Rúbrica

General

- Entrega correcta: Sí/No es el resultado de comprimir la carpeta *GrupoXY*
- *Assets* organizados: Sí/No (caos de *Assets*)
- Errores de compilación o ejecución: No/Sí **muy grave**
- Warnings: No/Sí
- Escenas pedidas: Sí/Número o nombre de las escenas que faltan
- Escenas correctas: Sí/Número del problema
- Mecánicas: Ok/Número de la que falla

Escenas correctas

Número de las especificaciones que fallan:

1. Cámara: color sólido de fondo, ortográfica, tamaño 16
2. *Sprites*: importación 8 px/ud (tamaño), sin filtro y sin compresión (nitidez)
3. *Sorting layers*: necesarias capas de ordenación y/o establecer el orden de dibujado de los *sprites* (igual orden e igual capa, lo dejáis al azar)
4. *Layers*: *Player, Bricks, Ball, PowerUps, Fire*, con detección de colisiones de *Bricks* con *Fire* y *Ball* y de *Player* con *PowerUps* y *Ball*; no necesarias las detecciones de colisiones entre objetos de una misma capa
5. *Prefabs* suficientes (al menos: *GameManager*, *Enlarge*, *Laser*, *Brick*, y *Fire*, sin pasarse)
6. La puntuación y corazones se mantienen dentro de la pantalla cuando cambiamos las dimensiones de la misma (colocados a pincho, sin usar anclas)
7. El panel (GO de la UI) se redimensiona y se reposiciona adecuadamente cuando cambiamos las dimensiones de la pantalla (a pincho, sin usar anclas)
8. El canvas tiene organizados jerárquicamente sus elementos y usan los *layouts* adecuados
9. Los botones han de tener configuradas las transiciones por *Color Tint*
10. Los botones deben comunicarse con el *GameManager* para hacer el cambio de escenas

11. Configuración física de los objetos: estáticos (ladrillos, *DeathZone* y bordes no necesitan *rigidbody*), físicos (*power-ups*, pala, balas) y el caso especial de la bola con doble personalidad cinemática/física
12. La pelota debía ser hija directa de la pala
13. Las escenas de prueba no funcionan y/o producen errores de ejecución
14. Falta *GameManager* en las escenas de los niveles
15. Los *power-ups* no tienen el componente `SetInitialSpeed`
16. Sale el mensaje de “un power-up se ha chocado con algo distinto del jugador; debes tener mal la configuración de las capas de colisión” (una pista estupenda)
17. Falta el punto de *spawn* de la pelota como hijo de la pala; no puede ser ella misma su propio punto de *spawn*
18. Sobra `UIManager` en la escena de menú; no hace falta para nada, complica el código del `GameManager` y/o provoca errores de ejecución
19. Falta añadir el componente `DestroyOnCollision` a varios *gameObjects* que han de destruirse ante alguna colisión

Mecánicas

Número de las mecánicas que fallan:

1. La pala tiembla al llegar a los límites laterales o los sobrepasa
2. Los ladrillos no se destruyen al colisionar con la bola y/o las balas
3. Los ladrillos no suman puntos
4. La bola no hace *respawn* al caer a la zona de muerte
5. Las vidas no se actualizan cuando el jugador pierde una bola
6. No se produce el fin de juego cuando el jugador se queda sin vidas
7. El mensaje de ganar/perder no es el adecuado
8. Al cambiar varias veces entre menú y juego se producen errores
9. Al desactivar los *power-ups* no se deshace su efecto
10. La bola se acelera (al chocar con los extremos de la pala o al ir en diagonal o en otras situaciones)
11. Habiendo ganado se puede perder (nivel 2), o habiendo perdido se puede ganar (nivel 1), porque el juego no se interrumpe
12. Al perder un nivel, no se puede volver a la escena de menú
13. El cambio de nivel no respeta la puntuación ni las vidas
14. Funcionamiento errático e incorrecto de los *power-ups*
15. No se produce el paso automático al siguiente nivel al ganar el primero
16. La bola se alarga y se encoge al activar/desactivar uno de los *power-ups*

