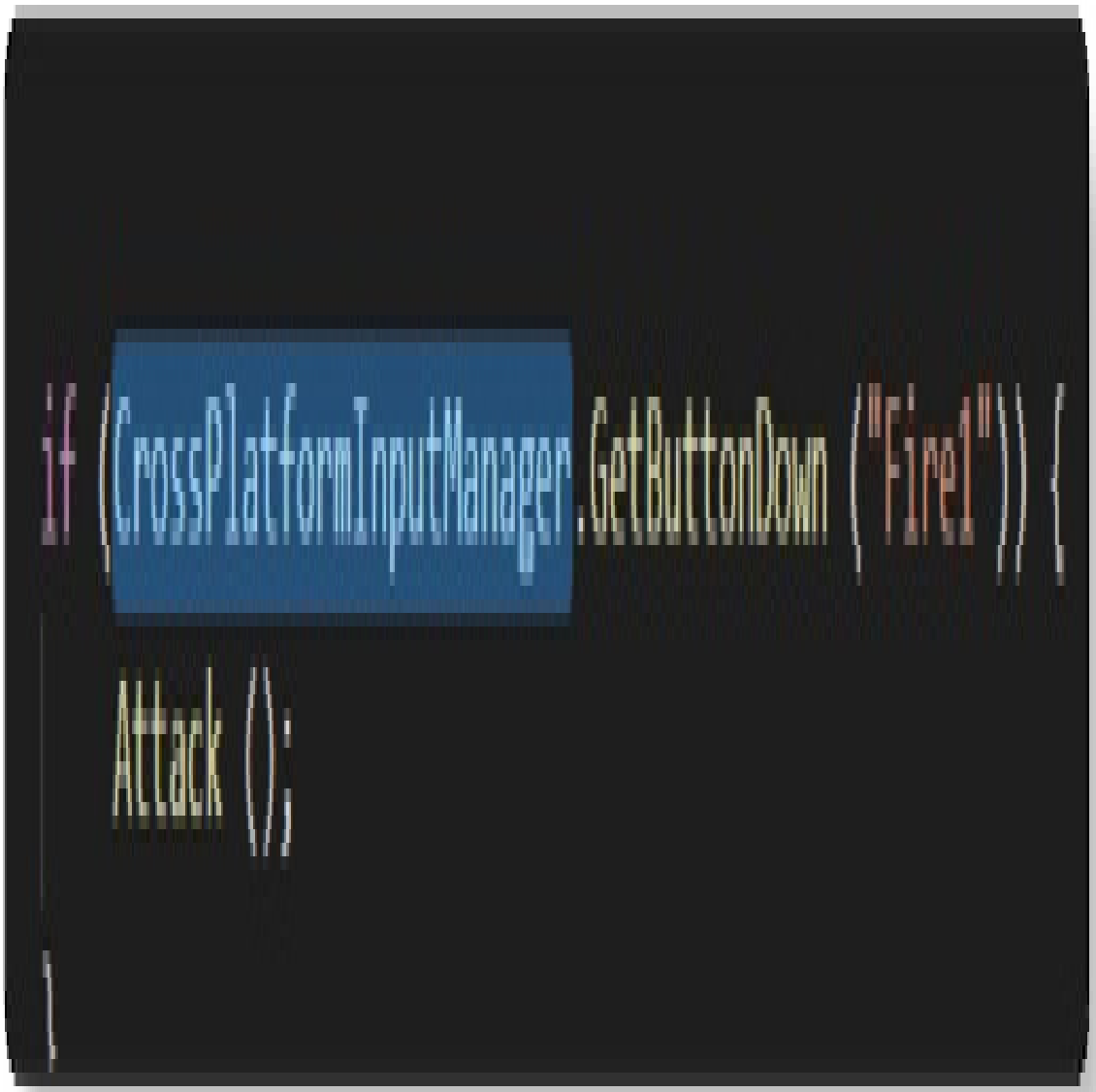
Evento de botón “PointerUp”

El evento “PointerUp” se refiere a cuando el botón no se encuentra presionado.

Vaya al script “CharacterMovement” y edite el código que se muestra en la función “Update” para asignar la programación en los botones virtuales para el ataque y para el salto.


void Update () {
moveDirection = CrossPlatformInputManager.GetAxis ("Horizontal");
if (grounded && CrossPlatformInputManager.GetButtonDown ("Jump")) {
anim.SetTrigger ("IsJumping");
rigidbody.AddForce (new Vector2 (0, jumpSpeed));
}
}

Edición de código para control de salto

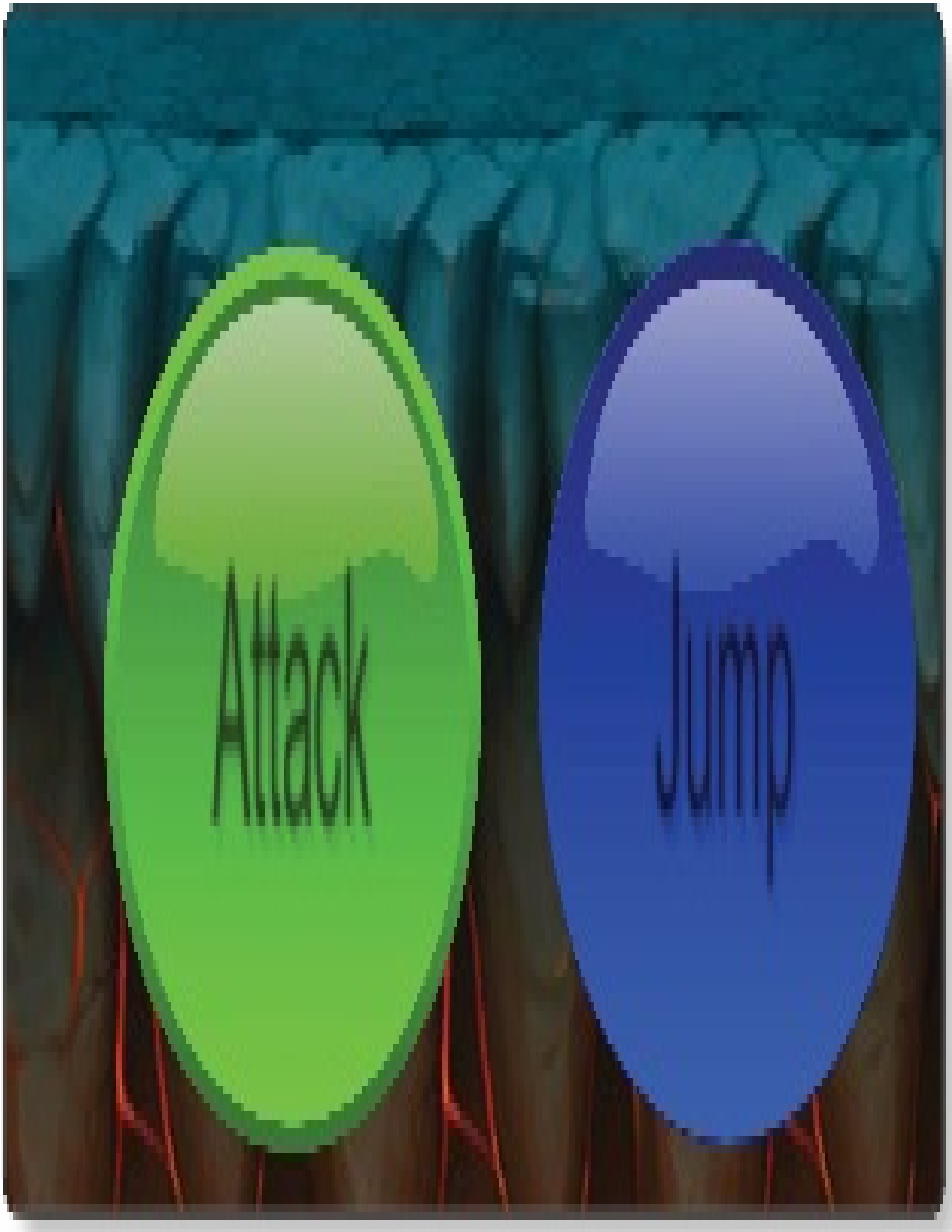

if (CrossPlatformInputManager.GetButtonDown ("Fire1")) {
Attack ();
}

Edición de código para control de ataque

Previsualice el juego para comprobar:

La funcionalidad de los botones virtuales.

El comportamiento de los botones cuando son oprimidos y cuando no lo están.


Attack
Jump

Previsualización de estado de los dos botones

Implementación de Adds

En este apartado programaremos nuestro proyecto para entrar en el programa de Adds o anuncios.

