PEC2 - Un Juego de Plataformas

*Nota: A lo largo de la implementación de la PEC he tenido muchísimos problemas con la versión de Unity y Visual Studio. Este último no me detectaba los Packages y pese a investigar mucho por la red, he tenido que realizar la última parte de la práctica sin Debugger y sin notificaciones de error de texto. Además, mi ordenador tiene ya muchos años y la visualización del vídeo no es la mejor, ya que PlayMode de Unity va con muchísimo retraso.

*Nota2: La implementación de la práctica es un compendio de diferentes puntos de vista enfocados a un mismo objetivo. Aunque el código no es lo más limpio del mundo, la idea era mostrar distintas estrategias para lograr un mismo propósito; con sus pros y sus contras. Sobre todo, a la hora de detectar colisiones, se ha implementado de dos formas muy distintas el suelo, los bloques, y los enemigos.

En el caso de los objetos y el suelo, se ha utilizado Raycasting (que ha parecido ser la mejor opción). Para los enemigos, se ha creado un GameObject vacío, situado a los pies de Mario como child, cuya función principal es hacer de trigger y detectar la colisión con Goomba.

En la imagen podemos observar también un collider que actúa como trigger en la cabeza de Mario. Esta función quedó finalmente deshabilitada a causa de los artefactos que introducía y la intersección con el collider de los pies en momentos del juego (rompía bloques con los pies, en vez de con la cabeza).

- Cómo jugar?

El juego se conforma de 4 escenas diferentes: Pantalla de inicio, cutscene, pantalla de juego y pantalla final.

- Pantalla de inicio: Se muestra al inicio del juego o tras la pantalla final. Se pide al usuario pulsar la tecla Start para iniciar el juego.

- CutScene: Se muestra tras la pantalla inicial. Nos hace un resumen de las estadísticas generales del juego como la puntuación o el número de vidas que nos quedan. Si morimos, ya sea a manos de un enemigo o cayendo al vacío, nos saltará esta escena de juego mostrándonos las oportunidades que nos restan para terminar.

- Pantalla de juego: Compuesta por el mapa de juego principal. Dispone de todos los elementos como enemigos, bloques destructibles y no destructibles, obstáculos, etc. Es donde se lleva a cabo la acción principal del juego. Las mecánicas son muy sencillas:

    - Flechas del teclado para mover al jugador.
    - Barra espaciadora para saltar.

- Pantalla final: Se muestra tras perder todas las vidas o llegar al final del nivel. No es más que una adaptación de la Cutscene.

- Estructura del juego:

A continuación, se detallan los scripts implementados:

- NewGame: Sencillamente resetea las vidas a 3 para iniciar una nueva partida y llama a la escena CutScene tras pulsar la barra espaciadora.

- CutScene: Maneja las transiciones de escena tras perder una vida. Actualiza los datos a mostrar en los objetos del canvas cogiéndolos del script que maneja los datos guardados de la partida. Realiza una comparativa entre la puntuación obtenida durante la partida y la mayor puntuación que se ha logrado para mostrar en highScoreText la más alta.

  Muestra en lifesText el número de vidas que nos quedan y en scoreText la puntuación obtenida en la partida en cuestión. Por útlimo, guarda todos los datos y llama a la Coroutine que inicia la pantalla de juego tras dos segundos.

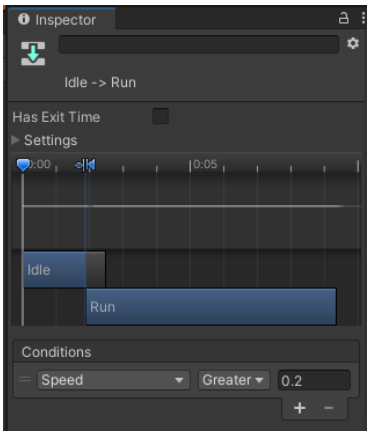
- PlayerMovement: Aquí es donde está prácticamente la totalidad de las funciones. Disponemos de los parámetros que controlan el movimiento de Mario, los parámetros que controlan el salto, los parámetros que determinan la física del juego y los diferentes layers y componentes.