17. La bola se mueve de forma extraña
18. Los *power-ups* caen a plomo y acelerando
19. Al ganar el último nivel, no se puede volver a la escena de menú
20. Los *power-ups* son destruidos por objetos no previstos (ladrillos o bola, por ejemplo)
21. Hay un número de vidas distinto de 3
22. El *power-up* `Enlarge` alarga la pala un 50% en lugar de un 25%
23. El comportamiento de la bola es totalmente determinista
24. Al volver a jugar no se reinicia la puntuación y/o las vidas

Errores destacables en los scripts

Generales

1. Organización incorrecta del script: variables públicas, variables privadas, métodos, descomposición correcta en métodos.
2. Declaraciones incorrectas de variables: faltan variables configurables desde el editor, hay variables públicas que no deben serlo, variables de instancia de la clase que deberían ser de método, variables de más innecesarias.
3. Modificar o acceder directamente a variables de otra clase **muy grave**
4. No comentar el código razonablemente (escribir comentarios que no aportan nada al código, dejar comentarios de Unity al Start y Update, no añadir comentarios a los nuevos métodos y no hacerlo encima del código sino a la derecha (o debajo), y obligando a hacer *scrolling* para poder leerlos -sin respetar el límite de 80 caracteres por línea-)
5. Dejar métodos Start o Update vacíos
6. No indentar el código razonablemente
7. No usar convenios de letras (primera letra mayúscula en scripts y métodos, primera letra minúscula en variables)
8. No hacer cacheo de componentes o valores en Awake o en Start
9. Accesos repetidos al mismo componente en el cuerpo de un método (se ha de acceder una sola vez y guardarlo en una variable)
10. **Muy grave**: usar variables públicas para obtener acceso a información perfectamente accesible por código
11. Organización del código mejorable
12. Conviene proteger e todo aquello que pueda ser `null` (y mostrarnos, si procede, un mensaje por consola)
13. Es preferible usar la versión de `Instantiate` genérica enseñada en clase

`Instantiate<GameObject>`

14. Importaciones innecesarias de librerías de Unity como `using System.Collections;`

Brick

1. No debe implementarse el método `OnCollisionEnter2D` sino el método `OnDestroy()`, ejecutado por Unity cuando un objeto se destruye
2. No se necesita el componente `DestroyOnCollision` para implementar la funcionalidad pedida
3. La gestión de los golpes o de los puntos no es responsabilidad de este *script*
4. No implementado el comportamiento especificado al destruirse

Deathzone

1. No procede preguntar por la etiqueta de la bola para después preguntar por su componente `LostBall`, sino que directamente se pregunta por el componente y sabemos que si no es `null` es porque se trata de la bola.
2. No procede acceder dos veces seguidas al mismo componente; se debe acceder una sola vez y guardarlo en una variable
3. Falta la destrucción del resto de los objetos que entren en contacto

SetInitialSpeed

1. La variable para el *rb* debería ser local al único método en el que se usa
2. Basta con establecer la velocidad una vez.
3. La velocidad debe ser constante, es decir, no debe variar durante el movimiento
4. Al asociarse este componente a un objeto físico, no tiene sentido que el movimiento sea cinemático ¿…?
5. Este componente no necesita hacer nada una vez por *frame*

DestroyOnCollision

1. No hay ninguna necesidad de preguntar por la etiqueta del objeto con el que se colisiona (menos aún varias veces), independientemente de con quién se colisione, pues siempre se ha de hacer lo mismo, es decir, modificar golpes y, solo si procede: si se suman puntos, avisar a *GameManager*, si hay un objeto que instanciar, hacerlo, y destruirse.
2. Este componente no debe destruir ningún otro objeto, al margen de a sí mismo cuando proceda