El sistema para Adds es un sistema preestablecido en el cual podemos monetizar nuestros juegos a partir de programar que en un momento específico en nuestro juego podamos implementar dichos Adds.

Para poderlos previsualizar con Adds reales debemos de tener nuestra cuenta de desarrolladores y subir nuestro juego, como en este punto todavía no realizamos esto, vamos a crear una previsualización de Adds simulada que nos permite Unity y poder visualizar en que parte se van a mostrar los Adds una vez que nuestro juego se encuentre en la tienda virtual.

Primeramente, debemos de crear un proyecto en la página de Unity referente a la monetización con Ads y es lo que haremos a continuación.

En el explorador web vaya al siguiente link.

https://unity3d.com/es/unity/features/ads

Haga “Sign In” con su Unity ID.

Clic en el botón “Comienza a monetizar ahora”.

Unity Ads
Recibe un pago por todo el arduo trabajo que realizas
Unity Ads permite a los editores integrar anuncios publicitarios en video a tus juegos móviles de una forma nueva, que no solo aumenta la participación de los jugadores, sino que además te permite ganar más dinero durante el ciclo de vida del gamer. Unity Ads ofrece el ingreso promedio por usuario (ARPU) más alto de todas las redes globales de anuncios publicitarios en video recompensados. Para los anunciantes, lleva a su propia puerta a un público de calidad.
Comienza a monetizar ahora
Comienza a anunciar ahora

Website de Unity Ads

NOTA:

La página se encuentra actualizada al momento de escribir este libro, puede cambiar dependiendo las actualizaciones realizadas al momento de realizar el ejercicio.

En la pantalla que se abre haga clic en el botón “New Project”.

No Ads-enabled Projects

Your organization currently has no Ads-enabled projects! Set up a new project by clicking the button below.

NEW PROJECT

Creación de un nuevo proyecto para monetizar

Asigne el nombre del proyecto y clic en el botón “Add Project”.

Add new project

Project name

GhoulimanAddTest01

Store IDs (optional)

The store IDs are used to fetch store information such as app category and icon. Our ad optimization will use this information to maximize your revenue. If the game hasn't been published yet you can always add the IDs later.

Apple App Store ID

Google Play Store ID

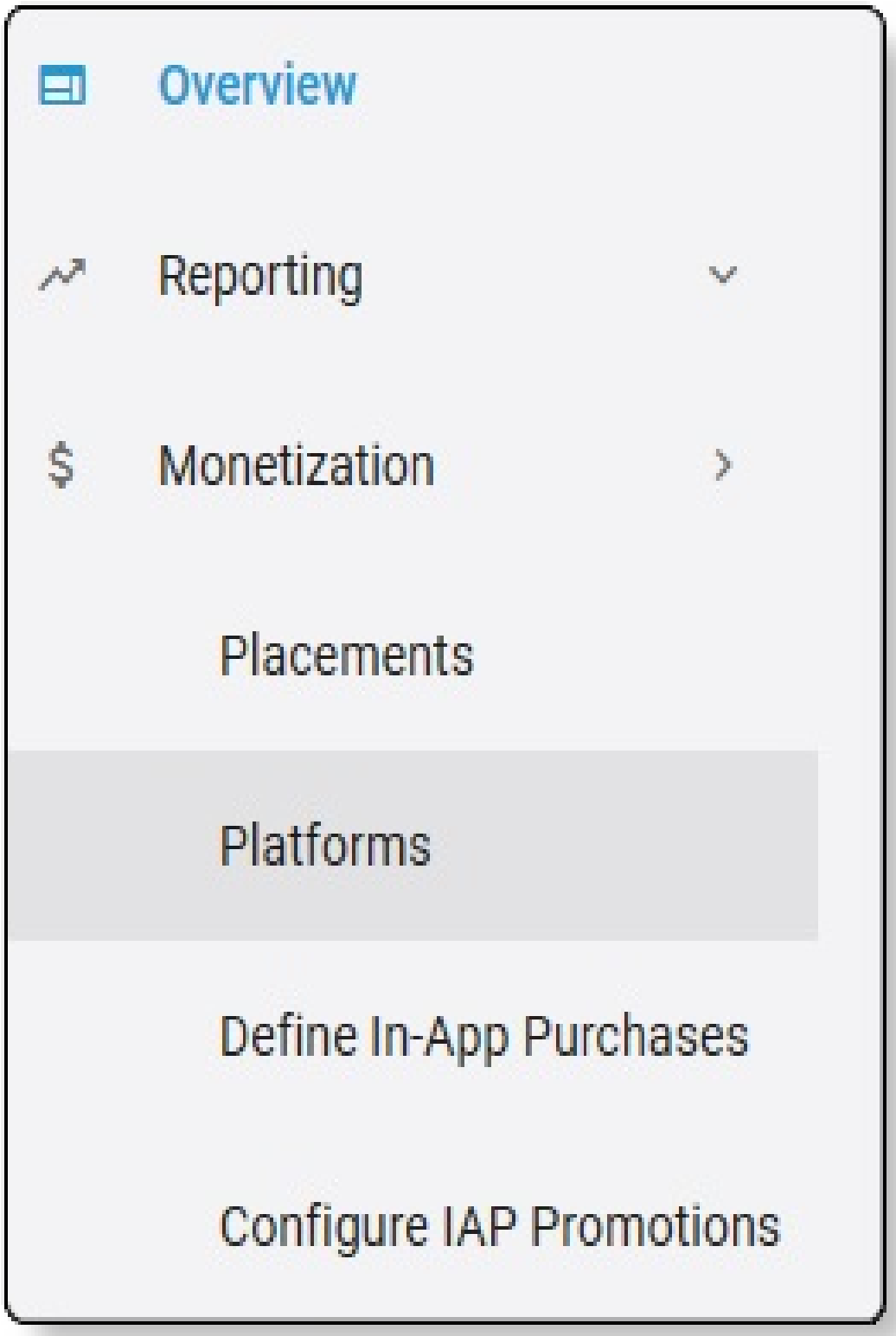
☐ This game is directed to children under the age of 13 in the United States

In accordance with the Children's Online Privacy Protection Act (COPPA), we require all products that use Unity Ads to identify whether or not they are directed at children under the age of 13 in the United States

CANCEL ADD PROJECT

Valores para un nuevo proyecto de Ads

Clic en la pestaña “Platforms” en el menú izquierdo para observar las “Game ID” para la plataforma “Apple App Store” y “Google Play Store”.


Overview
Reporting
Monetization
Placements
Platforms
Define In-App Purchases
Configure IAP Promotions

Pestaña “Platforms”

PLATFORM	GAME ID	STORE GAME ID	STORE GAME NAME
Apple App Store	2672268	ADD STORE DETAILS	-
Google Play Store	2672269	ADD STORE DETAILS	-

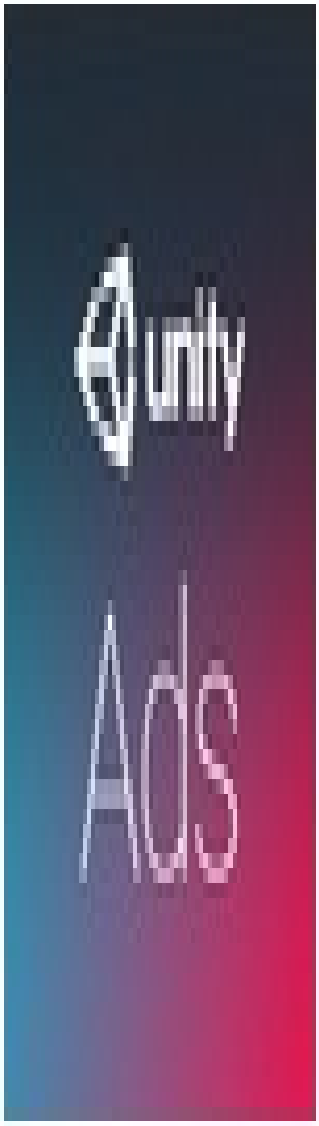
ID para ambas plataformas

NOTA:

Para cada plataforma de venta se genera un numero de ID diferente, como en este ejercicio lo estamos creando para Android, tomaremos en cuenta el número referente a GooglePlay.

Tome nota del número de ID para GooglePlay.

Vaya a Unity y busque en “AssetStore” el paquete “Unity Ads”.



Unity Ads

Services/In-Game Ad...