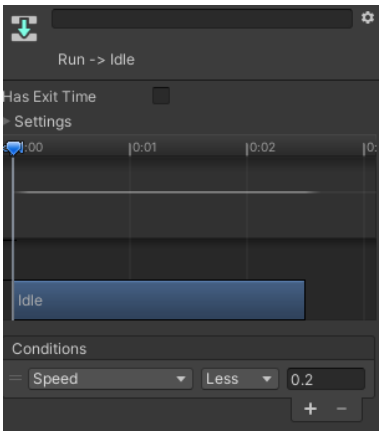
  Primero, inicializamos los valores del RigidBody de Mario y el offset de los Raycast que vamos a utilizar en el script. Hecho esto, en Update actualizamos la dirección a la cual se enfoca el player recogiendo el valor de las flechas izquierda y derecha y generamos los rayos que nos servirán para detectar colisiones con el suelo y los enemigos mediante groundLayer y enemyLayer.

Para realizar el seguimiento de la colisión con el suelo se usan tres variables: isOnGround, wasOnGround e isOnEnemy. Además, desactivamos la animación de salto si está en el aire y generamos un temporizador asociado al salto que genera una sensación más fluida entre pulsaciones de la tecla space.

En FixedUpdate tratamos todo lo correspondiente a las físicas para evitar artefactos. Aquí llamamos a PlayerMove(), ModifyPhysics(), Jump() y Raycasting().

- PlayerMove(): Recibe por parámetro la dirección inicializada en Start(). y añadimos una fuerza horizontal al Rigidbody de Mario para que se mueva hacia izquierda o derecha según la tecla pulsada. Al estar moviendo el personaje, la animación Run de Mario se activa en cuanto la variable speed supera cierto índice y se para cuando está por debajo.

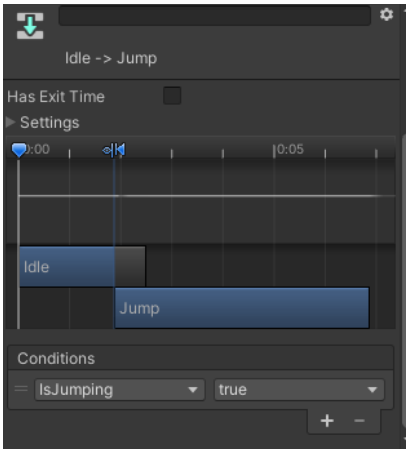




Por último, realiza un control de velocidad para que Mario no supere la velocidad máxima permitida y cambia la orientación del sprite en función de adónde mira el personaje.

- ModifyPhysics(): Modifica los parámetros del Rigidbody de Mario como el linear drag y la gravedad para hacer de la experiencia de juego algo más agradable. Nos permite saltar más alto cuando más rato mantengamos pulsada la tecla espaciadora, controla el desplazamiento lateral de Mario añadiendo fricción al cambiar de dirección y hace la gravedad más notable.

- Jump(): Añade una fuerza vertical y pone a true la variable IsJumping del Animator para que se muestre la animación de salto.



- Raycasting(): En esta función detectamos colisiones con los ladrillos y los bloques especiales. Generamos dos rayos hacia arriba para que cubran toda la amplitud de Mario con colliderOffset y entramos en los condicionales. En caso de que colisionemos con algo detectamos si se trata de un ladrillo o un bloque especial y realizamos las acciones correspondientes.

    - Brick: Si hemos cogido la seta y tenemos una vida extra, instanciamos las partículas de destrucción del ladrillo y desactivamos el renderer para dejar de verlo y el collider para anular futuras colisiones. Si no Mario es pequeño, llamamos a la función de balanceo del ladrillo que hará que se mueva y vuelva a su posición original.

    - LootBox: Sin entrar en la complejidad del lootbox, detectamos la colisión y en función de lo que la contenga la instancia en cuestión activaremos una animación u otra. Si contiene seta, activaremos el gameobject correspondiente; si contiene una moneda, activaremos el otro gameobject. Tenga el objeto que tenga, desactivaremos el sprite del lootbox y activaremos el del emptyblock para indicar que el bloque ha dejado de contener elementos.

