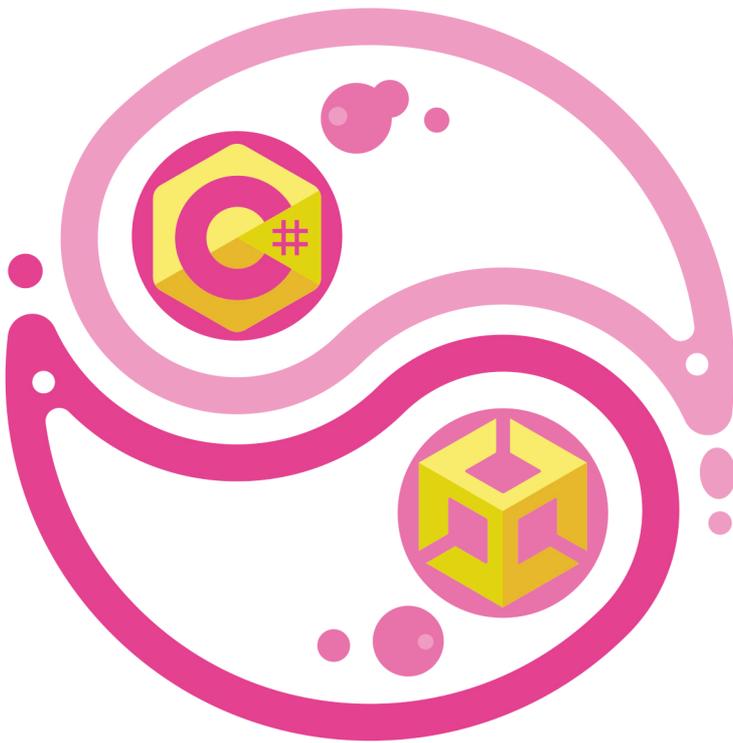


C# PARA UNITY

100 EJERCICIOS PRÁCTICOS

Cómo mejorar tu kungfú



Jorge García Colmenar

Informática

C# PARA UNITY

100 EJERCICIOS PRÁCTICOS

Cómo mejorar tu kungfú

Jorge García Colmenar

 **FC
EDITORIAL**

edü[®]
Conocimiento a su alcance
BOGOTÁ - MÉXICO, D.F.

García Colmenar, Jorge

C# para Unity. 100 ejercicios prácticos / Jorge García Colmenar -- Bogotá
: Ediciones de la U, 2024

392 p. ; 24 cm.

ISBN 978-958-792-715-3 e-ISBN 978-958-792-716-0

1. Informática 2. Lenguaje de programación I. Tít.

621.39 ed. Thema: UFC

Edición original publicada por © Fundación Confemetal (España)
Edición autorizada a Ediciones de la U de Colombia

Área: Informática

Primera edición: Bogotá, Colombia, junio de 2024

ISBN. 978-958-792-715-3

© Jorge García Colmenar

© Fundación Confemetal Editorial. Príncipe de Vergara, 74 - 28006, Madrid
Tel. (91) 7823630 - Fax (91) 5631741

www.fundacionconfemetal.com / E-mail: editorial@fundacionconfemetal.es
Madrid, España

© Ediciones de la U - Carrera 27 #27-43 - Tel. (+57) 601 6455049

www.edicionesdelau.com - E-mail: editor@edicionesdelau.com
Bogotá, Colombia

Ediciones de la U es una empresa editorial que, con una visión moderna y estratégica de las tecnologías, desarrolla, promueve, distribuye y comercializa contenidos, herramientas de formación, libros técnicos y profesionales, e-books, e-learning o aprendizaje en línea, realizados por autores con amplia experiencia en las diferentes áreas profesionales e investigativas, para brindar a nuestros usuarios soluciones útiles y prácticas que contribuyan al dominio de sus campos de trabajo y a su mejor desempeño en un mundo global, cambiante y cada vez más competitivo.

Coordinación editorial: Adriana Gutiérrez M.

Carátula: Ediciones de la U

Impresión: DGP Editores SAS

Calle 63 #70D-34, Pbx. (+57) 601 7217756

Impreso y hecho en Colombia

Printed and made in Colombia

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro y otros medios, sin el permiso previo y por escrito de los titulares del Copyright.