Unity Technologies

★★★★ (202)

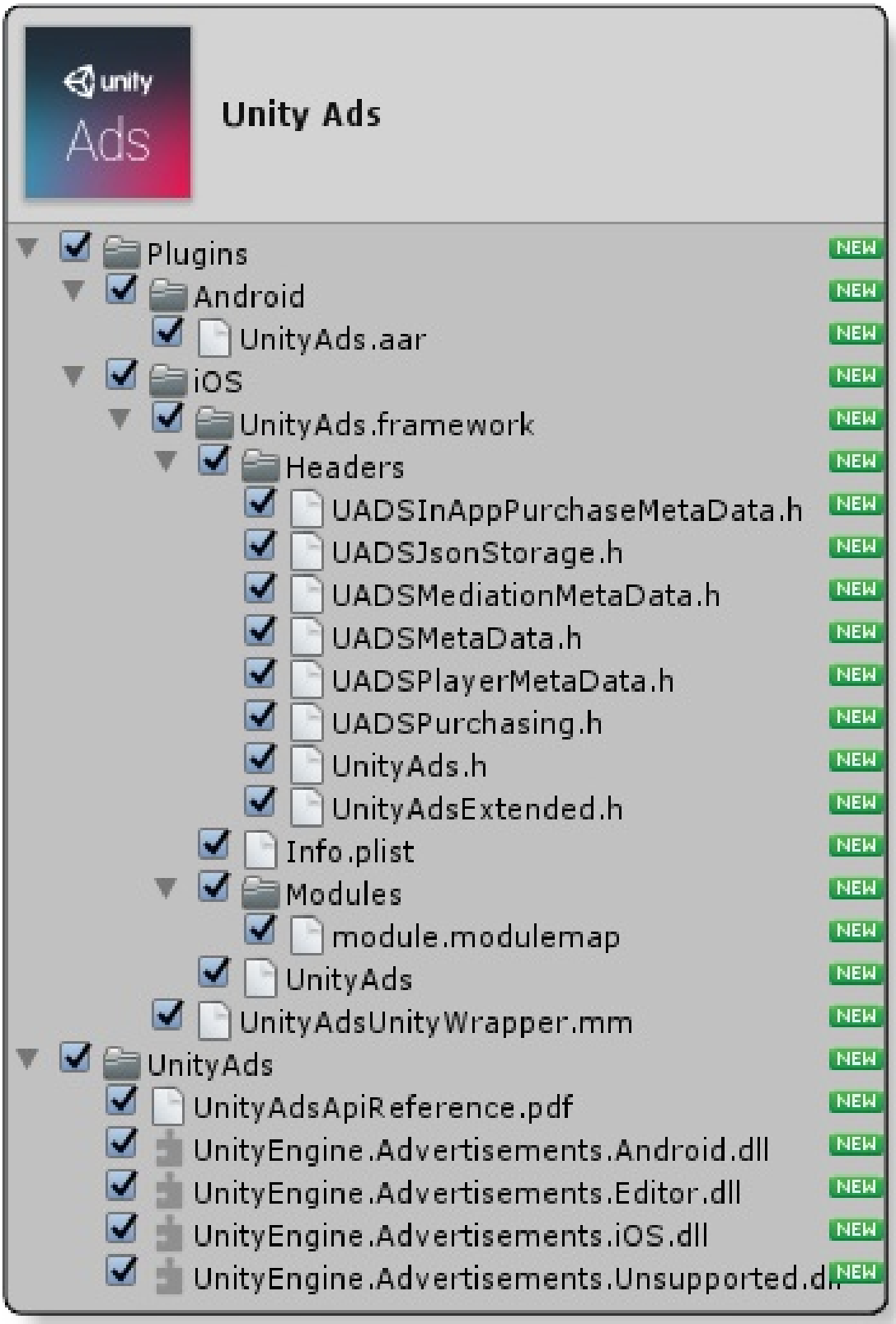
FREE

Miniatura de UnityAds en Asset Store

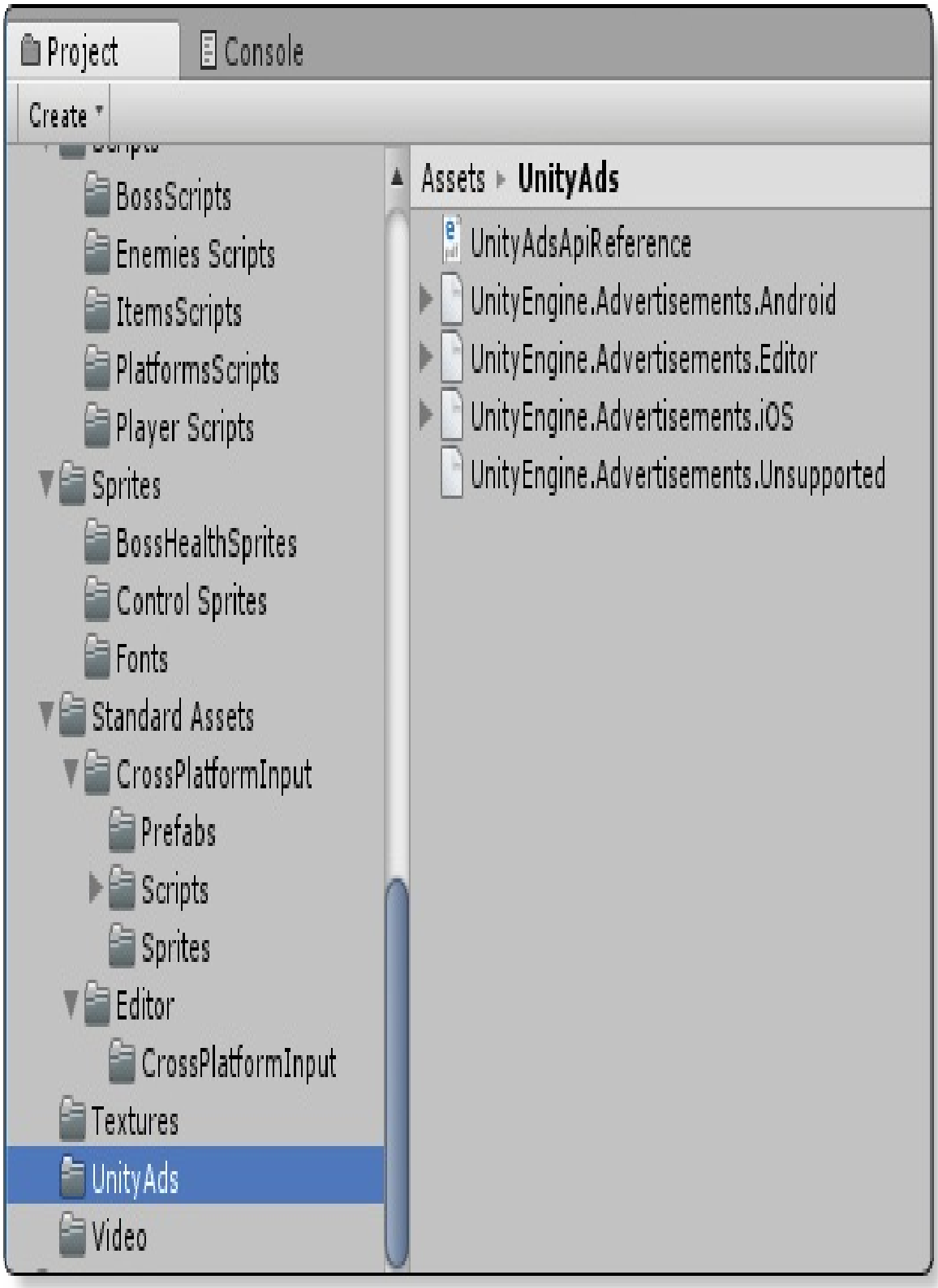
Scene
Asset Store
Language:English
Filters
Unity Ads
Services/In-Game Advertising
Unity Technologies
★★★★ (202)
FREE
Download
Now with IAP Promo open beta, our new version of Unity Ads (2.2) has several new features that will help drive more revenue for you, so you can keep doing what you love doing best - making mobile games.
With this new version brings:
• IAP Promo (open beta): A free service that allows you to easily surface & promote in-app
unity
WATCH A VIDEO TO UNLOCK A NEW CHARACTER!
WATCH NOW
IAP Promo open beta
VPAID & display interstitials
Improved webview

Unity Ads en Asset Store

Descargue todos los elementos del paquete como se muestra.


