#Unity para retrasados

##Como usar la documentación de Unity3D

­­­­­­­­

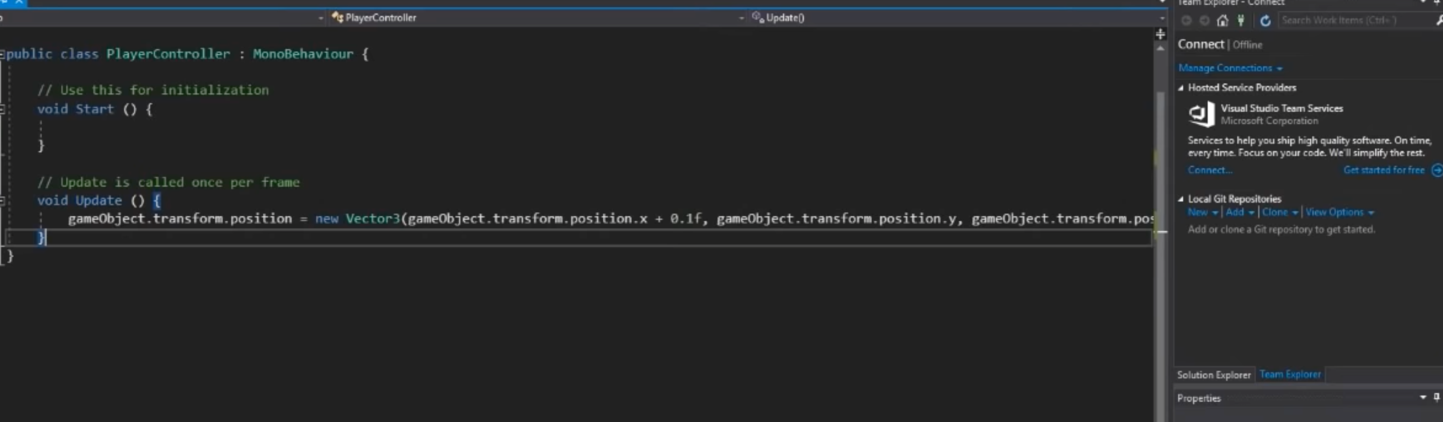
#UPR Movimiento sencillo 2D:

Casi cualquier tipo de objeto se puede crear partiendo de un emptyObject y añadiendo los componentes necesarios, como por ejemplo un sprite 2D (sprite renderer..) Si quisiéramos usar pixel art y añadimos una imagen a nuestro sprite veremos que se ve borroso, eso es porque por defecto Unity suaviza la imagen, se puede cambiar en la configuración del sprite, cambiando el **filter mode** de **bilinear** a **point**. Otra cosa que sucederá es que el tamaño del sprite será minúsculo en comparación con el objetivo de la cámara, lo que se debe hacer es cambiar la config de la cámara y no del sprite para no volver a encontrar este mismo problema en el futuro con otros sprites.

Supongamos que el sprite tiene 100 pixeles por unidad (Pixels per unit) y la cámara renderiza en 30 unidades de tamaño (size en el inspector de la cámara. SOLO SE APLICA AL EJE VERTICAL, osea que tiene 30 unidades de arriba abajo solamente, EL HORIZONTAL DEPENDERÁ DEL TAMAÑO DE CADA PANTALLA.La relación mas usual horizontal/vertical es 16:9 ). Imaginemos que nuestro personaje mide poco mas que 10 pixeles, asi que configuraremos el sprite para que muestre 1 px por ud para que la escala de tamaño muestre la resolución que buscamos.

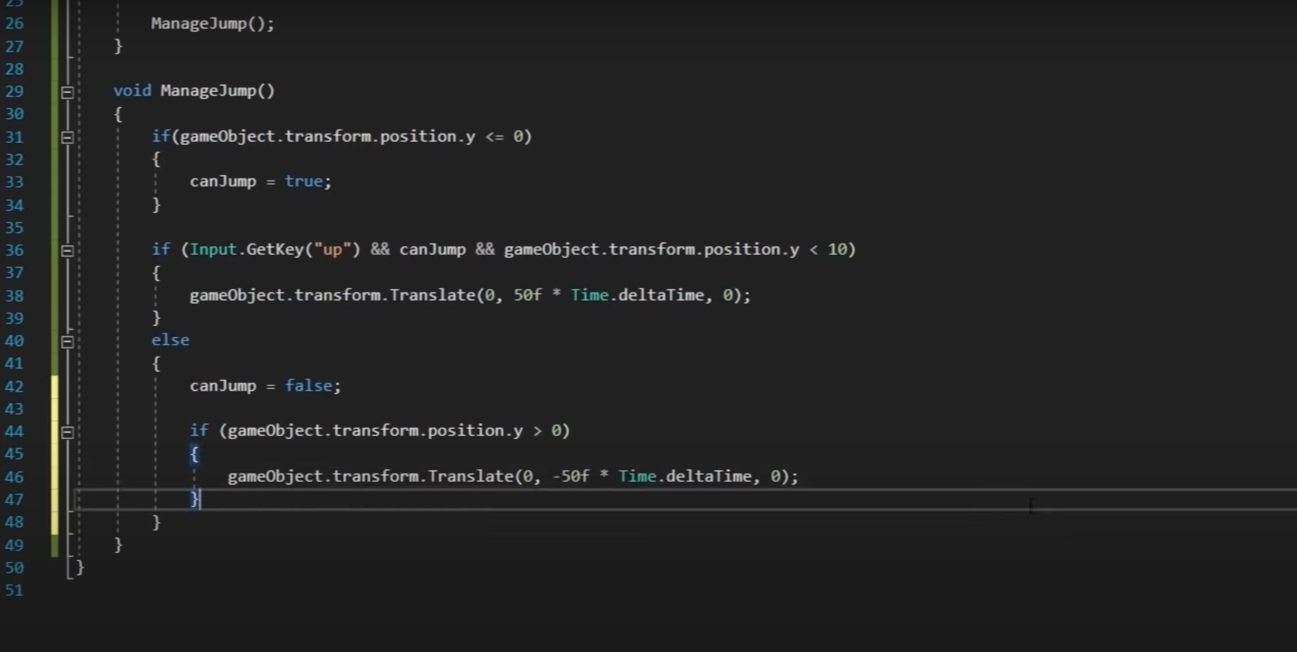


Cuando queremos hacer que un GO se mueva 20 uds a la derecha por ejemplo, esta es una manera incorrecta de hacerlo, porque lo que estamos haciendo es decirle que se mueva a una posición concreta de la escena, independientemente de donde esté, no que se mueva desde su posición actual.

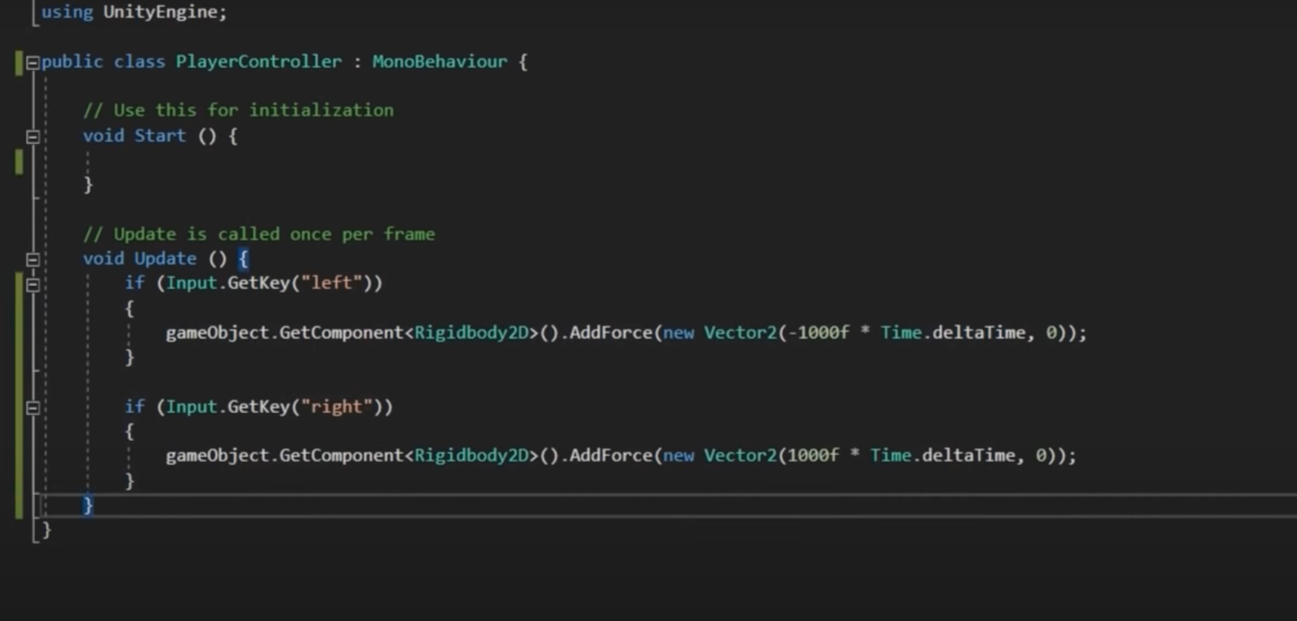


Esta es la forma correcta, hacemos que se mueva desde su posición x, tantos frames como pongamos al numero que sumamos a x en el update, y almacenamos la misma posición que tiene en tanto en `y´ como en `z´. Es importante considerar que se debe multiplicar por time.deltatime el numero de frame que va a avanzar para que se adapte al numero de frame de cada procesador, asi si era 0.1 frames a la derecha, cada frame, con time.deltatime, será por segundo.