ÍNDICE

Sobre el autor	17
Introducción	19
Cómo no usar este libro	20
¿Cómo usar este libro?.....	20
Apunte para profesores	21

PRIMERA PARTE

Cien ejercicios básicos de programación	25
1. Crea una nueva variable llamada <code>velocidad</code> y asígnale el valor de <code>5.0F</code> . Imprime su valor por consola.....	26
2. Utiliza dos variables públicas de tipo <code>float</code> para mostrar por consola sus valores sumados, restados, multiplicados y divididos.....	27
3. Crea dos variables privadas de tipo <code>string</code> , concaténalas y muestra el resultado por consola.....	29
4. Crea una variable privada booleana y dos variables públicas <code>int</code> . Si al ejecutar el juego la suma de las variables <code>int</code> es par, la booleana debe ser <code>true</code> . Muestra su valor por consola.....	30
5. Crea un <i>script</i> que imprima por consola «Hoy es jueves» si es jueves, y «Hoy no es jueves» en caso contrario.....	33
6. Calcula el factorial de un número dado por una variable pública de tipo <code>int</code>	37
7. Crea un <i>script</i> que demuestre el orden de ejecución de los métodos <code>Start</code> y <code>Awake</code> de <code>MonoBehaviour</code>	40

8. Crea un <i>script</i> que imprima por consola, en cada fotograma, el tiempo que ha transcurrido desde el fotograma anterior.....	43
9. Crea un <i>script</i> que imprima por consola el tiempo que ha pasado desde el inicio de la ejecución cada 100 fotogramas.....	46
10. Crea un <i>script</i> que inicialice una probabilidad aleatoria entre 0 y 100 y que cada 10 segundos genere un número entre 0 y 100. Si el número es mayor que la probabilidad, imprime por consola «Acierto» y si no, imprime «Fallo».	49
Transformaciones y destrucciones	54
11. Crea un cubo que se desplace en el eje X a una velocidad de 1 unidad por segundo. Al llegar a la posición 10, debe recolocarse en la posición 0.....	55
12. Haz que un cubo rote a la izquierda o a la derecha respondiendo a las pulsaciones del teclado correspondientes a los cursores ←, → o a las letras A y D.	60
13. Haz que un cubo aumente la escala si mantenemos pulsada la tecla espaciadora y la disminuya si no la pulsamos. La escala mínima debe ser 1 en cada eje.	64
14. Crea un cubo que se desplace uniformemente entre $x=0$ y $x=10$. Debe ir y volver continuamente.	68
15. Crea un simple controlador de personaje que desplace un cubo en los ejes X y Z al pulsar los cursores.	70
16. Crea un objeto que se desplace en el eje Z una unidad al soltar la tecla espaciadora.	72
17. Crea un cubo que se destruye al clicar en él.	74
18. Crea un cubo que se destruya a los tres segundos de clicar en él.	76
19. Crea un cubo que se desactive entre tres y cinco segundos después de clicar en él.	80

20. Crea un cubo que cambie de nombre para indicar cuántas veces has clicado en otro cubo.....	85
--	----

SEGUNDA PARTE

A vueltas con los bucles	91
21. Crea un bucle que sume los primeros quinientos números pares.	92
22. Crea una línea de diez cubos en el eje X usando un bucle, sin usar prefabs.....	94
23. Instanciar una pared de cubos vertical usando un prefab.	96
Variables y asignaciones	97
Bucle anidado	98
Instanciación de cubos	98
24. Instanciar un cubo hecho de cubos usando un prefab.	100
25. Instanciar un bloque de cubos como en el ejercicio 24, pero utilizar diferentes prefabs para generar capas de piedra, tierra y césped, como si fuese una sección de suelo del juego Minecraft.....	103
El operador ternario	105
26. Instancia una pirámide hecha de cubos.	107
Colecciones y bucles	110
27. Crea un cubo que se desplace uniformemente entre los puntos de un <i>array</i> o lista pública de vectores 3.....	110
Un poco más sobre diferencias entre listas y <i>arrays</i>	112

28. Crea un cubo que se desplace uniformemente entre los puntos de un <i>array</i> o lista pública de <i>Vector3</i> , pero que elija aleatoriamente el siguiente punto de la colección al que desplazarse.	114
29. Crea unos cuantos objetos con el <i>tag</i> «desaparecer» en la escena, manualmente. Al pulsar la tecla espaciadora, todos los objetos deben desactivarse. Los objetos no pueden tener ningún <i>script</i> añadido. El ejercicio ha de funcionar sin modificar el código aunque se añadan o quiten objetos de la escena.....	118
El bucle <i>foreach</i>	122
30. Instancia diez cubos en la escena en posiciones aleatorias entre -10 y 10 en cada eje. Al pulsar el espacio, deben destruirse los que tengan posiciones positivas en el eje X.	123
Key Code	125

TERCERA PARTE

A por los componentes	129
31. Crea una esfera que cambie de color cuando se clica en ella.	130
El método <i>GetComponent</i>	131
32. Crear dos esferas. Al clicar sobre una, la otra cambia de color. ...	132
La propiedad <i>material</i>	134
33. Genera un efecto de movimiento modificando el <i>offset</i> de la textura de un material de un plano o <i>quad</i>	135
La función <i>SetTextureOffset</i>	137

A por las físicas	139
34. Crear una esfera en el punto $Y=10$ y un suelo en el $Y=0$. Hacer que la esfera caiga al pulsar el espacio. La esfera tiene que rebotar como un balón de baloncesto.....	139
35. Crea una esfera en el punto $Y=10$ y un suelo en el $Y=0$. Haz que la esfera caiga al pulsar el espacio. La esfera tiene que rebotar como un balón de baloncesto.....	143
La función <code>Compare tag</code>	146
36. Crea una esfera en el punto $Y=10$ y un suelo en el $Y=0$. Haz que la esfera caiga al pulsar la tecla espaciadora. La esfera tiene que rebotar como un balón de baloncesto y destruirse al rebotar dos veces.	149
37. Crea una esfera en reposo sobre un suelo. Al pulsar la tecla S, recibe una fuerza puntual hacia arriba y hacia la derecha.	151
38. Crea una esfera en reposo sobre un suelo. Mientras la tecla S permanezca pulsada, la esfera debe recibir una fuerza constante hacia arriba y hacia la derecha.	155
39. Crea unos cuantos objetos con el <code>tag «newton»</code> . Al pulsar espacio, debe activarse la gravedad de todos esos objetos. Los objetos no pueden tener ningún <code>script</code> añadido. El ejercicio tiene que funcionar sin modificar el código aunque se añadan o quiten objetos de la escena.	158
40. Crea un <code>Raycast</code> que nos diga en qué objeto de la escena hemos hecho clic.....	161
41. Al clicar en un objeto, destruir todos los demás que estén en rango 10 desde ese objeto.	163
El caso de <code>Camera.main</code>	167
42. Crea una cápsula centinela que rota sobre sí misma. Emite un <code>Raycast</code> de distancia 5 que destruye solo a esferas si las toca.	169

43. Crea una cápsula centinela que rota sobre sí misma. Emite un <code>Raycast</code> de distancia 5 que destruye las esferas solo si las toca.	171
Más componentes: iluces, cámara, acción!	174
44. Utilizando dos luces <i>point</i> roja y azul, crea el efecto de un coche de policía apagándolas y encendiéndolas alternativamente.....	175
45. Crea un efecto de luces de navidad. El <i>script</i> debe funcionar independientemente del número de luces. Utiliza <i>point light</i> y un <i>array</i> público para enlazarlas (referenciarlas) al <i>script</i> . Basta con que cada luz alterne, no es necesario programar diferentes modos.	178
46. Crea un efecto de luces de navidad. El <i>script</i> debe funcionar independientemente del número de luces. Utiliza <i>point light</i> y un <i>array</i> público para enlazarlas (referenciarlas) al <i>script</i> . Basta con que cada luz alterne, no es necesario programar diferentes modos.	180
47. Crea un efecto «bombilla a punto de fundirse».	182
Ventajas de las corrutinas.....	186
Limitaciones de las corrutinas	186
48. Crea un semáforo de tráfico, incluyendo la luz ámbar. Como buen semáforo, debe pasar más tiempo en rojo que en verde.....	188
49. Al clicar con el botón izquierdo del ratón, cambia el color de fondo de la cámara. Al clicar con el botón derecho, el fondo se pone blanco.....	190
50. Crea un escenario simple y coloca cuatro cámaras. Cada una debe renderizarse en una cuarta parte de la pantalla, como en el Mario Kart, por ejemplo.	193

51. Crea cinco objetos en las posiciones $X=0$, $X=10$, $X=20$, etc. Pulsando los números 1 a 5, la cámara debe desplazarse para enfocar al objeto adecuado (el número 1 se corresponde con el objeto en $X=0$).	195
52. Crea un sistema de partículas que emita solo cuando las teclas Q y P estén pulsadas a la vez.	198
53. Crea un objeto con un <code>AudioSource</code> que solo reproduce sonido mientras no se ve.....	202
El componente <code>AudioSource</code> en Unity.....	205

CUARTA PARTE

Interfaces de usuario (UI)	212
54. Crea un sencillo reproductor de música.	213
55. Añade un Slider para cambiar el volumen a tu reproductor de música.	218
Crea un Slider	219
Configura el Slider	219
Asigna el Slider	221
El evento <code>onValueChanged</code>	221
56. Sobre el ejercicio anterior, el volumen debe permanecer de una ejecución a otra (usando <code>PlayerPrefs</code>).	223
57. Crea un botón de UI que al pulsarse instancie una esfera en una posición aleatoria entre -10 y 10 en cada eje.	227
58. Crea un temporizador de diez segundos con una imagen radial.	231

59. Crea una interfaz de usuario simple con un texto o una imagen centrada en la pantalla a modo de mirilla. Haz que la cámara rote en función del movimiento del ratón (como en un juego de primera persona).....	237
60. Sobre el ejercicio anterior, lanza un <code>Raycast</code> , en dirección de la mirilla al hacer clic, que instancie un sistema de partículas al colisionar.	239
Desglose del SRP	242
61. Crea un sistema de diálogo simple, estilo novela visual, con <i>arrays</i> o listas para el guion, y un panel y texto en la interfaz de usuario. Al pulsar la tecla espaciadora aparece la siguiente línea de guion.	244
Principales características y funciones del componente <code>Text</code>	246
62. Sobre el ejercicio anterior, carga el guion desde un archivo en la carpeta <code>Resources</code>	247
Carpetas especiales	250
63. Crea un contador de clics sobre un botón de UI.....	252
64. Sobre el ejercicio anterior, hacer que el contador permanezca de una ejecución a otra (sin usar la clase <code>PlayerPrefs</code>).....	255
La serialización	258
La directiva <code>Using</code>	259
65. Sobre el ejercicio anterior, crea un botón para resetear los clics.	261
La encapsulación	262

QUINTA PARTE

66. Crea un objeto que se mueva de forma senoidal en un eje. El eje y la amplitud del movimiento deben estar dados por parámetro.	267
--	-----

67. Crea una esfera que siga suavemente al objeto del ejercicio anterior.....	270
68. Crea una trampa de pinchos que aparece y desaparece del suelo.	273
69. Crea tres escenas. Cada una debe tener 2 botones que permitan ir a las otras escenas.	275
Scene Manager.....	277
70. Crea la clase <i>Glitchosaurio</i> . Los <i>glitchosaurios</i> tienen fuerza, resistencia y tamaño. Pueden moverse, comer, esconderse y <i>buguearse</i> . No es necesario que implementes el interior de los métodos.	279
71. Crea la clase <i>Cria</i> que hereda de <i>Glitchosaurio</i> . Las crías de <i>glitchosaurio</i> tienen carisma y sabor.	285
72. Los <i>glitchosaurios</i> pueden comer cualquier <i>glitchosaurio</i> de menor tamaño si su fuerza es mayor que la resistencia de la víctima. Si la víctima es una cría, la fuerza del cazador se incrementa tanto como el valor del sabor de la cría.	288
73. Genera 300 <i>glitchosaurios</i> con valores aleatorios y guárdalos en una lista.	290
Los patrones de diseño.....	293
74. Elimina todos los <i>glitchosaurios</i> de la lista cuya fuerza y resistencia sean iguales.....	294
Las expresiones lambda.....	295
Language Integrated Query (LINQ).....	296
75. Imprime por consola el nuevo número de <i>glitchosaurios</i>	298
76. Crea un <i>script</i> que genere una esfera cada 0.25 segundos y esta reciba una fuerza aleatoria hacia arriba y a la derecha.	301
Variables públicas.....	302
Aplicación de la fuerza ascendente.....	302

77. Sobre el ejercicio anterior, crea un suelo debajo del punto de instancia. Las esferas que toquen el suelo recibirán una fuerza ascendente hacia la izquierda, también aleatoria.....	304
78. Sobre el ejercicio anterior, el generador inicia «apagado» y se enciende y apaga pulsando la tecla espaciadora.	307
La función <code>CancelInvoke</code>	309
79. Sobre el ejercicio anterior, crea un texto en la UI que lleve la cuenta de las esferas generadas.	311
80. Crea un cubo. Cada vez que hagas clic en él con el botón derecho, debe recibir una fuerza hacia arriba y un torque aleatorio.....	316
81. Haz un <i>script</i> para que un objeto se vuelva rojo cuando sube y azul cuando baja, teniendo en cuenta su velocidad (puedes usar el cubo del ejercicio anterior).....	319
La propiedad <i>velocity</i>	322
82. Haz que una cámara siempre apunte a un objeto (puedes usar el cubo del ejercicio anterior).	324
83. Crea un cubo que instancia otro cubo al clicar en él, y luego se destruye. El cubo instanciado debe tener el mismo comportamiento.	326
84. Aumenta o disminuye la intensidad de una luz con la rueda del ratón.....	330
85. Crea un contador de clics y muestra el número en la UI.....	332
86. Imprime por consola la posición del puntero del ratón en la pantalla.	334
87. Haz que un cubo sea más rojo cuanto más cerca esté de la cámara.....	336
La clase <code>Color</code> en Unity.....	337

88. Crea dos escenas y permite que se pueda cambiar de una a otra. Crea un <code>AudioSource</code> en la primera escena y haz que la música siga sonando al cambiar de escena, pero que no se duplique.....	339
89. Haz que dos objetos se intercambien los nombres cuando colisionan.....	342
90. Crea cinco cubos. Cada cubo debe corresponderse con los números 1 al 5. Al pulsar uno de esos números, el cubo correspondiente aumenta en $\theta \cdot 1$ su escala en Y. Usa solo un <i>script</i> para todos los cubos.	345
91. Escribe un <i>script</i> que imprima por consola el número de objetos que haya en la pantalla cada vez que se pulse la tecla espaciadora.	348
92. Haz que un objeto que sale por la parte derecha de la pantalla entre por la izquierda y viceversa (como los asteroides y la nave del mítico Asteroids).....	349
93. Haz un objeto que avance hacia la cámara mientras la cámara no lo ve.	352
94. Haz que una esfera siga al ratón. La cámara debe estar en $Z = -1\theta$ y la esfera en $Z = \theta$	355
95. Haz que el color de fondo de la cámara cambie de forma progresiva y aleatoria.....	358
96. Crea un <i>script</i> que modifique la gravedad de la escena cada vez que se pulsan los cursores. La flecha de cada cursor indica la dirección de la gravedad.	362
97. Crea dos capas (<i>layers</i>) para cubos y esferas y crea algunos cubos y esferas con sus correspondientes capas. Deja las capas desactivadas en la propiedad <code>Culling Mask</code> de la cámara. Luego haz un <i>script</i> que active y desactive la capa de los cubos pulsando la tecla C y la de las esferas pulsando la tecla E.	364

98. Añade un SphereCollider a una luz de tipo point. Haz que el radio del Collider sea igual al rango de la luz. Haz que el rango de la luz aumente si se pulsa el botón izquierdo del ratón y disminuya si se pulsa el derecho.	369
99. Genera un triángulo desde código.	372
100. Genera un plano desde código (usando dos triángulos).	376
Referencias bibliográficas	381
Recursos	385
Contenidos extra	389

SOBRE EL AUTOR

Jorge García Colmenar ha trabajado como programador en importantes empresas (RD Sistemas, Grupo ZED, etc.), implementando bases de datos y algoritmos para juegos por SMS. Además, como autónomo, desarrolló aplicaciones y videojuegos de *smartphone* para clientes como José Luis Moreno o las Hermanas Hospitalarias.

Desde 2013 ha estado permanentemente formando en el desarrollo de videojuegos, habiendo sido profesor y coordinador de grado en escuelas como CES o Alfonso X el Sabio, e impartiendo cursos de formación para ADECCO. Principalmente enseña las asignaturas de programación y diseño de videojuegos, aunque también ha desarrollado conferencias y seminarios en diversos eventos como el Madrid Games Week, Valencia Indie Summit o ESADA.

Jorge es el desarrollador principal de cKolmos Narrative, un estudio *indie* de videojuegos, con el que ha publicado juegos comerciales en ocho idiomas y en plataformas como PlayStation, Nintendo Switch o XBOX, además de haberlo hecho en tiendas de juegos como Steam o Meta Quest, incluyendo juegos de realidad virtual.

Entre sus aficiones está la pasión por el go, un juego muy relacionado con la inteligencia artificial, tema de su interés y en el que tiene estudios de máster por la UNIR.