Unity Ads
Plugins
Android
UnityAds.aar
iOS
UnityAds.framework
Headers
UADSInAppPurchaseMetaData.h
UADSJsonStorage.h
UADSMediationMetaData.h
UADSMetaData.h
UADSPlayerMetaData.h
UADSPurchasing.h
UnityAds.h
UnityAdsExtended.h
Info.plist
Modules
module.modulemap
UnityAds
UnityAdsUnityWrapper.mm
UnityAds
UnityAdsApiReference.pdf
UnityEngine.Advertisements.Android.dll
UnityEngine.Advertisements.Editor.dll
UnityEngine.Advertisements.iOS.dll
UnityEngine.Advertisements.Unsupported.dll
NEW

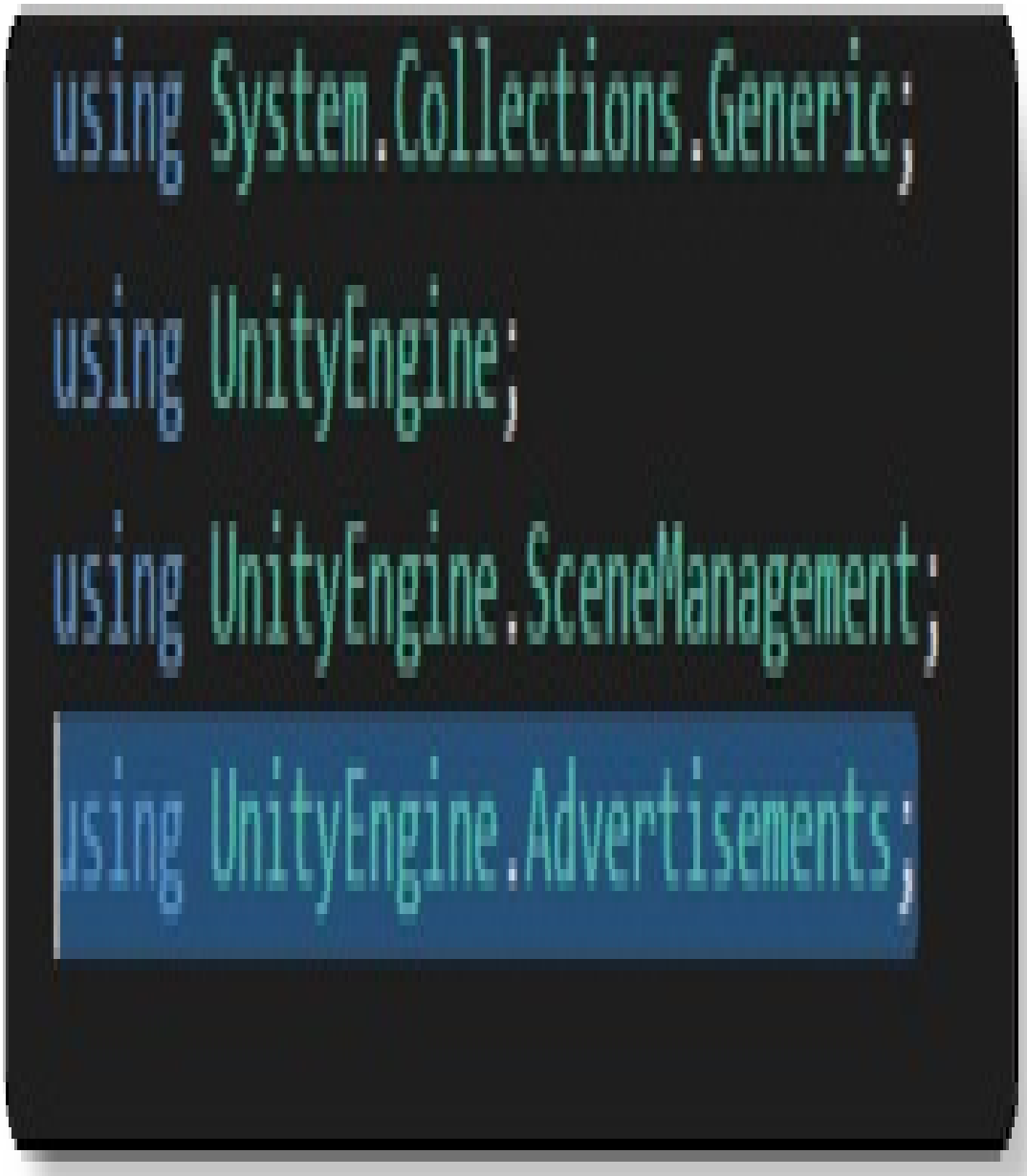
Descarga de paquetería “UnityAds”


Project
Console
Create
BossScripts
Enemies Scripts
ItemsScripts
PlatformsScripts
Player Scripts
Sprites
BossHealthSprites
Control Sprites
Fonts
Standard Assets
CrossPlatformInput
Prefabs
Scripts
Sprites
Editor
CrossPlatformInput
Textures
UnityAds
Video
Assets ▸ UnityAds
UnityAdsApiReference
UnityEngine.Advertisements.Android
UnityEngine.Advertisements.Editor
UnityEngine.Advertisements.iOS
UnityEngine.Advertisements.Unsupported

UnityAds agregado al proyecto

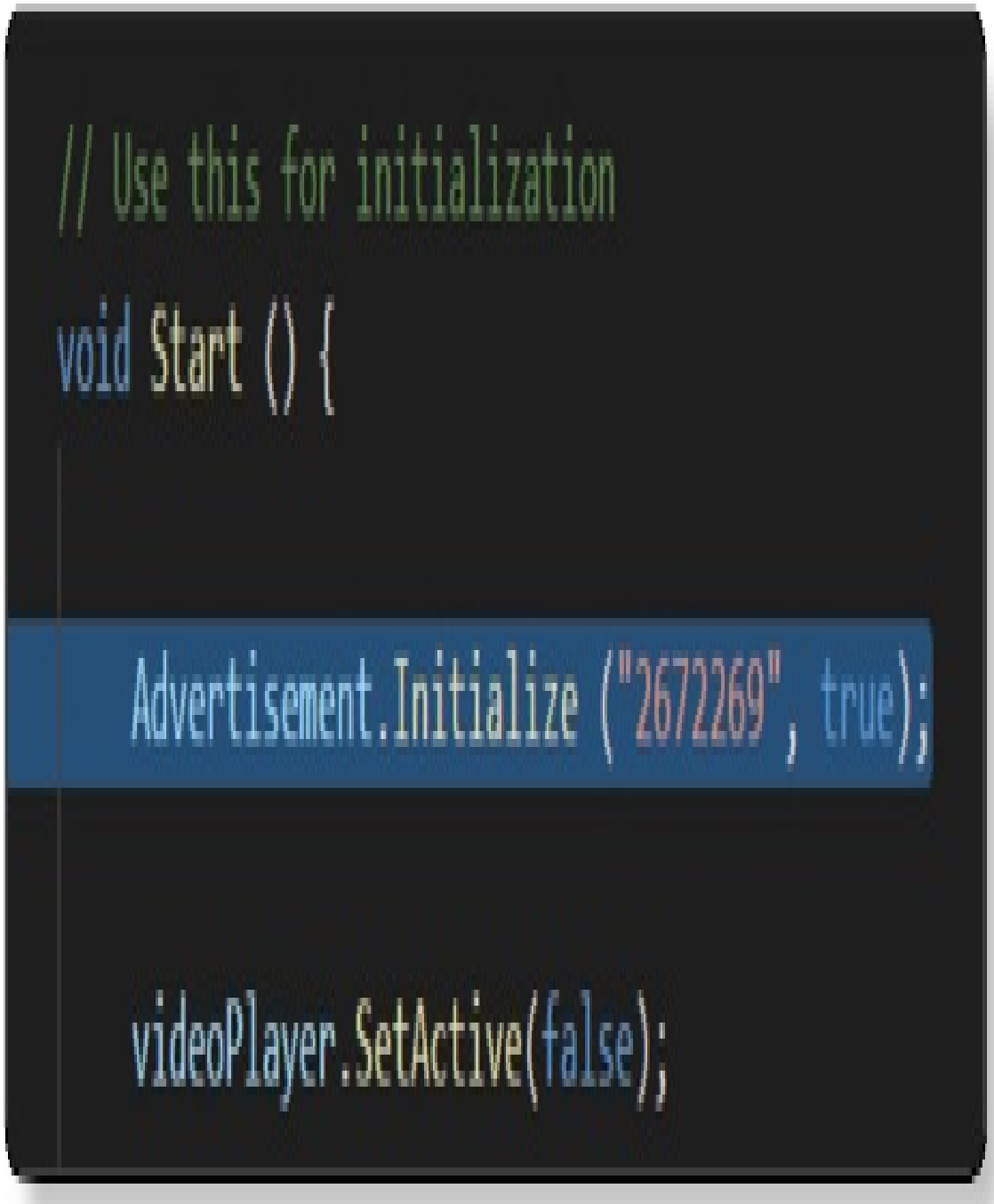
A continuación, queremos colocar el “Add” después de que corra el video de victoria.