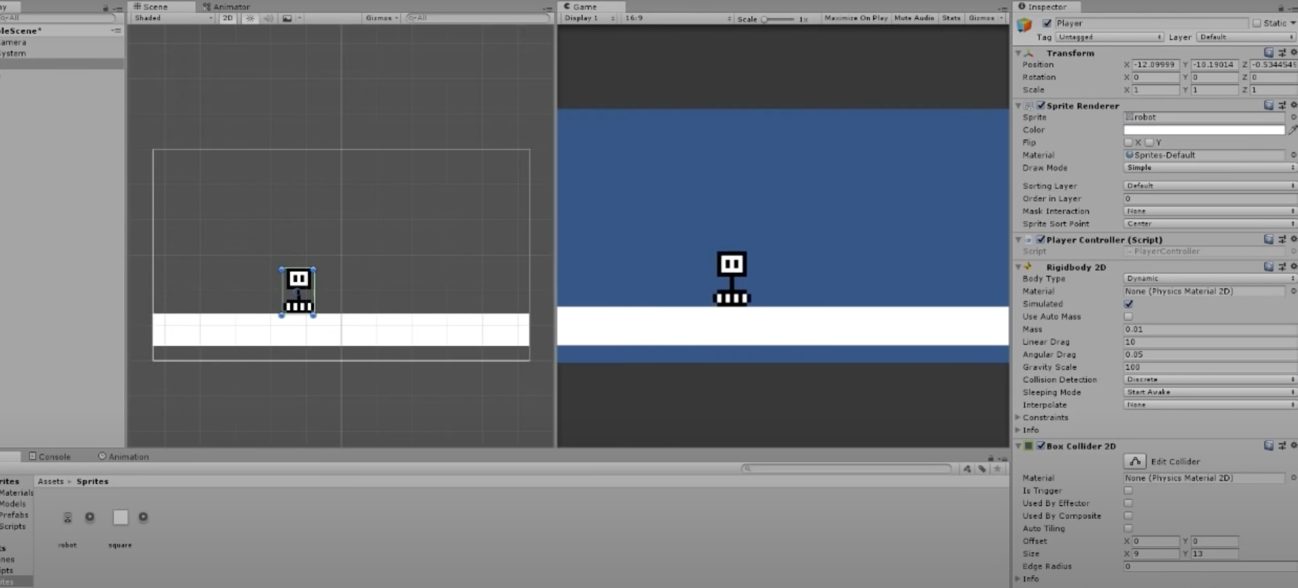
Este sería un método para hacer que salte de forma manual (sin físicas)



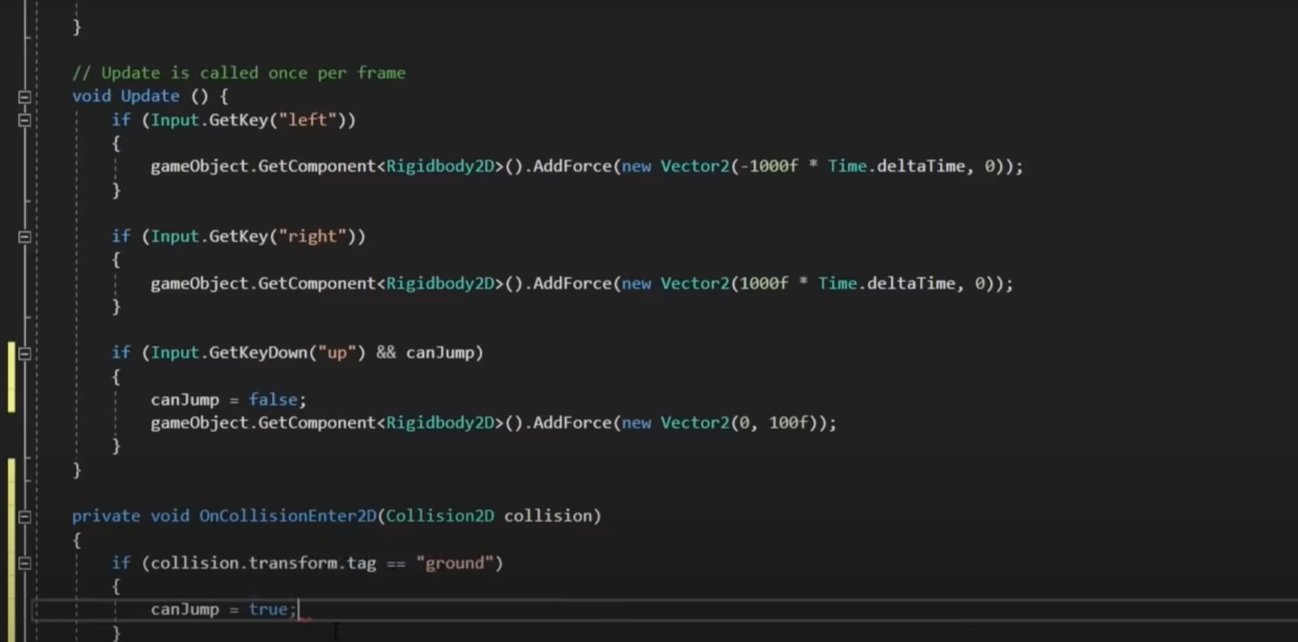
Presta especial atención al uso del booleano canJump. Cuando es true puede santar si es menor que 10 en y, para ponerle limites al salto. Y si es falso, que caiga (dado que no actúa por física.)

Para hacer esto mismo pero usando físicas, lo primero que hay que hacer es añadir un rigidbody 2D para que le afecte la física y un collider 2D para que pueda colisionar (chocar) con otros objetos.

Primera cosa interesante aquí es que necesitamos ponerle tanta fuerza, porque Unity equivale 1 ud a 1m, como hemos configurado antes a nuestro sprite para que tenga 1px por ud significa que tiene varios pxs/metros por 30 uds que recoge la cámara, así que necesitamos mucha fuerza para que se note que se mueve. Los problemas que te encontraras ahora estarán relacionados con la resistencia al movimiento, la gravedad, la masa del objeto… Se trata de ir jugando con los valores, en el ejemplo de este tutorial lo que ha encontrado equilibrado ha sido esto:



\*\*Es importante entender bien como funcionan las físicas, por ejemplo, cuando le ponemos mucha gravedad el objeto se mueve los frames equivalentes a la fuerza que tiene, en este caso podría ser una gravedad de 981(que se hace en edit > Project settings > pshysics 2D), que es 100 veces mas de lo que hay en la tierra y que el gravitiy scale del sprite fuera 100 tambien, esa fuerza hace que se mueva muchos frames y al motor no le da tiempo a calcular si ha colisionado con algo, para solucionar eso podemos poner el collision detection en continous en vez de en discrete, asi comprobará si en cada frame se ha saltado algo de fisica y recalculará para que no atraviese ningún collider. Para añadirle una función de salto lo que haremos será añadirle un tag (que se llame ground, por ejemplo) al suelo de nuestro juego, y utilizar la propiedad collider para que solo salte una vez si esta colisionando con el suelo para que no salte cada vez que pulsamos la tecla:



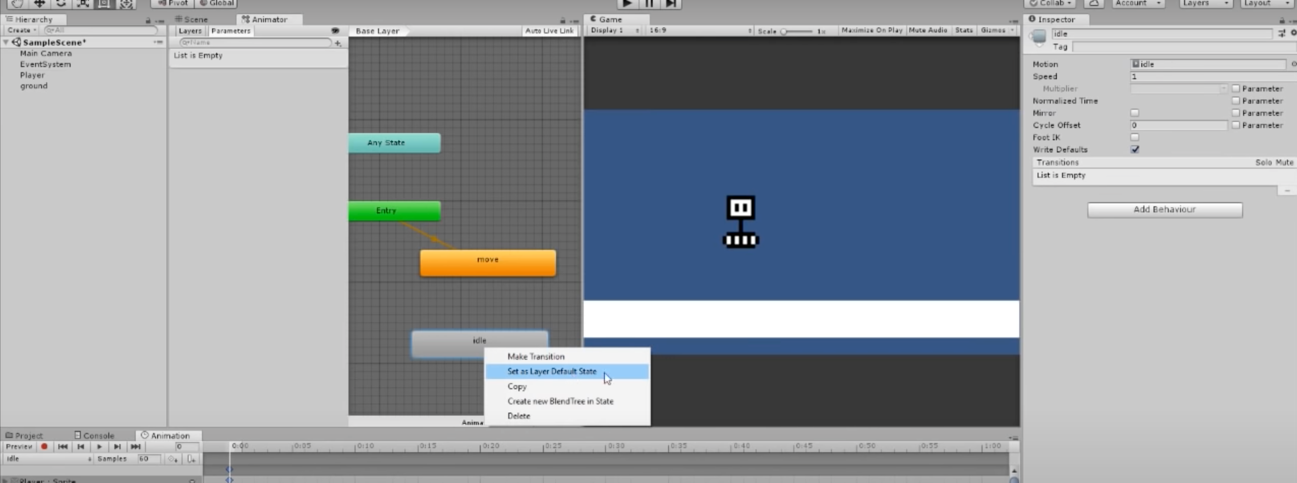
Asi solo salta si el sprite ha tocado primero el suelo (de nuevo presta especial atención al booleano)

#UPR Animaciones:

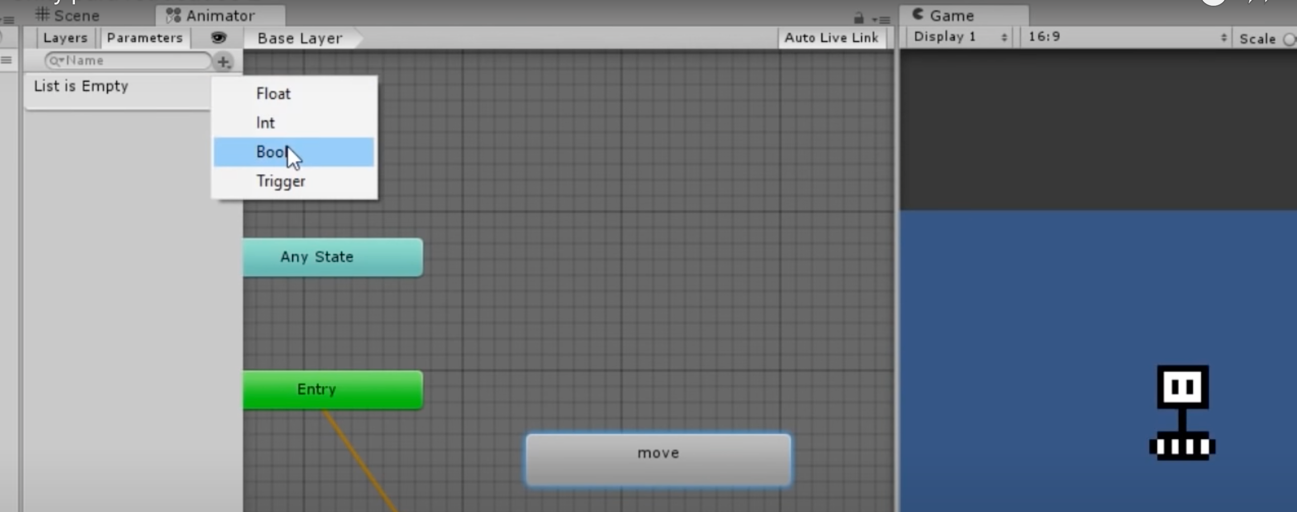
Tenemos que tener un sprite sheet (3 o mas sprites juntos en una sola imagen, uno quieto y otros moviéndose) > para poder usarlo cambiamos el sprite mode del spSheet de single a multiple > sprite editor > en Slice seleccionamos en type: grid by cell size y seleccionamos un tamaño de 9X13. En el GO player ponemos el sprite del sprite sheet y empezamos a trabajar las animaciones: creamos una carpeta para las animaciones y dentro otra para el personaje. Dentro de la carpeta creamos un animation y seleccionamos en el inspector la casilla de loop time (siempre que queramos que se repita. En el caso del ejemplo queremos que se repita cada vez que se mueve el sprite) > arrastramos el componente animation al player (el objeto que vayamos a animar)

En la pestaña de Animation seleccionamos la animación que queramos trabajar.

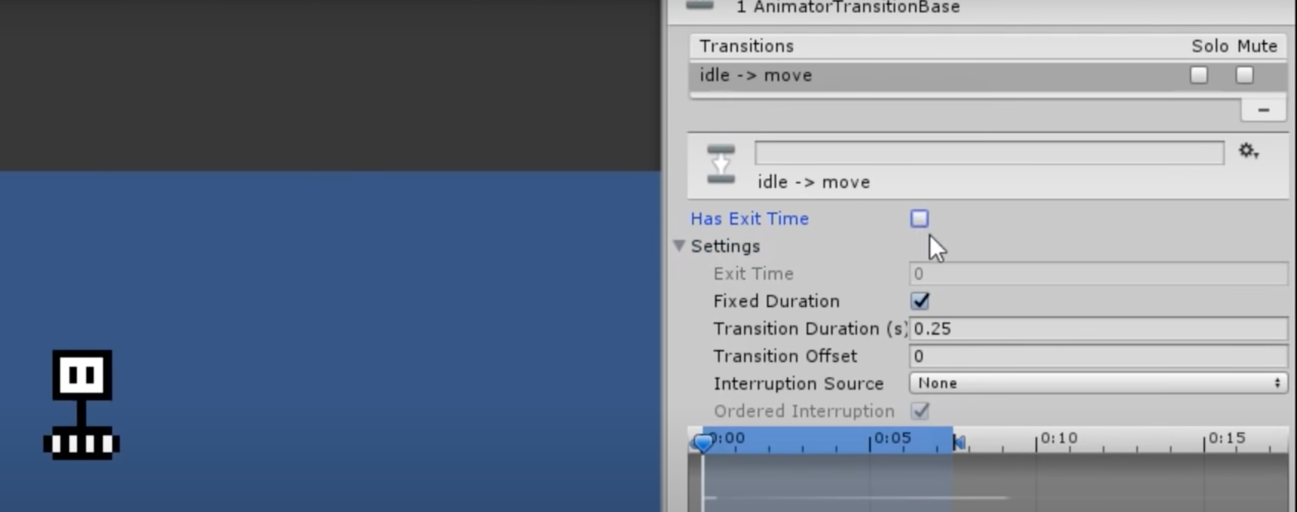
Le damos a grabar y luego cambiamos el sprite en el sprite renderer y elegimos el sprite de la primera animación del sprite sheet, avanzamos al segundo frame y elegimos el sprite de la segunda animación y pasamos al tercer fram para elegir de nuevo la primera animación para que se repita el bucle. El campo samples (los frames a los que se reproduce la animación) seguramente tengamos que modificarlo, en el ejemplo del tutorial lo pone a 10 frames por segundo > pasamos a la ventana Animator, donde tocaremos los nodos de la animación, donde tendremos solamente una animación repetida en bucle, así que añadimos otra a la que llamaremos idle (el player no hace nada) arrastramos de nuevo al player la nueva animación y la trabajamos: seleccionamos la nueva animación en el Animation > Add property > sprite renderer > sprite. Por defecto Unity nos creara dos frames pero solo necesitamos uno, a no se que queramos hacer un animación rollo pestañear o cosas asi (supongo). Hacemos click derecho sobre la animación idle y la ponemos por defecto.

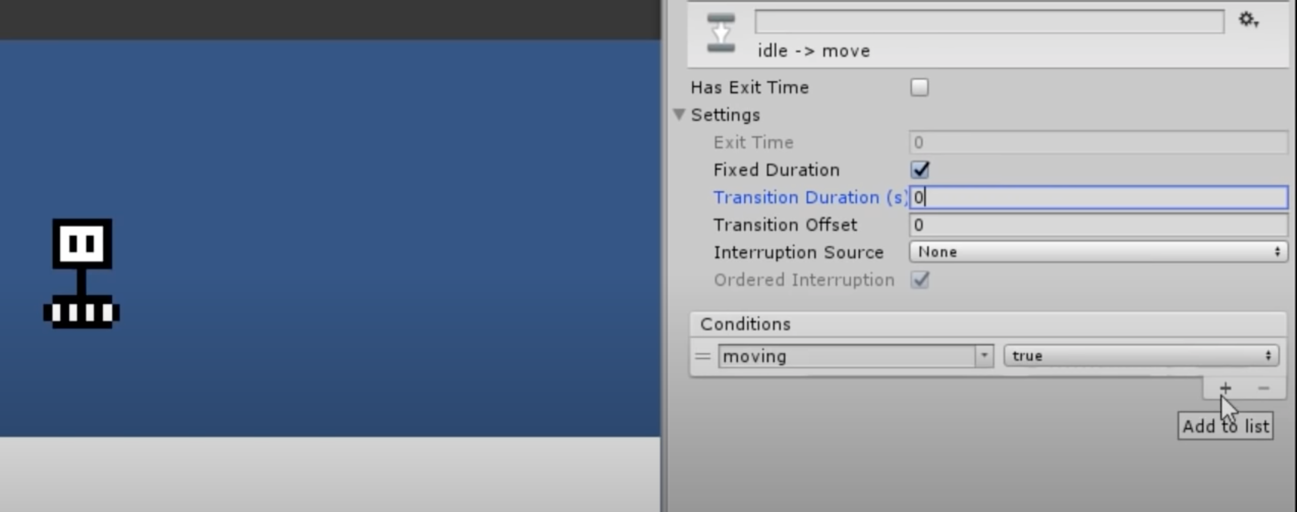


Añadimos un parámetro booleano al que llamaremos moving para ayudarnos luego en código.