  También gestionamos los sonidos que se lanzan cuando destruimos o colisionamos con bloques. Por último, aplicamos una fuerza vertical a Mario cuando este aplasta un enemigo.

- Además de estas funciones también disponemos de un OnCollisionEnter que detecta si colisionamos con Enemy o con Mushroom.

    - Colisión con Enemy: Si tenemos una vida extra y colisionamos con un enemigo, se nos quita la vida y Mario vuelve a ser pequeño. También cambiamos la longitud de los rayos del Raycast para adaptarlos al sprite de Mario pequeño. Si no tenemos vida extra, Mario pierde una vida y llamamos a la corrutina Died();

    - Colisión con Mushroom: Activa la animación de Mario grande y nos da una vida extra. Además, destruye el objeto con el que colisionamos, es decir, la seta desaparece.

  Para terminar, para calcular la longitud de los rayos se ha utilizado OnDrawGizmos(). Esto nos permite ver en el scene mode los rayos de color rojo y hasta donde llegan. En función de si Mario es grande o pequeño, los valores de rayLenght varian entre 1 y 0,7 flotante.

- PlayerUnit: Clase de la que heredan PlayerMovement y otros scripts. Contiene los audiosources y audioclips para gestionar los sonidos del juego. Además, lleva la cuenta de las vidas del jugador y contiene la corrutina Died(), usada para la muerte de Mario. En Update(), si Mario cae por debajo de -30 llamamos a la corrutina. Esta gestiona la memoria que almacena la cantidad de vidas, activa la animación de muerte y pausa el juego. A los 3 segundos, si aún nos quedan vidas cargamos la escena Cutscene; si estamos a 0, carga GameOver. Por último, restaura el juego y pone Time.timeScale a 1.

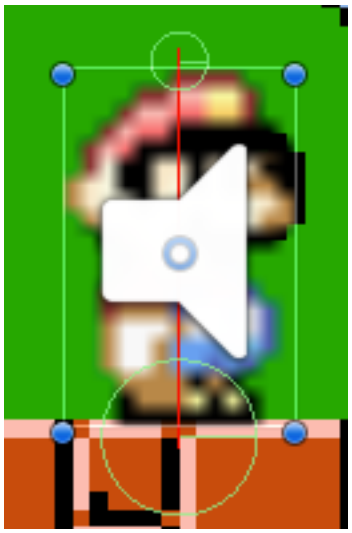
- CameraSystem: Gestiona el movimiento de la cámara y el seguimiento a Mario, modificando su posición. Limita la visión en horizontal y vertical de Mario fijando boundaries. Esto hace que no podamos salir del mapa en ningún momento; ni por la izquierda ni por arriba.

- Brick: Contiene la corutina Bounce que se encarga de hacer subir el bloque hasta cierta posición y, al llegar arriba del otro, baja hasta la posición original. Para una sensación más fluida de movimiento, se ha utilizado Time.deltaTime para no depender del frame rate.

- LootBrick: Hereda de Brick para poder usar la corutina también con los bloques especiales. Contiene todos los gameobjects que son hijos del principal para activarlos y desactivarlos en su debido momento. De esta forma gestionamos las animaciones de las monedas y los powerups mediante SetActive(). También dispone de dos booleanos que determinan qué objeto es el que contiene el bloque.

- EnemySpeed: Detecta colisiones con los objetos del mapa para cambiar de dirección siempre y cuando tengan el tag Brick o Ground (para evitar artefactos y cambios de dirección inesperados). A diferencia de las colisiones con el suelo y los bloques, la colisión de Mario con los enemigos se ha gestionado con OnTriggerEnter2D.

  En este caso, si el enemigo hace trigger con los pies de Mario, lanza la animación de muerte de Goomba, cambia el tag del enemigo a “Untagged” y se destruye a si mismo al cabo de medio segundo. Por último, nos suma la puntuación correspondiente tras matarlo. En la siguiente imagen podemos ver el collider de los pies de Mario y el raycast.