Vaya al script “BossHealth” y mande llamar la librería que se muestra.


using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.Advertisements;

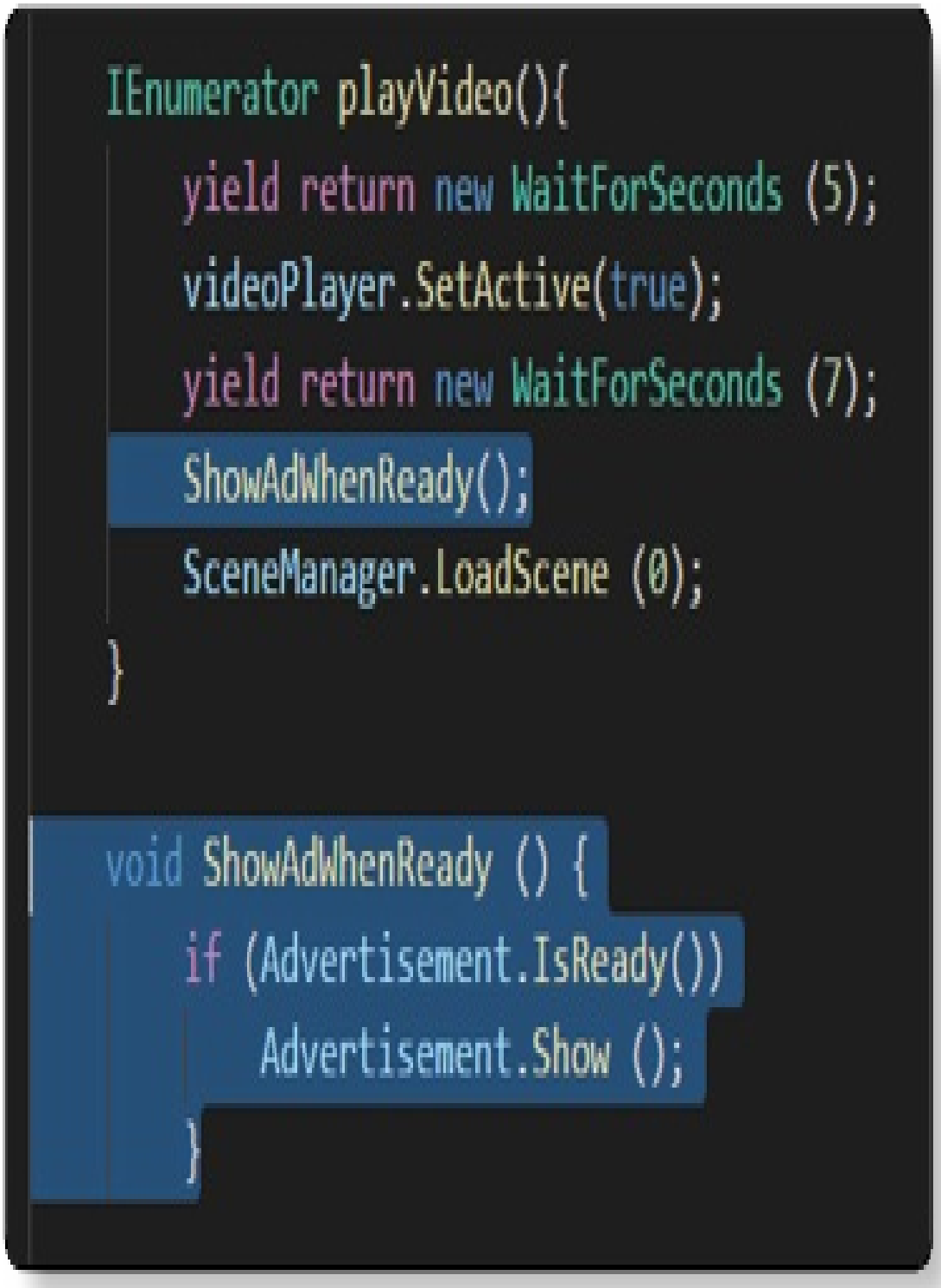
Llamar librería

Vaya a la función “Start” e inicialice el “Advertisement” como se muestra.


// Use this for initialization
void Start () {
Advertisement.Initialize ("2672269", true);
videoPlayer.SetActive(false);

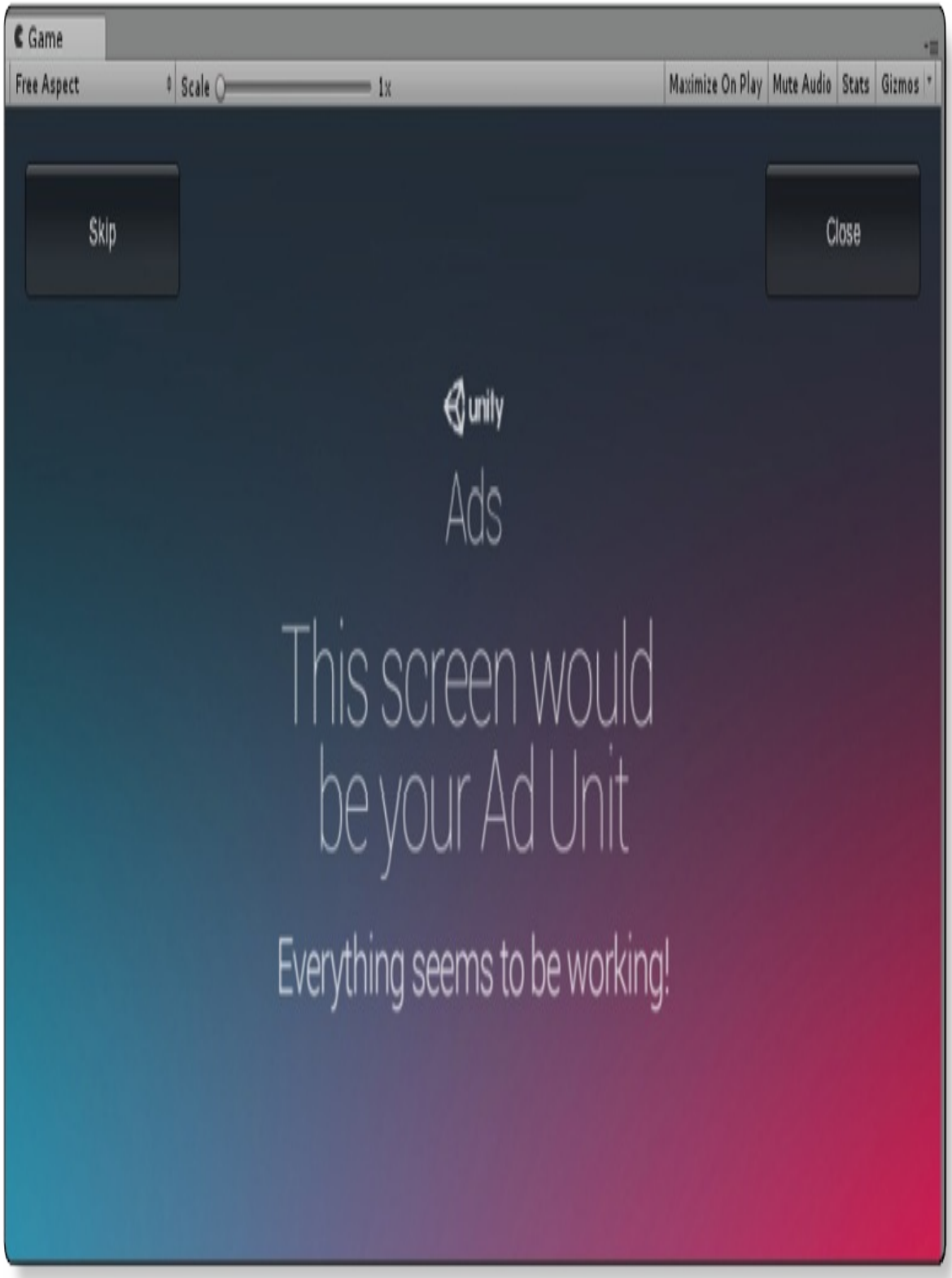
Inicialización de Advertisement

Haga la siguiente función para mostrar el Add y mándela llamar en la parte del código donde finaliza el video.


IEnumerator playVideo(){
yield return new WaitForSeconds (5);
videoPlayer.SetActive(true);
yield return new WaitForSeconds (7);
ShowAdWhenReady();
SceneManager.LoadScene (0);
}
void ShowAdWhenReady () {
if (Advertisement.IsReady())
Advertisement.Show ();
}

Creación de función

Previsualice el juego para observar que la simulación del Ad corre al finalizar el video.


Game
Free Aspect
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
Skip
Close
unity
Ads
This screen would be your Ad Unit
Everything seems to be working!

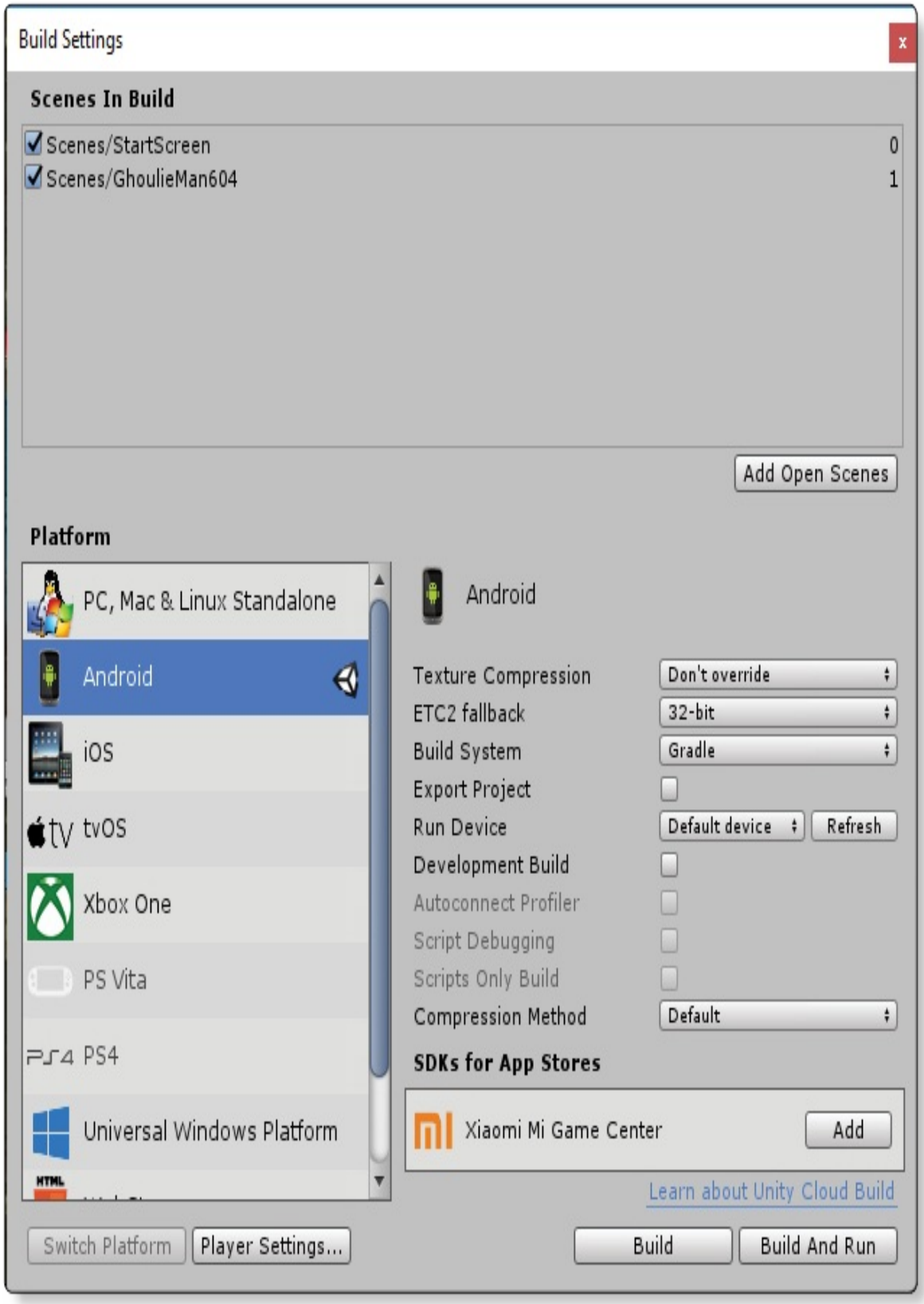
Previsualización de simulación de Ads

Exportación del juego para plataforma Android

En este apartado generaremos el archivo APK, es decir, el archivo instalador de nuestro juego en la plataforma Android, y mismo que podrá ser subido en la tienda “GooglePlay” para monetizar nuestro proyecto.

Vaya a Unity y abra “BuildSettings”.

Seleccione la plataforma “Android” y clic en “Switch Platform” cambiar la plataforma .


Build Settings
Scenes In Build
Scenes/StartScreen
0
Scenes/GhoulieMan604
1
Add Open Scenes
Platform
PC, Mac & Linux Standalone
Android
iOS
tvOS
Xbox One
PS Vita
PS4
Universal Windows Platform
Android
Texture Compression
Don't override
ETC2 fallback
32-bit
Build System
Gradle
Export Project
Run Device
Default device
Refresh
Development Build
Autoconnect Profiler
Script Debugging
Scripts Only Build
Compression Method
Default
SDKs for App Stores
Xiaomi Mi Game Center
Add
Learn about Unity Cloud Build
Switch Platform
Player Settings...
Build
Build And Run

Clic en el botón “Player Settings” y en “Inspector asigne las identidades del proyecto de manera personal como se muestra.

Inspector
Services
PlayerSettings
Company Name
Cineanimacion
Product Name
GhoulieMan
Default Icon
Select
Default Cursor
None
(Texture
2D)
Select
Cursor Hotspot
X 0
Y 0

NOTA:

En el campo de “Company Name” deberá de asignar un nombre personal.

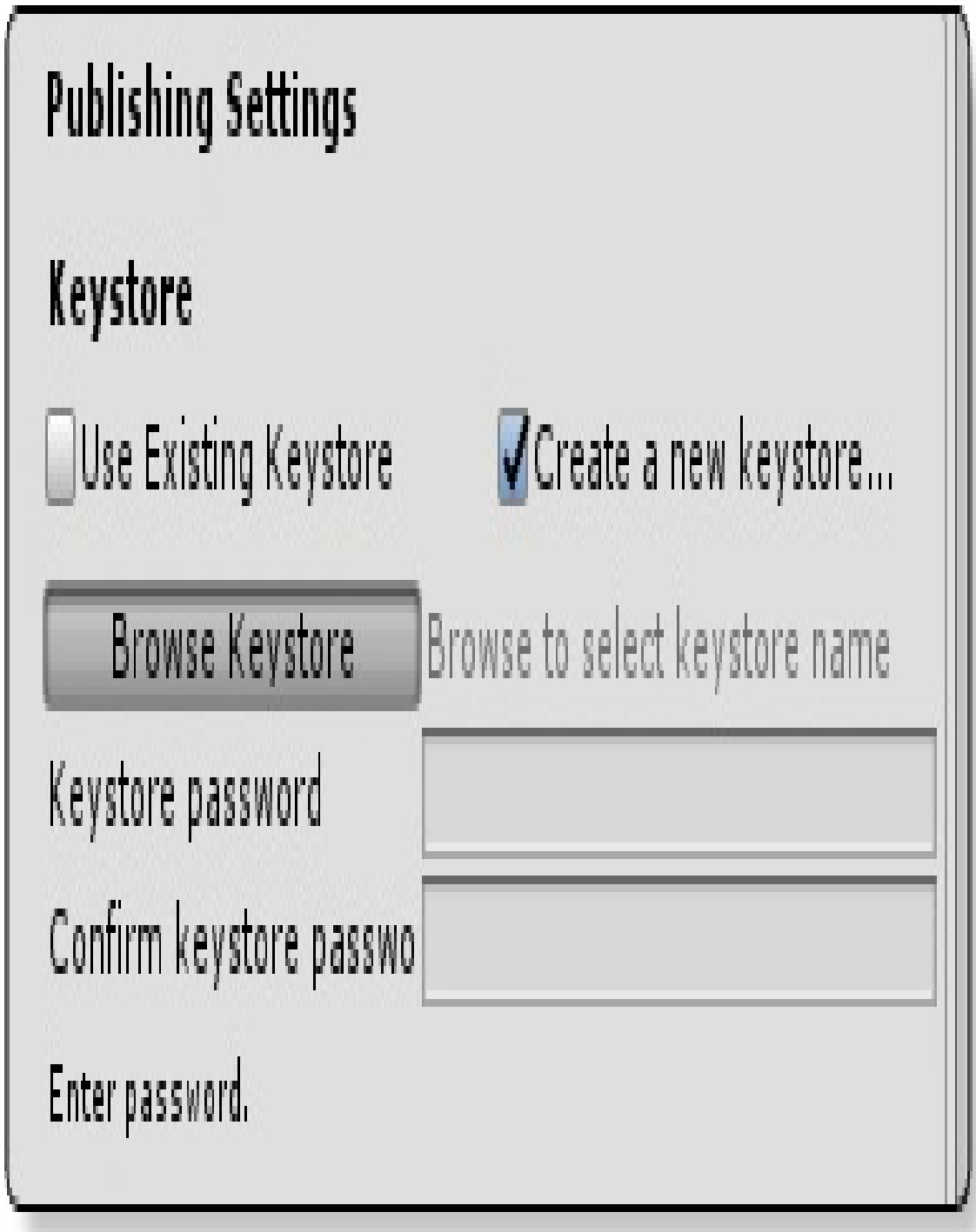
El “Default Icon” es el ícono que va a tener nuestra aplicación y en este caso fue utilizado el gráfico correspondiente a las vidas del Player.