INTRODUCCIÓN

Estimado estudiante:

Me complace presentarte **100 ejercicios en C# para Unity**. Este libro es una recopilación esencial de ejercicios para aquellos que desean mejorar sus habilidades en la programación de videojuegos utilizando el lenguaje de programación C# y el *software* Unity.

Como profesor de programación y desarrollo de videojuegos, he tenido la oportunidad de enseñar a muchos jóvenes creadores a lo largo de los años. Muchos de mis estudiantes han expresado interés en mejorar sus habilidades en la programación de videojuegos, pero no saben por dónde empezar o se pierden en el bosque de manuales de C# no específicos.

Durante años he ido creando y probando ejercicios para mis clases. En este libro he recopilado los más eficaces y divertidos y he descartado los aburridos o los que son excesivamente complejos. Están diseñados para ayudar de manera sistemática y eficaz a los nuevos desarrolladores a mejorar sus habilidades de programación de videojuegos en C# y Unity.

El libro presenta una serie de ejercicios que abarcan una amplia gama de conceptos, desde la creación de objetos en la escena hasta la implementación de mecánicas de juego complejas.

Gracias a los ejercicios, tendrás la oportunidad de aplicar los conceptos teóricos que hayas aprendido a la creación de juegos en Unity, mejorando así tus habilidades en la programación de videojuegos y tu capacidad para crear juegos emocionantes y dinámicos.

Mi objetivo es proporcionar una guía **práctica y efectiva** para mejorar tu kungfú en la programación de videojuegos porque, para mí, la programación de videojuegos es como un arte marcial. Descubrir, aprender y practicar cada técnica sirve para comprendernos mejor a nosotros mismos y para realizar proezas asombrosas.

Pero antes son precisos dos matices...

Cómo no usar este libro

100 ejercicios en C# para Unity no es un manual de programación ni un curso para empezar a programar. Necesitarás aprender los conceptos teóricos por tu cuenta antes de comenzar con este libro. Sin embargo, la información está estructurada para que puedas usarlo a la vez que estudias programación. No he dudado en incluir un montón de explicaciones ni tampoco en repasar conceptos que seguramente hayas estudiado, o que quizá no conozcas aún. Pero la información está estructurada con base en los ejercicios que, aunque siguen un orden parecido a aquel en el que se estudia el código, pueden variar mucho. Consulta las secciones y el índice para saber cuáles son los ejercicios más apropiados para ti y altera el orden como creas conveniente. Si tienes dudas, sigue el orden del índice, ya que está pensado para un aprendizaje con dificultad progresiva.

¿Cómo usar este libro?

Este libro se estructura en un listado de ejercicios, cada uno de ellos con la manera de llegar a su solución paso a paso y con comentarios sobre el método utilizado. La mejor forma de abordarlos es leer el enunciado y tratar de resolver el ejercicio en Unity. Los ejercicios están diseñados para que sea indiferente la versión de Unity que uses. Cada uno tiene un tiempo estimado de resolución.

Se da por hecho que no sepas resolver el ejercicio sin buscar documentación en la que apoyarte. Una recomendación que puedo hacerte es que, si no tienes ni idea de por dónde comenzar a resolverlo, lo divides en partes pequeñas y comienzas a solucionar lo que puedas. No tengas miedo de buscar respuestas en internet, mirar la documentación de C# y Unity, etc. De lo que se trata es de aprender y mejorar.

Es muy importante que no te frustres si no consigues resolver un ejercicio. Lo principal es que trates de solucionarlo por tu cuenta, sin mirar la respuesta antes de que se consuma todo el tiempo asignado. Gasta todo el tiempo del ejercicio en pensar formas de abordar el problema e investigar

alternativas. Cuando el tiempo acabe, tanto si lo has conseguido como si no, aplica la solución ofrecida y compárala con la tuya. Esta es la mejor forma de aprender a programar videojuegos. Si es doloroso o agobiante, piensa en el esfuerzo que le llevó a Son Goku, Aang o a la novia de Kill Bill mejorar sus habilidades, y nunca te des por vencido.

Apunte para profesores

Si eres profesor de programación y usas Unity, espero que encuentres en este libro una ayuda para tus clases. Diseñar y probar buenos ejercicios, como sabes, es un trabajo arduo y poco reconocido. Con este compendio podrás diseñar rutas de aprendizaje en tus clases o sorprender a tus alumnos con problemas que no se esperan. Muchos ejercicios plantean una forma básica en sus soluciones, pensada para desarrolladores que aprenden y no para *seniors* con muchas horas de vuelo, por lo que, en ocasiones, si hay una forma más eficiente pero más compleja de resolver el ejercicio, se menciona, pero no se profundiza en ella.

¡Vamos a empezar!

PRIMERA PARTE

En esta primera parte nos centraremos en los ejercicios más básicos para acostumbrarnos al uso de dos programas: el editor de código y el motor de videojuegos. El **editor de código** es el *software* donde escribimos nuestros *scripts*. El **motor de videojuegos** es donde se renderizan los gráficos y se ejecutan estos *scripts* para conseguir funcionalidad. Nuestro motor de videojuegos será **Unity** y nuestro editor de código podría ser cualquiera, hasta un bloc de notas, ya que el código no es más que texto.

Es importante destacar que el entorno de desarrollo integrado (IDE, por sus siglas en inglés) **Visual Studio** tiene integración con Unity y se instala a la vez, por lo que es el más usado como editor de código. Su resaltado de sintaxis y ayudas al programador son muy apreciadas y mejoran el tiempo de desarrollo, facilitando mucho el proceso de trabajo.

CIEN EJERCICIOS BÁSICOS DE PROGRAMACIÓN

Vamos a comenzar con un calentamiento. Los ejercicios listados a continuación solo requieren el uso de un editor de código asociado a Unity, que en tus clases de programación te habrán enseñado a utilizar.

Cada uno de los siguientes diez ejercicios puede hacerse en un *script* independiente, asociado a un objeto cualquiera de la escena de Unity. No vamos a usar aún ningún componente ni nada relacionado con videojuegos, simplemente vamos a practicar un poco con el código. Ten localizada la consola de Unity para ver los resultados de los ejercicios y comprobar si todo ha ido bien.

NOTAS

(puedes tomar anotaciones aquí)

 NOTAS

La dificultad de este calentamiento es **incremental**, siendo los primeros ejercicios muy fáciles y los últimos, algo más complejos. El ejercicio 10 es el jefe de área. Si crees que ya dominas bien la programación en C# puedes tratar de resolverlo y saltarte los nueve primeros, pero si fallas deberás comenzar desde el principio. ¡Nada de trampas!

1. Crea una nueva variable llamada **velocidad** y asígnale el valor de **5.0F**. Imprime su valor por consola.



Tiempo: 5 minutos

```
using UnityEngine;
public class EjemploVelocidad : MonoBehaviour
{
    // Declaración de la variable velocidad con valor
    // inicial de 5.0F
    public float velocidad = 5.0F;

    void Start()
    {
        // Imprime por consola el valor de la variable
        // velocidad
        Debug.Log("La velocidad es: " + velocidad);
    }
}
```

Este *script* en C# para Unity comienza importando el espacio de nombres `UnityEngine`, esencial para acceder a las funciones específicas del motor de juego. La clase `EjemploVelocidad` se declara heredando de `MonoBehaviour`, lo que permite que nuestro código pueda utilizar la funcionalidad de la librería de Unity.

Dentro de la clase, declaramos una variable pública llamada *velocidad* con un valor inicial de `5.0F`. Las variables marcadas como públicas se pueden ver en el inspector de Unity y permiten modificar sus valores iniciales.



El método `Start()` se emplea para ejecutar código al inicio del juego. En este caso, utilizamos `Debug.Log` para imprimir en la consola un mensaje informativo que incluye el valor de la variable. Esto sirve para depurar el código durante el desarrollo del juego, encontrar *bugs* o revisar el comportamiento del *script*, aunque un método más avanzado implicaría el uso de depuradores integrados en IDE como **Visual Studio**.

El código está estructurado de forma que la declaración de variables se encuentra al inicio de la clase, antes de la definición del método `Start()`.

2. Utiliza dos variables públicas de tipo **float** para mostrar por consola sus valores sumados, restados, multiplicados y divididos.