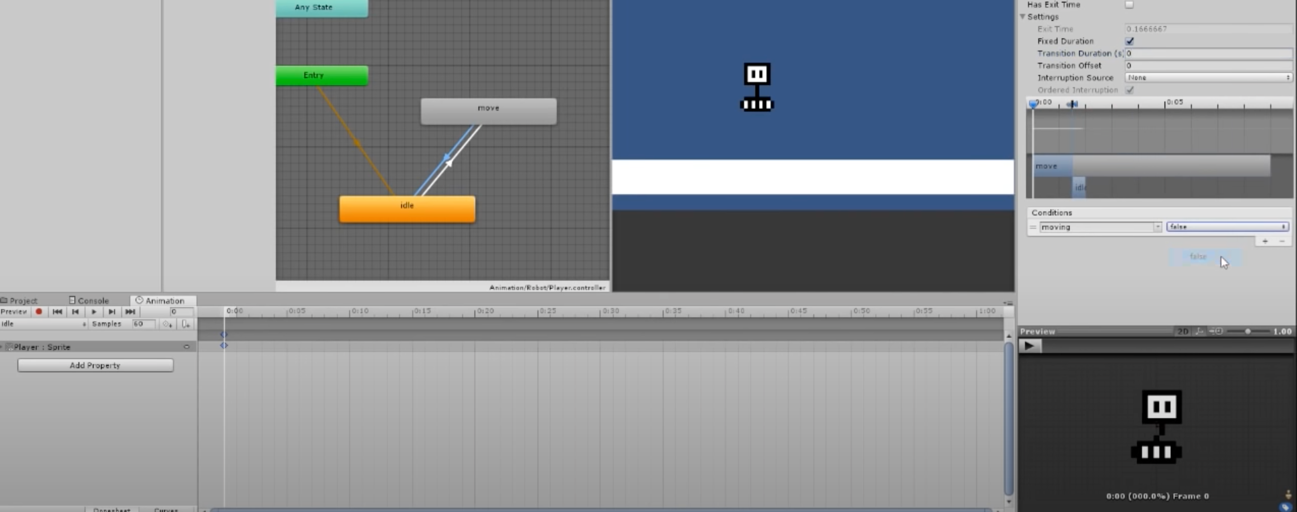


Desde idle hacemos clic D y hacemos una transición hasta move. Seleccionamos la flecha y cambiamos las opciones en el inspector.

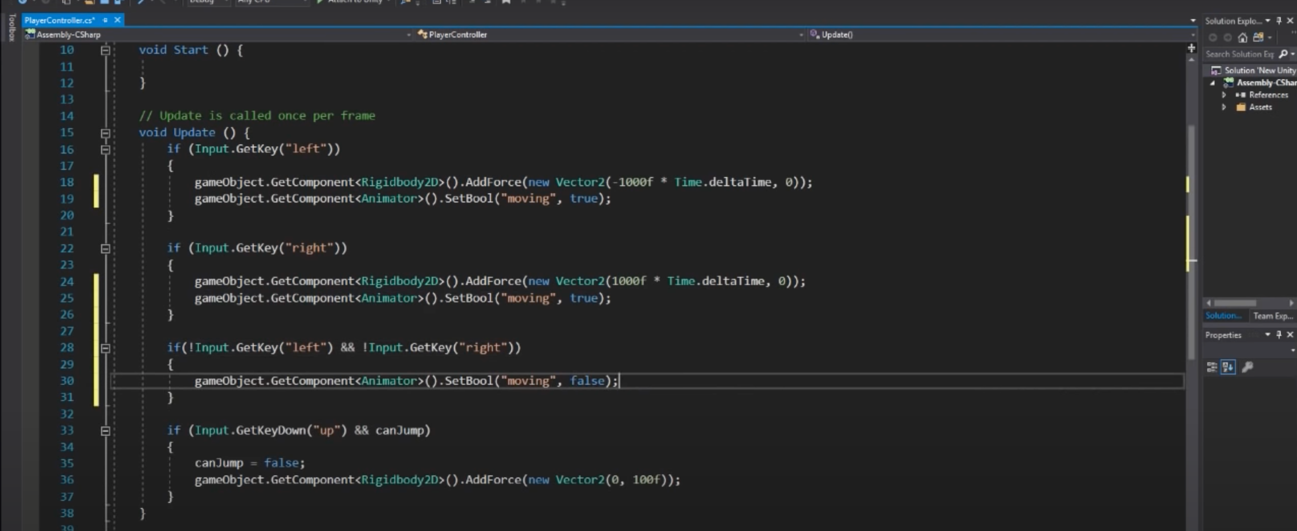


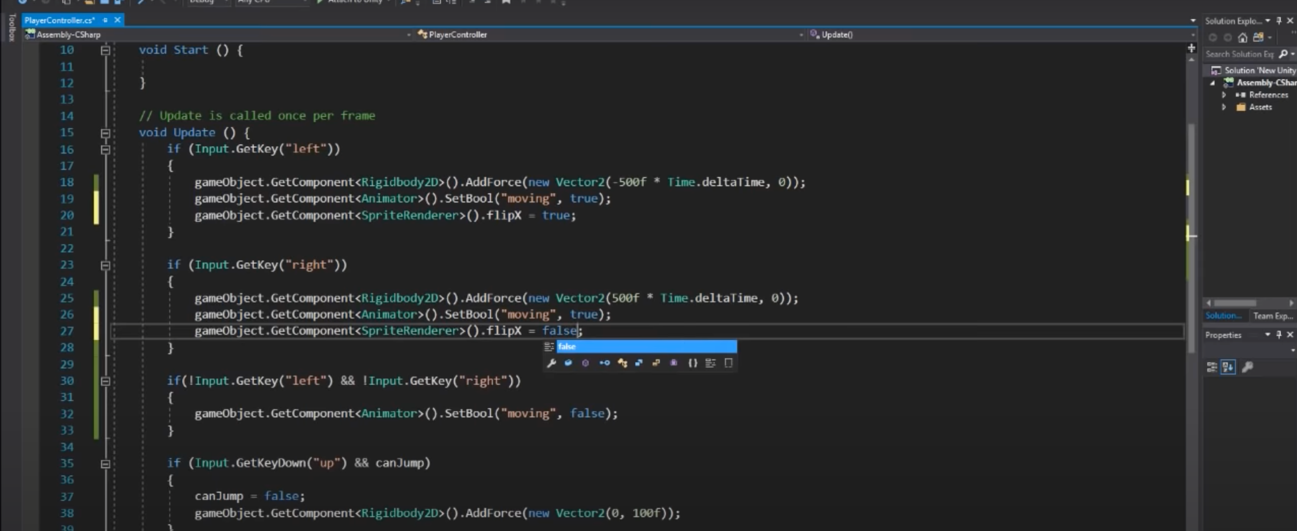
Deseleccionamos la casilla has exit time porque no queremos que tenga un tiempo de salida, va a ser permanente la animación de quieto. Ponemos también la duración de transición a 0.

Y añadimos la condición y la ponemos en true.

Hacemos lo mismo pero desde el movimiento a la quietud y lo ponemos en false.

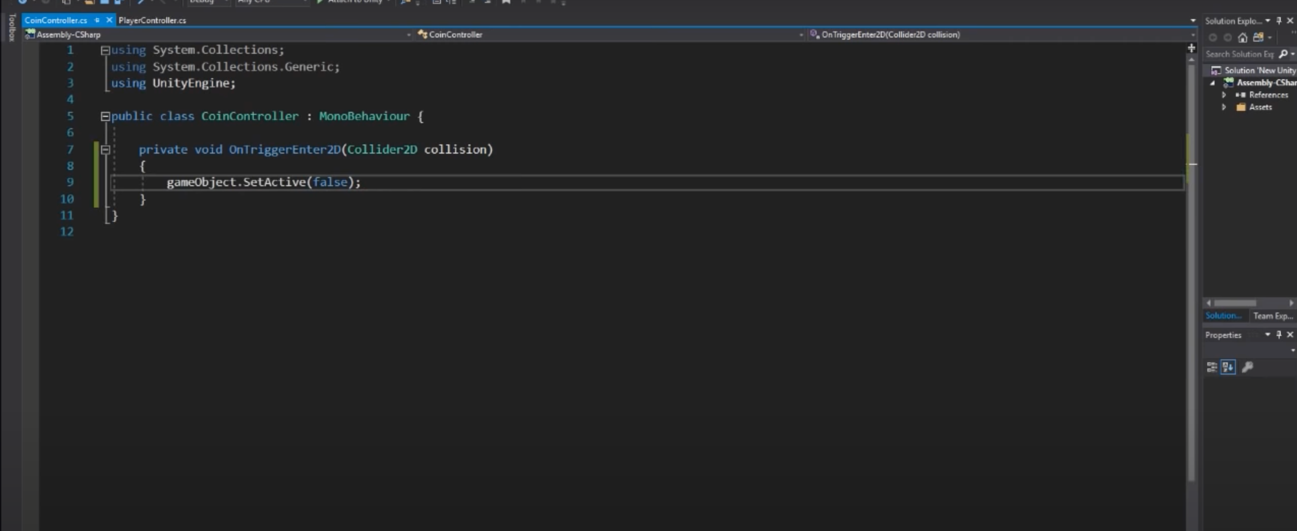
Ahora desde código establecemos las condiciones de transición:

lo que hemos hecho es, donde las condicionales de las teclas de movimiento izquierda y derecha hemos añadido el componente Animator con el boolean. \*\*Reparese en la condición de !Input, es decir si no se toca ni la tecla 🡨 ni la 🡪 entonces moving es false, no se mueve.