Vaya al apartado de Identification y asigne el “Package Name” con la nomenclatura personal “com.YouCompanyName.YourGameName” y asigne los valores conforme a la mínima versión de Android en la que pueda correr su juego como se muestra.

Identification
Package Name
com.Cineanimacion.Ghoulieman
Version*
1.0
Bundle Version Code
1
Minimum API Level
Android 4.1 'Jelly Bean' (API level 16)
Target API Level
Automatic (highest installed)

A continuación, queremos crear un “KeyStore” o clave personal para que únicamente nosotros podamos subir actualizaciones de nuestro juego.

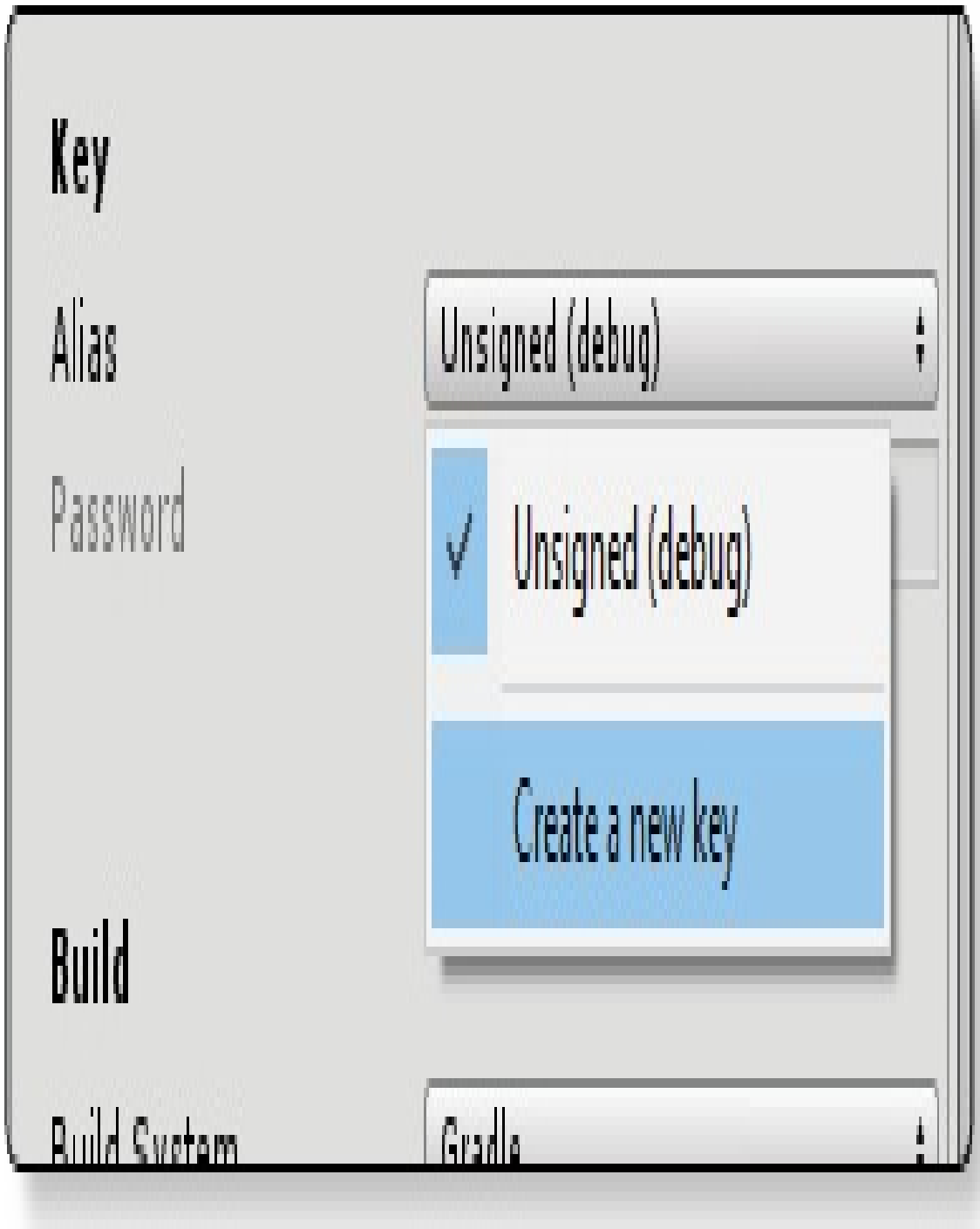
Vaya a “Publishing Settings” y marque la casilla “Create a new keystore” y clic en el botón “Browse KeyStore”.


Publishing Settings
Keystore
Use Existing Keystore
Create a new keystore...
Browse Keystore
Browse to select keystore name
Keystore password
Confirm keystore passwo
Enter password.

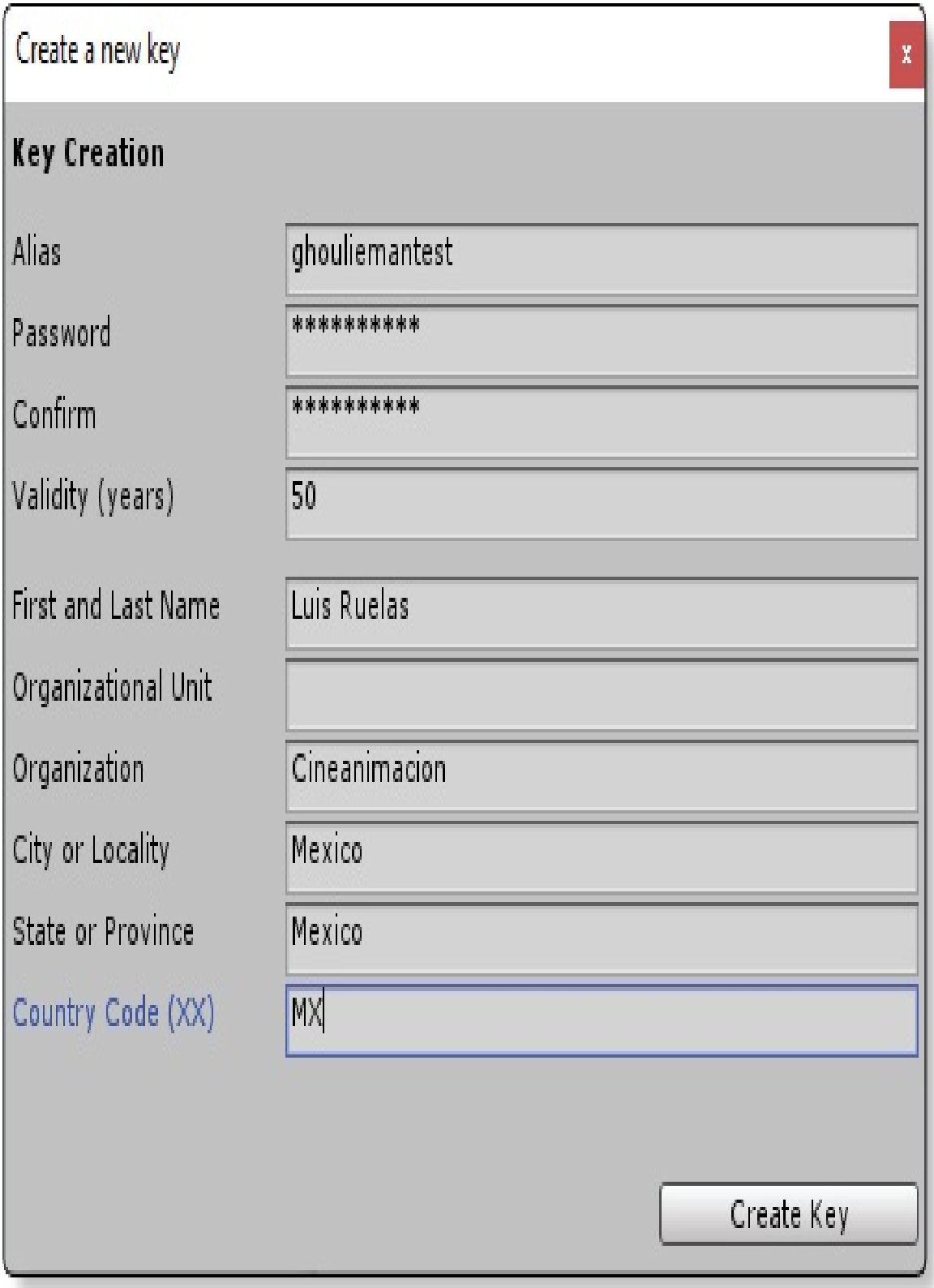
Elija la ruta en su ordenador donde vaya a guardar su “KeyStore” y nómbrelo “userKeyStore”