Tiempo: 5 minutos

```
using UnityEngine;
public class OperacionesMatematicas : MonoBehaviour
{
    // Dos variables públicas de tipo float
    public float numero1 = 10.0F;
    public float numero2 = 5.0F;

    void Start()
    {
        // Operación de suma
        float suma = numero1 + numero2;
        Debug.Log("Suma: " + suma);
    }
}
```

 NOTAS

```
// Operación de resta
float resta = numero1 - numero2;
Debug.Log("Resta: " + resta);

// Operación de multiplicación
float multiplicacion = numero1 * numero2;
Debug.Log("Multiplicación: " + multiplicacion);

// Operación de división (evitando división
// entre cero)
if (numero2 != 0)
{
    float division = numero1 / numero2;
    Debug.Log("División: " + division);
}
else
{
    Debug.Log("No se puede dividir entre cero.");
}
}
```

Estamos usando dos variables públicas de tipo `float`: `numero1` y `numero2`. Las variables de este tipo están pensadas para hacer operaciones matemáticas en coma flotante (con decimales).

Luego, para cada una de las operaciones que queremos hacer, declaramos una nueva variable `float` que almacenará el resultado.

La primera operación es la suma, donde las dos variables se suman y su resultado se guarda en la variable `suma`.

A continuación, se ejecuta la operación de resta, donde el valor de `numero2` es restado de `numero1`. El resultado se almacena en la variable `resta`, y se imprime en la consola el mensaje «Resta: [resultado de la resta]».

Posteriormente, se lleva a cabo la operación de multiplicación, donde los valores de `numero1` y `numero2` se multiplican entre sí. El resultado se almacena en la variable `multiplicacion`, y se imprime en la consola el mensaje «Multiplicación: [resultado de la multiplicación]».

Finalmente, se aborda la operación de división. Antes de realizar la división, se verifica si `numero2` es diferente de cero para evitar divisiones entre cero. En caso afirmativo, se procede con la operación y el resultado se almacena en la variable `division`. Se imprime en la consola el mensaje «División: [resultado de la división]». Si `numero2` es cero, se imprime el mensaje «No se puede dividir entre cero».



3. Crea dos variables privadas de tipo `string`, concaténalas y muestra el resultado por consola.



Tiempo: 5 minutos

```
using UnityEngine;
public class ConcatenacionStrings : MonoBehaviour
{
    // Dos variables privadas de tipo string
    private string cadena1 = "Hola, ";
    private string cadena2 = "mundo!";

    void Start()
    {
        // Operación de concatenación
        string resultadoConcatenacion = cadena1 +
            cadena2;

        // Muestra el resultado por consola
        Debug.Log(resultadoConcatenacion);
    }
}
```

 NOTAS

En este caso declaramos dos variables de tipo `string`: `cadena1` y `cadena2`. Al ser privadas, estas variables solo son accesibles dentro del ámbito de la clase en la que se definen. Asignamos algo de texto a cada variable.

A continuación realizamos la operación de concatenación, donde se combinan los valores de `cadena1` y `cadena2` en una nueva variable llamada `resultadoConcatenacion`. Esta variable ahora contiene la unión de ambas cadenas de texto.

Finalmente, utilizando `Debug.Log`, imprimimos en la consola de Unity el resultado de la concatenación para ver la cadena concatenada.

Este ejercicio sirve para demostrar **cómo podemos combinar cadenas de caracteres**.

4. Crea una variable privada booleana y dos variables públicas `int`. Si al ejecutar el juego la suma de las variables `int` es par, la booleana debe ser `true`. Muestra su valor por consola.



Tiempo: 5 minutos

```
using UnityEngine;
public class VerificacionSuma : MonoBehaviour
{
    // Variables privadas
    private bool esSumaPar;
    // Variables públicas
    public int numero1 = 5;
    public int numero2 = 7;
```

```
void Start()  
{  
    // Calcula la suma de las variables públicas  
    int suma = numero1 + numero2;  
    // Verifica si la suma es par  
    esSumaPar = (suma % 2) == 0;  
    // Muestra el resultado por consola  
    Debug.Log("¿La suma es par? " + esSumaPar);  
}  
}
```



En el método `Start()`, se realiza la suma de las dos variables enteras. Luego, mediante una operación de módulo (%), se verifica si la suma es divisible entre 2, lo que indica si es, o no, un número par. El resultado de esta comprobación se almacena en la variable booleana `esSumaPar`. Para realizar esta comprobación hemos utilizado el operador de comparación. Normalmente solemos verlo dentro de una condición (`if`), pero se puede usar también de esta manera. Devolverá `true` o `false`, como siempre, y ese valor se asigna a la variable booleana.

Finalmente, se utiliza `Debug.Log` para imprimir el resultado de la verificación en la consola de Unity. El mensaje indica si la suma de las variables enteras es un número par o no lo es. Este proceso permite una visualización rápida y eficiente de la condición booleana resultante.