Enlarge

1. Código mejorable utilizando un vector para guardar la escala inicial, lo cual facilita su recuperación
2. ¡No tiene ningún sentido hacer el mismo cambio en cada *frame*!
3. No hay código implementado para desactivar el efecto del *power-up*
4. No hay código implementado para activar su efecto cada vez que se **active** el componente
5. También hay que evitar que el cambio a la escala de partida afecte a los hijos al desactivar el *power-up*
6. No deberíais asumir que la escala es 1 (en ninguna de las coordenadas) y cablearla por código
7. El *sprite* no necesita actualizarse en este *power-up*
8. Este componente no respeta las especificiones de implementación: no tener ninguna variable pública y usar la escala
9. El cambio de escala no debe afectar a los hijos del *power-up*
10. No hace falta un booleano para saber si el componente está activado; el componente `PowerUpManager` asegura que si está activado, no se activará de nuevo
11. La escala tiene comportamiento multiplicativo y no aditivo

Laser

1. El acceso al componente debe realizarse una única vez. Para poder volver a utilizarlo basta cachearlo, es decir, guardarlo en una variable en un método que se ejecute una única vez.
2. Es preferible configurar el botón de salto `GetButtonDown("Jump")`
3. ¡No tiene ningún sentido hacer el mismo cambio en cada *frame*!
4. No hay código implementado para desactivar el efecto del *power-up*
5. No hay ninguna escala (ni *collider*) que tocar en el código (y menos aún inventársela cableándola por código); no tiene nada que ver con la funcionalidad de este componente
6. No hay código implementado para activar el efecto cada vez que se active el componente
7. No le corresponde a este *script* ocuparse de establecer la velocidad de las balas
8. No hace falta un booleano para saber si el componente está activado; el componente `PowerUpManager` asegura que si está activado, no se activará de nuevo

9. El *sprite* por defecto debe guardarse **una única vez**
10. No le corresponde a este *script* destruir ningún objeto
11. La lectura de la entrada **no** debe hacerse en `FixedUpdate`

PlayerController

1. Al no ser inmutable el ancho de la pala, no interesa calcularlo solo una vez para usarlo en `HitFactor`
2. No se tienen en cuenta las pulsaciones del teclado correspondientes al *eje horizontal*; solamente las de algunas teclas del teclado
3. La lectura de la entrada **no** debe hacerse en `FixedUpdate`
4. El vector debe normalizarse
5. Falta la implementación del método `HitFactor`
6. La escala no coincide con el ancho del objeto, por lo que no sirve usarla; hace falta la clase `Bounds`
7. No tiene sentido que la velocidad del *rigidbody* dependa de `Time.deltaTime`
8. Cambiar la velocidad de un objeto físico y tele-transportar (`Translate` o `MovePosition`) son cosas distintas

Ball

1. Al convertirse en física la bola (es decir, **en cada lanzamiento**), el vector de dirección debe tomar un **valor aleatorio** (por ejemplo, entre -1 y 1, para tener limitado el ángulo) y debe **normalizarse**
2. No procede acceder dos veces seguidas al mismo componente; se debe acceder una sola vez y guardarlo en una variable
3. Falta la implementación de `OnCollisionEnter2D`
4. Al colisionar con la pala y ajustar su dirección, el vector resultante también debe estar normalizado
5. No tiene sentido que la velocidad dependa de `Time.deltaTime`
6. Cuando es física, la bola no debe ser hija de la pala sino un objeto independiente de ella
7. No procede preguntar por la etiqueta o el nombre de la pala para después preguntar por su componente `PlayerController` , sino que directamente se pregunta por el componente y sabemos que si no es `null` es porque se trata de la pala.
8. Solo procede reaccionar a la pulsación de la tecla que lanza la bola si es cinemática; en otro caso, no debe hacerse nada
9. Si la bola colisiona con la pala es porque no es cinemática, por lo que

sobra comprobar que lo es

LostBall

1. La pelota debe ser hija de la pala
2. La posición de la pelota al *respawnearse* debe ser la misma del punto de *spawn*
3. Como he repetido muchas veces, no está permitido en esta asignatura usar los métodos `Find` y mucho menos de forma repetida: expliqué en los laboratorio que el acceso a la pala es sencillo, pues es el padre del punto de *spawn*
4. También he repetido muchas veces que no debe accederse o modificarse a una variable pública de otra clase desde el código
5. No hace falta guardar en una variable al padre inicial de la bola, puesto que puede conseguirse por ser el padre del *spawnPoint*

UIManager

1. No avisa en el `Start` al *GameManager* de que él es el responsable de la UI
2. `LifeLost` enrevesado o con parámetros de más: basta con desactivar la última posición que está activa del array y esto se puede hacer con una sola línea de código.
3. Lleva conteo de puntos en lugar de tan solo presentar lo que le pida el *GameManager*
4. El método `FinishGame` no recibe un parámetro indicando si se ha ganado o perdido el juego
5. El método `FinishGame` recibe un parámetro innecesario
6. Falta la implementación del método `RemainingLives` para pasar la información de puntos y vidas a la IU ante un cambio de nivel

GameManager

1. Mejorable la implementación de `PlayerLoseLife` : devolviendo el booleano `(playerLives > 0)`
2. Si se usa una variable booleana para saber si se está jugando o no, en algún momento habrá que cambiar su valor para reflejar la realidad…
3. En ningún momento se le pasa información al `UIManager` acerca de la puntuación y las vidas
4. Código reorganizable y mejorable en `LevelFinished`
5. No tiene sentido acceder a un componente del que ya tenemos una referencia (`theUIManager.GetComponent<UIManager>()`): `theUIManager` es de

tipo `UIManager`, es decir, es el propio componente
6. Tiene que haber un control de si se ha ganado o perdido un nivel, para impedir que se pueda ganar habiendo perdido, o perder habiendo ganado
7. Faltan inicializaciones de variables por si se vuelve a jugar cuando el juego ha terminado (se haya ganado o perdido)
8. No tiene sentido inicializar variables al cambiar de escena
9. No hay ninguna necesidad de implementar el método `Update`

Comentarios adicionales

- Sigue absolutamente **PROHIBIDO**:
    - **Usar más variables configurables desde el editor** de las indicadas en el enunciado
    - **Usar `Find` o análogos** (salvo que os lo pida explícitamente)
    - **Modificar o acceder directamente a variables de otra clase**
- Vigilad las **variables**
    - De instancia
    - De método: preferibles, siempre que sea posible
    - De bloque: más preferibles aún
- Repasad bien la **clasificación y configuración adecuada de objetos con *collider***, en función de si se mueven o no, así de cómo se mueven
- Faltan capas de ordenación con el consiguiente peligro de que no se vean los *sprites* correctamente
- Sobran las detecciones de colisiones de las capas físicas definidas consigo mismas, así como las de los ladrillos con la pala, por ejemplo
- Comentarios
    - Algunos que obligan a hacer *scroll* horizontal, lo cual es muy molesto
        - Idem con las líneas de código que sobrepasan los 80 caracteres
    - Algunos que no aportan nada: "// Declaramos variables"
- No procede estar buscando y accediendo al mismo componente una y otra vez (operación costosa que se evitaría guardando una referencia al componente en una variable)
- No puede ser que haya errores de ejecución en las escenas de prueba (o

en las del juego) por no proteger accesos a componentes por si son `null`

- El canvas no puede tener sus elementos colocados a pincho, con posicionamiento absoluto, sin usar anclas ni *layouts* donde se pedía
- Procurad que cuando destruís un objeto, el `Destroy` sea la última instrucción a ejecutar, para curaros en salud y evitar potenciales errores de ejecución
- Es preferible configurar desde el editor de Unity lo que sea posible que tener que hacerlo por código
    - Por ejemplo: gravedad a 0, panel oculto, etc.