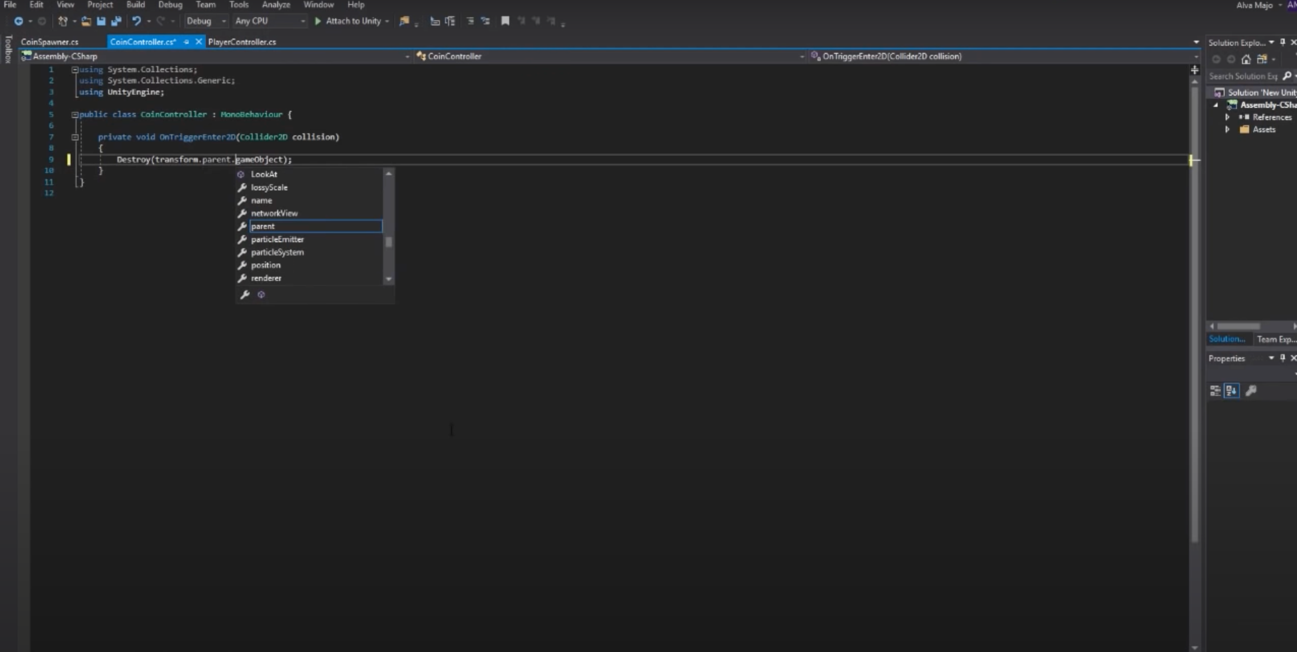
Para solucionar el problema de que solo se anima hacia un lado lo hacemos mediante código modificando el flipX en ambas direcciones (como es un checkbox, pues en código es un boolean)

Ahora vamos a poner monedas en el juego. Añadiremos un sprite de moneda, o un sprite cuadrado liso para hacer pruebas. Añadiremos un box collider 2D y en las opciones pondremos que es un trigger para que no choque con nuestro player sino que solo detecte cuando lo ha tocado. Para continuar con el orden de la clases haremos un nuevo script que se llame CoinController para gestionar lo que hace la moneda. Pondremos la función OnCollisionEnter2D y se lo arrastramos al coin.

(se puede usar setactive o destroy, lo que convenga, incluso hacer una pool\*\*Informate sobre eso)

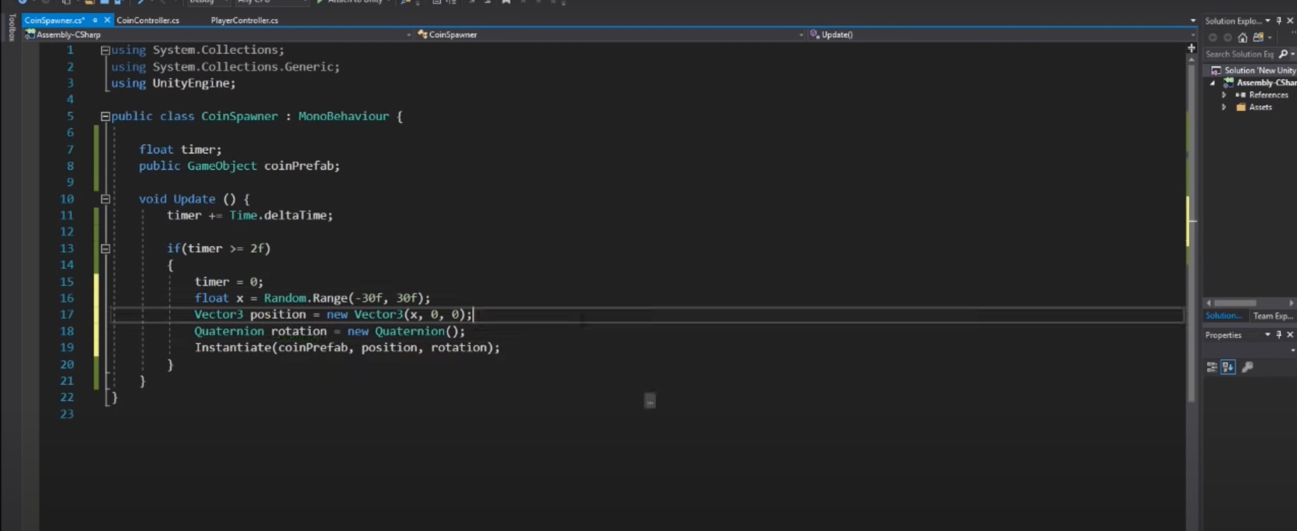


Como hemos hecho más abajo lo de emparentar el coin al empty coinContainer, para destruir la instancia habría que hacerlo así:



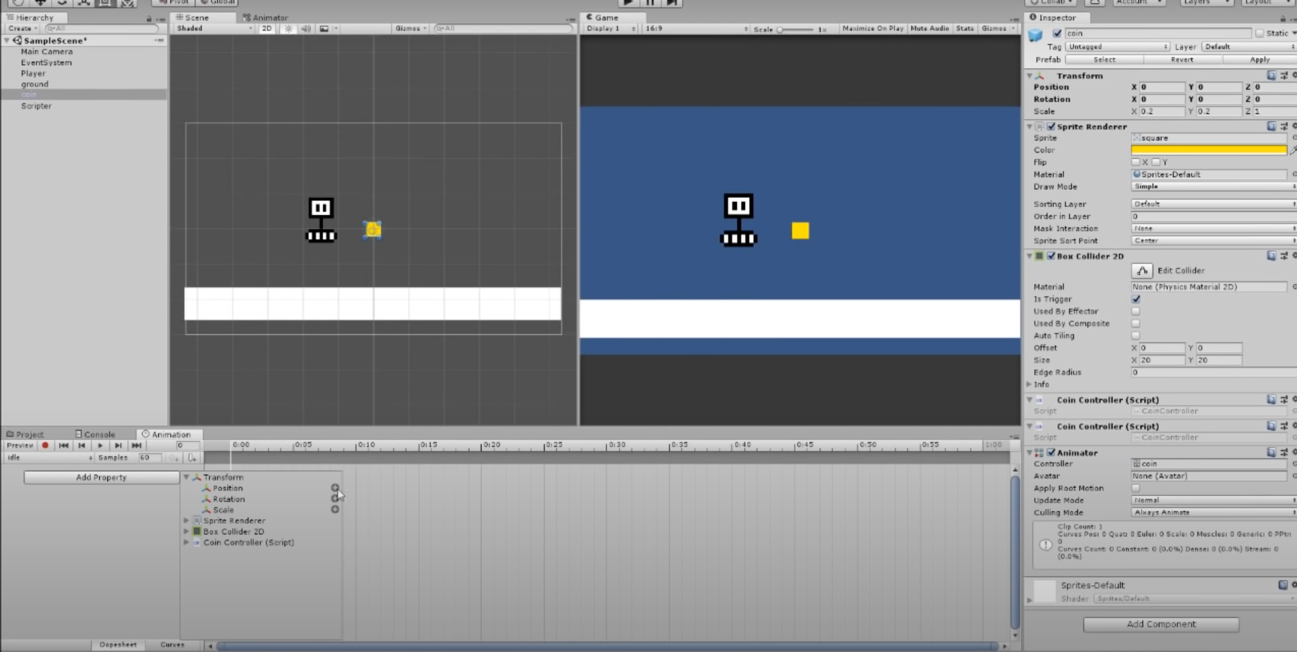
\*\*\*Una forma sencilla de hacer un contador: float timer;

(en el Update)timer+=Time.deltatTime.

Ahora analicemos el código para el CoinSpawner 

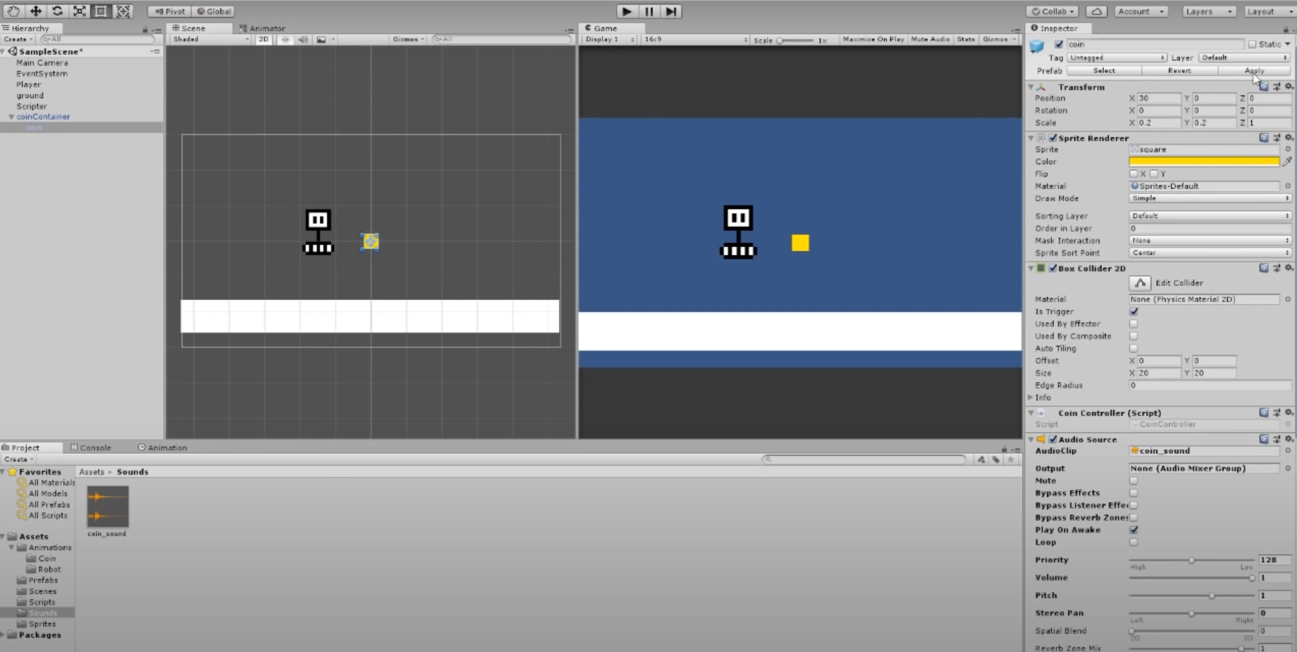
1. Declaramos las variables: el prefab que vamos a usar como moneda, y el contador
2. Escribimos el contador, como especificamos más arriba, en el Update
3. Fijamos la condición para que el bucle sea de dos segundos, creamos una variable local para randomizar la posición en el eje x, creamos un vector de posición con ese randomX, un Quaternion vacío ya que es 2d (aquí se podría usar Quaternion.identity [para alinearse con los ejes del mundo o de los padres]y ahorrar líneas de código)
4. Instanciamos el objeto

Hasta ahora el código que tenemos lo que hace es poner monedas en sitios aleatorios en el eje X, pero imaginemos que queremos que tengan una pequeña animación, un problema usual y poco intuitivo es que al poner una animación a un objeto que queremos que salga de manera aleatoria, saldrán en el mismo punto, donde parte la animación.

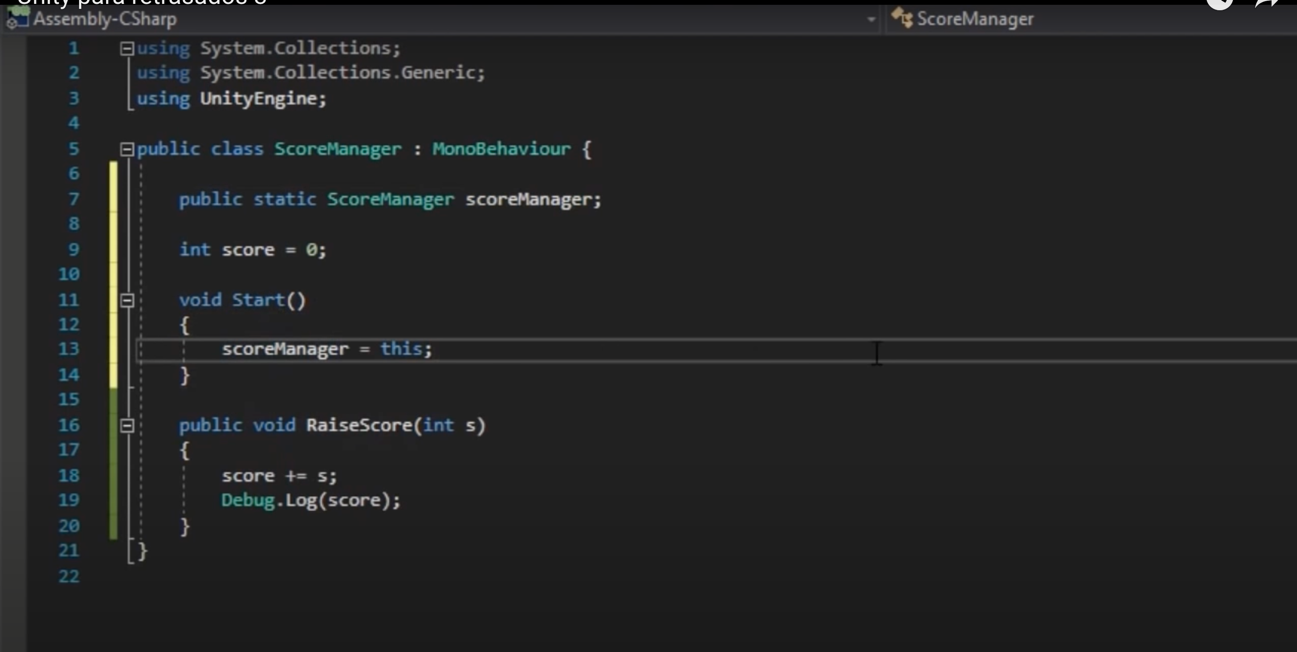


En la nueva animación idle para la coin, tras crearla y activar la casilla loop time, añadimos nuevos keayframes para la propiedad position y en medio de los dos grabamos un nuevo keyframe moviendo la moneda hacia arriba. Pero para solucionar el problema anterior debemos crear un empty llamado coinContainer que contenga la moneda y hacerlo prefab, asi, usaremos en el código el prefab coinContainer arrastrándolo hasta el espacio coin prefab de nuestro script de arriba CoinSpawner que estará esperando en el empty llamado scripter (que usaremos para crear scripts que no requieran estar en ningún objeto, como spawner y cosas asi)

Para poner sonido a la moneda cuando aparezca se le añade un audio source > arrastramos el clip de audio > y dejamos clicada la casilla play on awake para que suene en cuanto se instancie la moneda

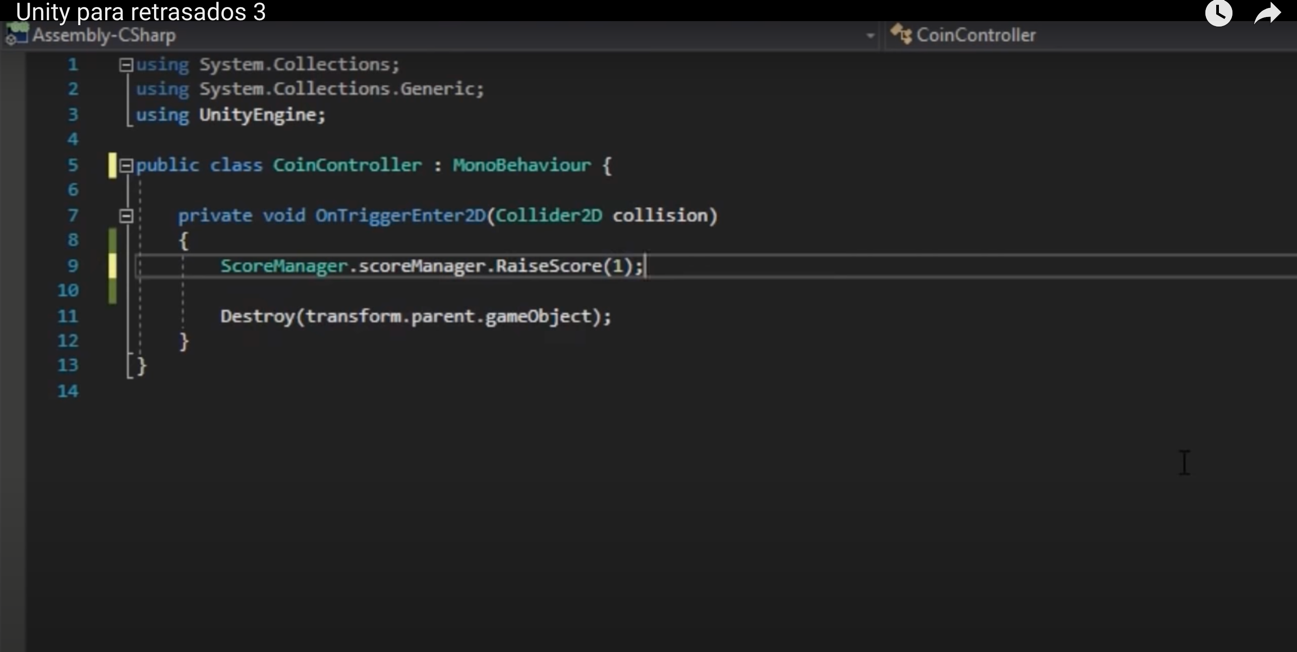


Ahora haremos un ScoreManager (como vamos a llamarla desde otro script hay que hacer la función publica ):



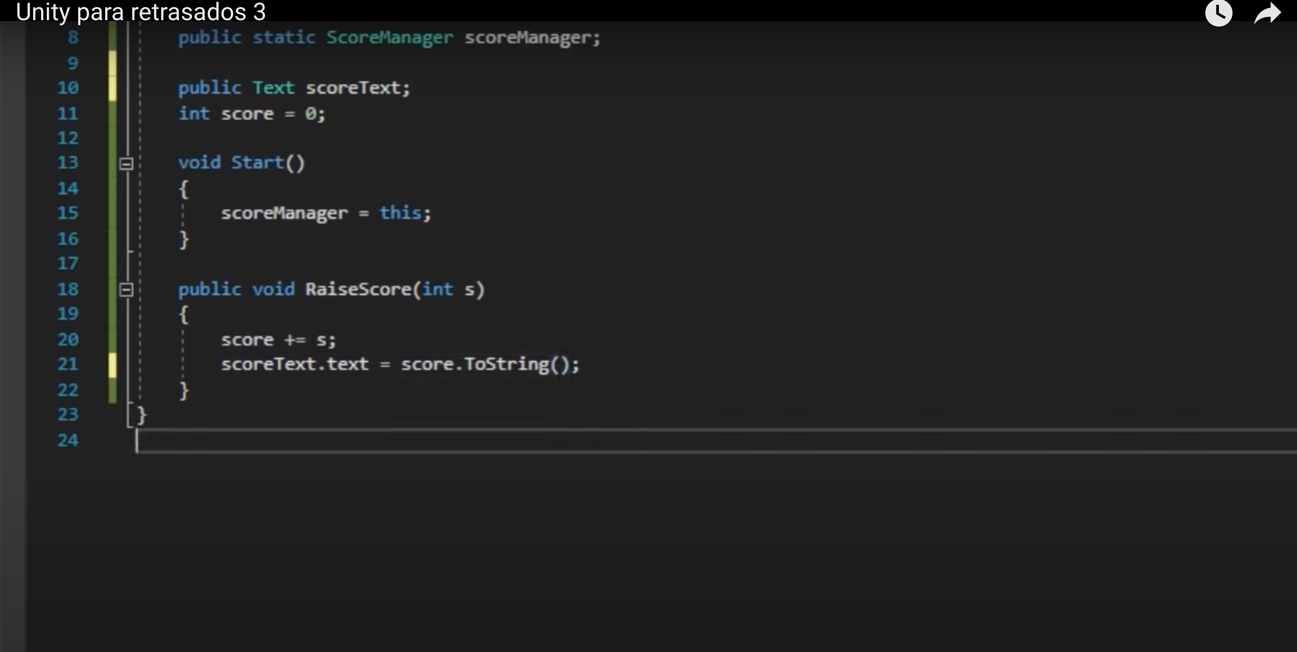
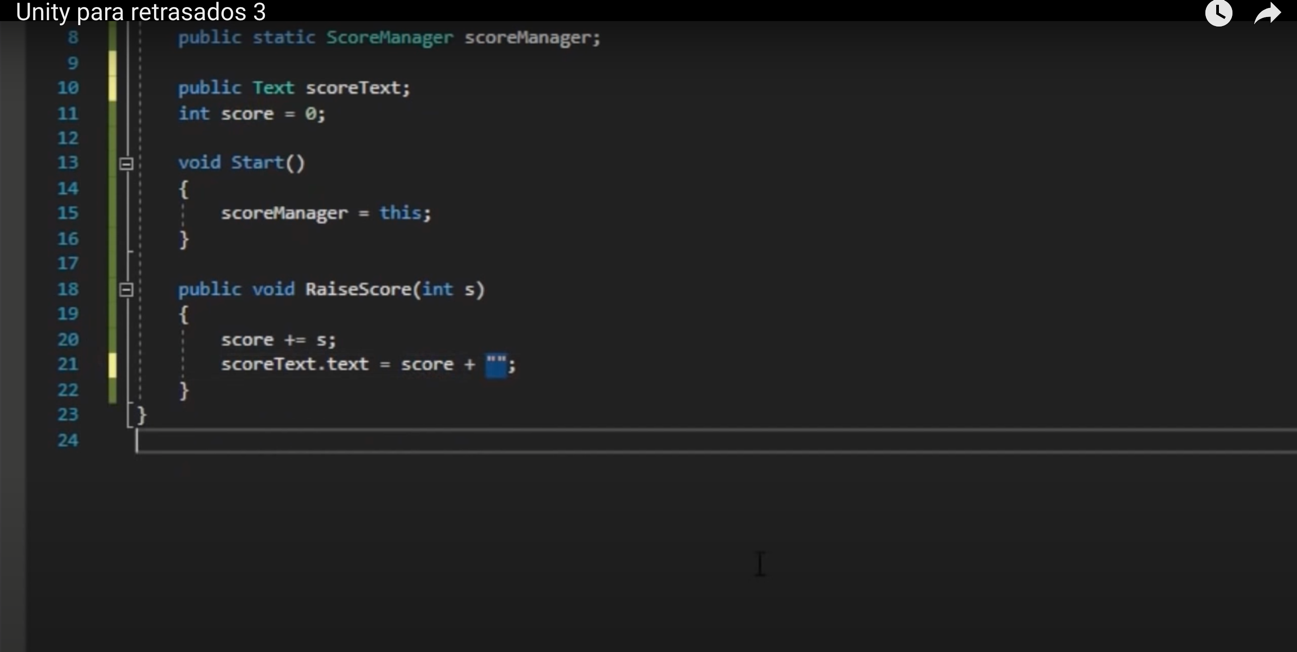
Lo que hemos hecho en este script es:

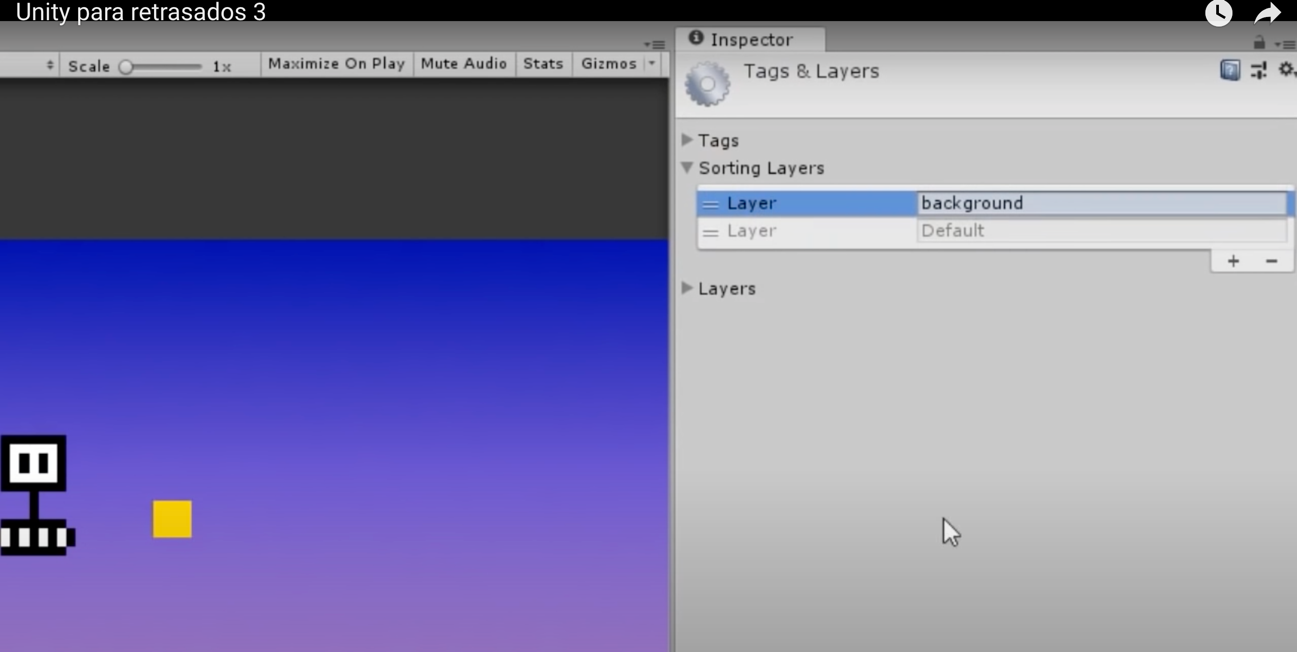
1. Hemos metido la clase ScoreManager en una variable estática (lo cual significa que solo hay una instancia de la misma)
2. Hemos escrito el contador de puntos //\*\*Con scoreManager = this; hemos dicho que la variable se convierte en todo este script



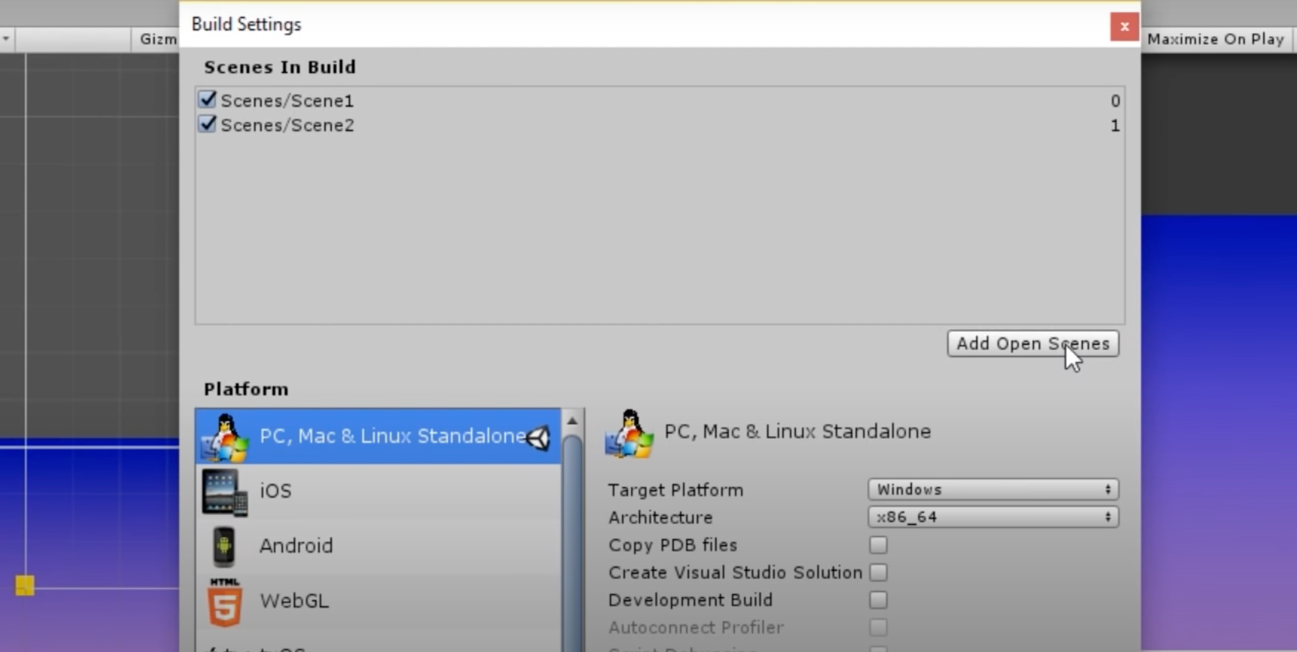
Y aquí llamamos a la función RaiseScore(aumentar puntuación) y le asignamos el valor de devuelto 1 int ya que queremos que cuando el player colisione con el trigger coin se le sume 1 point.

Una vez debugueadas las comprobaciones de la puntuación y comprobado que todo funciona bien ya podemos añadir una interfaz. Primero añadimos la colección de Unity para UI > using UnityEngine.UI; y añadinos un UI text, que nos añadirá un Canvas también, a modo de pantalla.

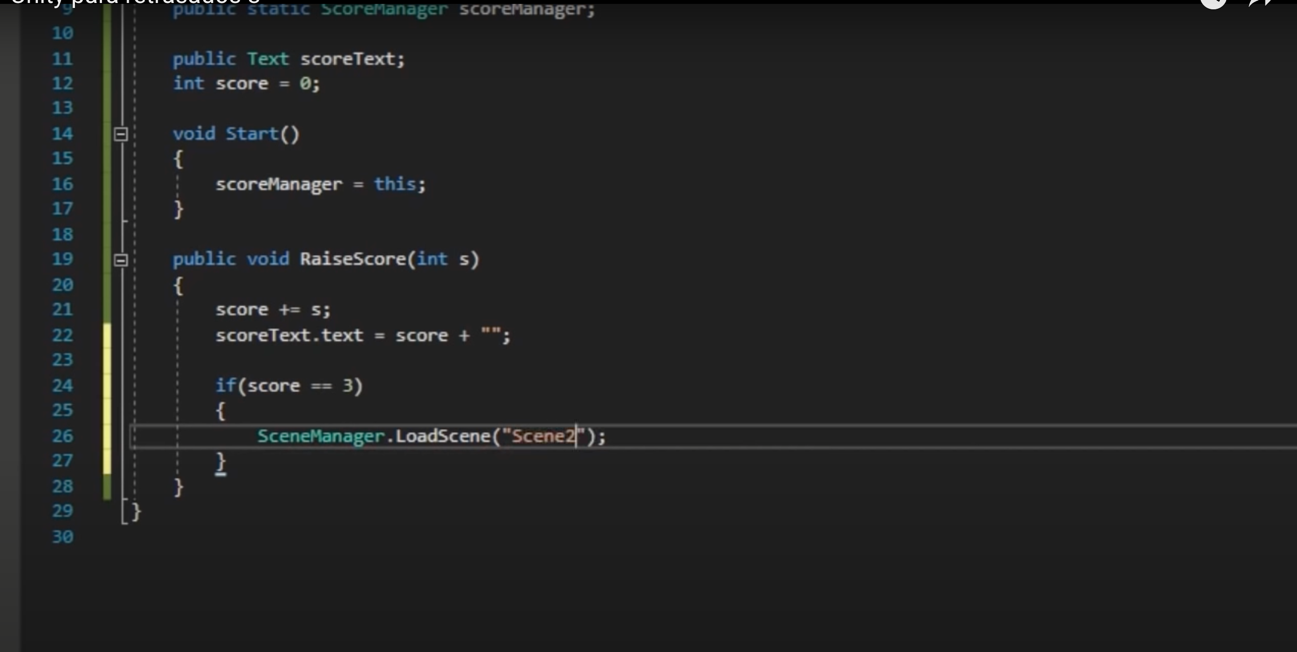
Luego añadimos la variable text y la sustituimos por el debug, accedemos a text y lo igualamos a nuestra variable score (para pasarla a string, ya que es un integer, este es un modo y mas abajo pongo otro mas sencillo y rápido.l

Para hacer un cambio de escena podríamos hacerlo en una escena completamente nueva, pero cuando hay muchos elementos que vamos a reutilizar en el cambio de escena se puede guardar escena como y ponerle otro nombre, por ejemplo, Scene2. Para hacer pruebas sencillas, solo cambiaremos el fondo, pero el problema ahora es que el fondo se podría renderizar por encima de otros objetos, porque es algo que decide Unity de forma impredecible si no lo configuramos con el sorting layer(capa de clasificacion) del renderer. Asi que creamos una nueva para el fondo y la ponemos por encima de la default para que se renderice por debajo

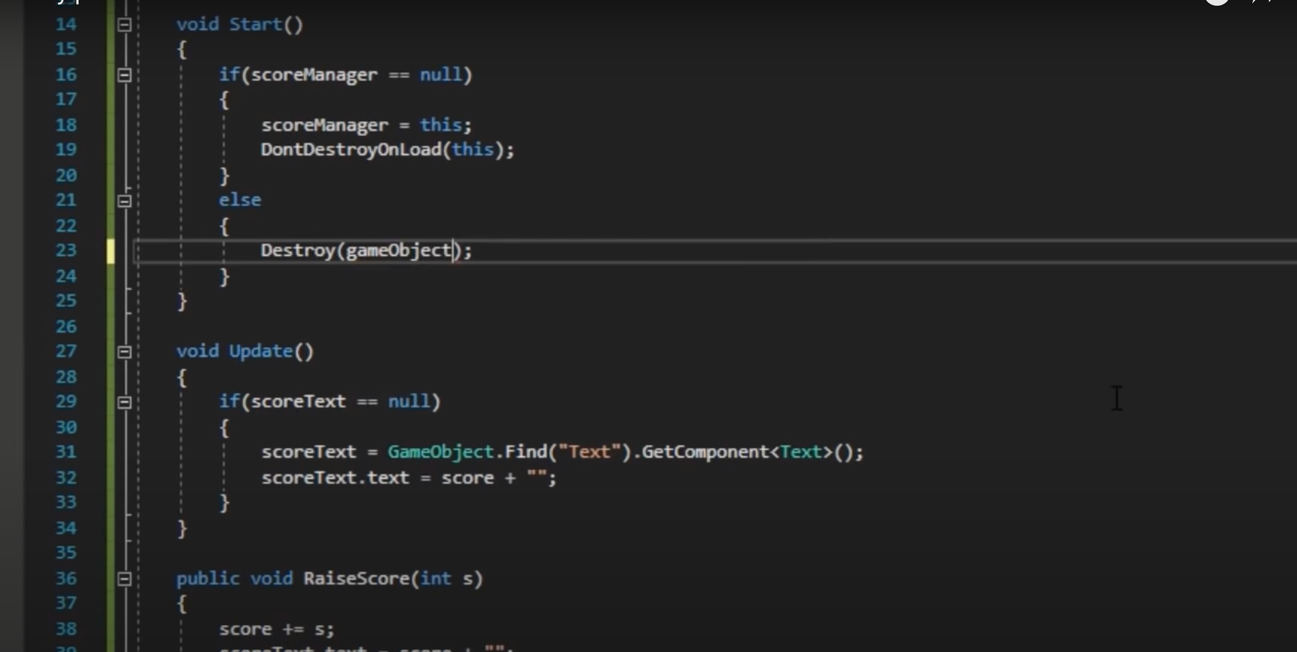
Ahora los demás sprites quedan en la capa default, si por cualquier motivo quisiéramos que la moneda renderice sobre el suelo por ejemplo, cambiaríamos el valor de order in layer a 1 para que se renderice por delante en la misma capa (default en este caso)

Ahora para añadirla en Unity, teniéndola abierta:

Vamos a hacer que cuando pase algún evento se cambie de escena, mediante código, por ejemplo que cuaando los puntos lleguen a 3. Lo primero es instalar otra colección using UnityEngine.LoadManagement; y poner las líneas de código para cargar la escena



\*\*Lo de rellenar la referencia estática consigo mismo (el script) habrá que hacer una comprobación para no tener errores en el cambio de escena como veremos mas adelante.



Para que en el cambio de escena no haya dos scoreManager ponemos en el start la condición de que si la referencia esta vacia le asigne un valor (mediante el script en el que esta esa línea que es el de RaiseScore) y que no lo destruya al cargar la escena si es el caso, y **si ya tiene un valor que se autodestruya** el objeto entero con el script para que no haya dos script iguales y genere problemas. \*\*Es lo que se llama singleton que sirve para decirle al motor, que no borre un elemento al pasar de una escena a otra y que si al crearse hay otra copia igual, se borre a si mismo para no interferir con la copia. El otro problema es que el texto scoreText se perderá cuando cambiemos la escena, por eso hemos creado la función update una búsqueda del GameObject Text y que coja el componente Text para que no se pierda el string de los datos si se cumple la condición, que en este caso es: si el valor está vacío.­