Haga un “password” personal y confírmelo.

En el apartado “Key” asigne en “Alias” la opción “Create New Key”.


Key
Alias
Unsigned (debug)
Password
Unsigned (debug)
Create a new key
Build
Gradle

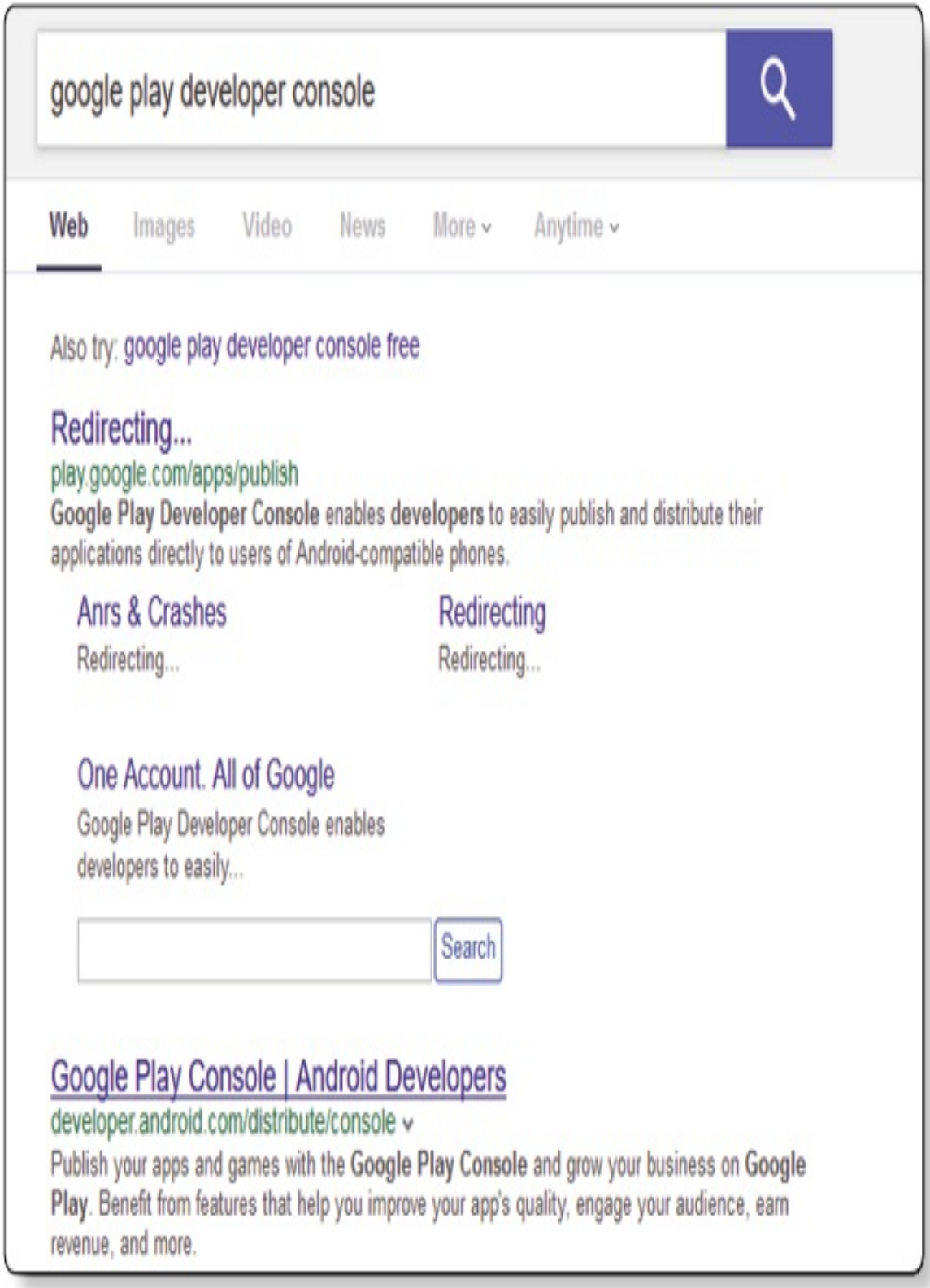
En la ventana que se abre, ingrese sus datos correspondientes incluyendo el password creado.


Create a new key
Key Creation
Alias
ghouliemantest
Password
****
Confirm
****
Validity (years)
50
First and Last Name
Luis Ruelas
Organizational Unit
Organization
Cineanimacion
City or Locality
Mexico
State or Province
Mexico
Country Code (XX)
MX
Create Key

Vaya a Build Settings y clic en “Build”, asigne un nombre y una ruta para crear su archivo .apk.

Instale el. apk en su dispositivo móvil para comprobar que su juego corre satisfactoriamente.

Para subir su juego a “GooglePlay” deberá de contar con una licencia de desarrollador, misma que puede adquirir en el sitio que se muestra.


google play developer console
Web
Images
Video
News
More
Anytime
Also try: google play developer console free
Redirecting...
play.google.com/apps/publish
Google Play Developer Console enables developers to easily publish and distribute their applications directly to users of Android-compatible phones.
Anrs & Crashes
Redirecting...
Redirecting
Redirecting...
One Account. All of Google
Google Play Developer Console enables developers to easily...
Search
Google Play Console | Android Developers
developer.android.com/distribute/console
Publish your apps and games with the Google Play Console and grow your business on Google Play. Benefit from features that help you improve your app's quality, engage your audience, earn revenue, and more.

https://developer.android.com/distribute/console/
Developers
Platform
Android Studio
Google Play
Android Jetpack
Docs
Blog
Buscar
Google Play
LAUNCH PLAY CONSOLE
OVERVIEW
PLAY CONSOLE
PLAY STORE
PLAY BILLING
PLAY SERVICES
GUIDES
RESOURCES
STORIES
Google Play Console
Publish your apps and games with the Google Play Console and grow your business on Google Play. Benefit from features that help you improve your app's quality, engage your audience, earn revenue, and more.
SIGN IN
NEW FEATURES
Do more with the Play Console

material adicional

El material adicional de este libro puede descargarlo en nuestro portal web: http://www.ra-ma.es.

Debe dirigirse a la ficha correspondiente a esta obra, dentro de la ficha encontrará el enlace para poder realizar la descarga. Dicha descarga consiste en un fichero ZIP con una contraseña de este tipo: XXX-XX-XXXX-XXX-X la cual se corresponde con el ISBN de este libro.

Podrá localizar el número de ISBN en la página IV (página de créditos). Para su correcta descompresión deberá introducir los dígitos y los guiones.

Cuando descomprima el fichero obtendrá los archivos que complementan al libro para que pueda continuar con su aprendizaje.

INFORMACIÓN ADICIONAL Y GARANTÍA

RA-MA EDITORIAL garantiza que estos contenidos han sido sometidos a un riguroso control de calidad.

Los archivos están libres de virus, para comprobarlo se han utilizado las últimas

versiones de los antivirus líderes en el mercado.

RA-MA EDITORIAL no se hace responsable de cualquier pérdida, daño o costes provocados por el uso incorrecto del contenido descargable.

Este material es gratuito y se distribuye como contenido complementario al libro que ha adquirido, por lo que queda terminantemente prohibida su venta o distribución.