Esta ha sido una parte de la PEC “complicada”. Finalmente se ha decidido dejarlo así para mostrar los pros y los contras de una y otra versión. En este caso, usar un gameobject vacío (hijo de Mario) que haga trigger con el enemigo ha sido más engorroso que detectar la colisión mediante Raycast. Aun así, se ha mantenido en el código para mostrar que hay más de una posibilidad a la hora de detectar colisiones.

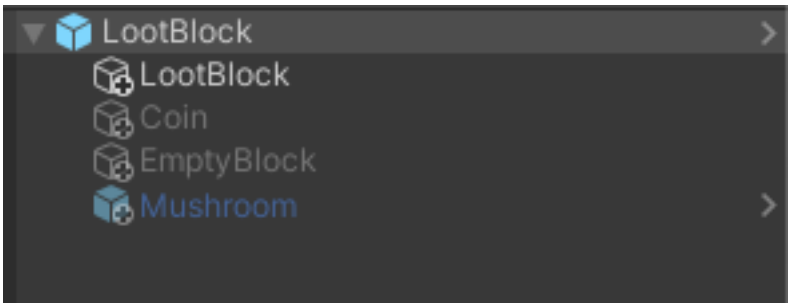
- MushroomSpeed: Esta clase funciona exactamente igual que EnemySpeed, salvo el elemento de trigger; que evita cambios de direcciones inesperados al colisionar con el suelo o los ladrillos.

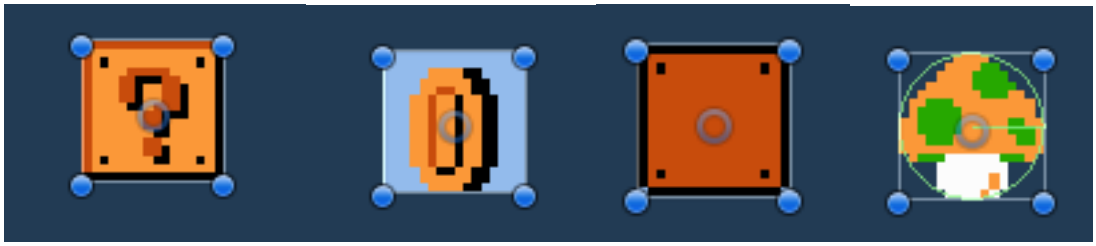
- ScoreSystem: Guarda las variables que gestionan el temporizador y la puntuación. En esta clase también está “FinishFlag”, que marca (como trigger) cuando hemos cruzado la línea de meta. Básicamente, muestra por pantalla el valor de la puntuación actual de la partida y el tiempo que resta para llegar al final. Si no completamos el nivel y se acaba el tiempo, llamamos a la corrutina Died(). En el caso de completar el nivel satisfactoriamente, llamamos a TimeScoreSum() y llamamos a la corrutina FinishLevel(). FinishLevel() lanza la animación de victoria de Mario, espera 6 segundos y cambia a la escena GameOver.

- GameOver: Pantalla final que muestra contenido durante 6 segundo y carga la escena principal del juego.

- DataManager: Almacena en memoria todos los valores de la vida y la puntuación del personaje. Se trata de un objeto empty que no se destruye tras el cambio de escena mediante el método DontDestroyOnLoad(). E método Savedata() genera un nuevo file y guarda los valores de la partida en la clase gameData. Por el contrario, LoadData() carga los datos almacenados durante el juego. Esta clase nos permite almacenar y recuperar valores entre escenas de forma alternativa a PlayerPrefs.

En cuanto a los gameObjects a destacar, entre ellos está el LootBlock:



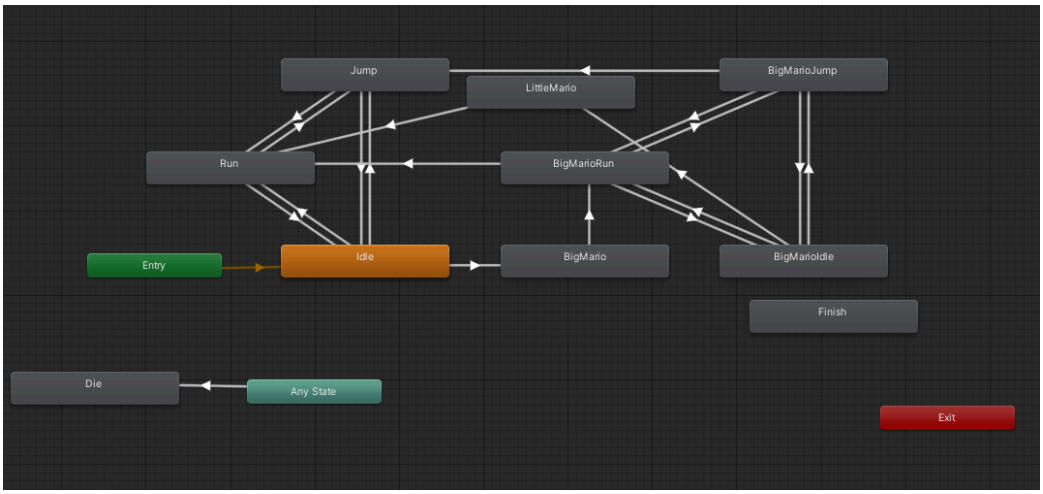


LootBlock incluye varias capas que esconden los objetos que hay detrás. La primera capa de todas, que lleva asociada la animación de la figura posterior, es el LootBlock inicial. Cuando lo golpeamos, dependiendo de si tiene moneda o tiene mushroom, lanza la segunda o la cuarta animación. Al mismo tiempo, pasa a desactivarse LootBlock para activarse EmptyBlock.

Por otro lado, todo el mapa está hecho usando Tilemaps. Esto me causó ciertos problemas con los colliders, puesto que todos los objetos estaban asociados a la capa ground (tanto tuberías como bloques) para detectar colisión con el suelo y permitir el salto. Además, ya que usar composite colliders no detectaba bien las colisiones con los objetos como la seta o el enemigo, se han puesto en una capa separada que en vez de usar composite collider, usan su propio boxcollider asociado al sprite.

- Cosas a mejorar

    - Animaciones de Mario. Hay partes mejorables, como la transformación de grande a pequeño. Obtenía un mensaje de error al pasar de AnyState a Big o Little Mario que ignoraba la animación. Finalmente, la animación de pequeño a grande sólo ocurre del estado Idle a Big Mario y la animación a pequeño se hace sin transición.



- El paso de datos de la puntuación obtenida y el guardado del highscore no están del todo correctos. Esta fase final fue cuando tuve problemas con los Packages y me fue difícil debuggar para hacer un buen seguimiento de las variables y terminar de ajustar mejor el paso de parámetros.

- La colisión con el enemigo, debido al uso de triggers en vez de Raycasting, entorpeció el script que gestiona el salto de Mario tras pisar un Goomba. Finalmente, se ha quedado en un híbrido entre ambas para lograr un resultado, aunque no ótpimo, aceptable. Desactivar el collider de Goomba tras ser pisado generaba un efecto raro en la jugabilidad y, como consecuencia, después del salto hacia arriba, podemos volver a pisarlo de nuevo y la puntuación se duplica. Con algo de tiempo se podría haber perfeccionado la colisión.

- La mezcla entre gameobjects y tilemaps hizo el proyecto un poco confuso en un principio. La intención era usar TileMaps animados. De hecho, el package asociado a estos está incorporado en el proyecto. Al tener implementados los scripts para los gameobjects y toda la trazabilidad hecha entre métodos y variables, fue difícil de adaptar para poder realizar de forma completa y ordenada, todo el proyecto con TileMaps.

- Recursos: Todos los sprites utilizados durante el juego han sido extraídos de SpritersResource (https://www.spriters-resource.com/). Aquí los escogidos: