Práctica Unity Motores

Roselló Morell, Sergio
sergio.rosello@live.u-tad.com

April 21, 2018

Contents

1 Inspiración e idea generál

El juego en el que me he decidido inspirar es: 1942, un juego desarrollado por Capcom para máquinas recreativas en 1984. Mi juego debería ser capaz de tener un loop completo, en el que fuese posible jugar como en las máquinas recreativas. El jugador solamente va a jugar en una escena con scroll semi-infinito. La máxima puntuación se mantiene entre partidas, de forma que un jugador puede ver si ha superado su máximo o no. Al final de las oleadas de aviones pequeñas, va a llegar la oleada del boss, en la que simplemente aparecerá el boss en la pantalla contra el jugador.

2 Estructura de código

El código está planteado de la forma más sencilla posible, en la que solamente existe lo esencial para que funcione. A continuación explicaré los distintos scripts usados y por qué los he planteado así.

3 Controladores de movimiento y comportamiento

3.1 BehaviourController

Este script es la base de cualquier actor, ya sea enemigo o personaje principal. Se encarga de actualizar la velocidad del objeto, gestionar su vida y adquirir los componentes necesarios para el correcto funcionamiento de las clases hijas (Enemigos y Actor principal).

3.1.1 Enemigos

Los enemigos adquieren el comportamiento de BahaviourController. Una opción a valorar era que el enemigo básico (BasicEnemyController) tuviese los comportamientos añadidos a BehaviourController para los enemigos y que los enemigos más complejos extendiesen de este pero finalmente decidí que, al ser enemigos básicos no tenía sentido hacerlo por tanto, cada enemigo hereda directamente de BehaviourController, luego ya añado las características de cada enemigo.
Actualmente tengo dos enemigos:

- **BasicEnemyController**
  Aparece de arriba de la pantalla y llega hasta abajo para dar la vuelta y salir por donde ha aparecido.
- **SteerEnemy**
  Aparece en una posición aleatoria de los bordes de la pantalla y hace un circulo.
- **BasicBossEnemyController**
  Funcionamiento muy similar a BasicEnemyontroller, de hecho, hereda de

el. Este script simplemente se encarga de disparar y de activar el sonido cada vez que se dispara.

Todos los enemigos son destruidos si salen de la visión de la cámara.

3.1.2 Actor principal

Este script se encarga de la gestión del avión de jugador.

- **Mantener al jugador entre los límites de la pantalla**
  Gestiona que el actor no pueda salir del mapa, usando Screen.width y Screen.height calculo el aspect ratio de la pantalla, para averiguar el tamaño de la pantalla en X. Una vez hecho esto, me aseguro de que el actor se quede entre esas coordenadas. El método se llama keepPlayerWithinBounds()
- **Gestión del movimiento**
  Usa el método DetermineDirection para asignar los axis sobre los que controlar el jugador. Este método se ejecuta al llamar a base.Update en su propio update. Actualiza la variable velocity con la nueva velocidad cada frame.
- **Gestión de los disparos**
  Los disparos usan un temporizador para poder regular la cadencia de disparo.
- **Gestión de la animación**
  Asigna los booleanos necesarios para gestionar las animaciones del jugador.
- **Gestión de la vida**
  Cuando el BoxCollider2D colisiona con cualquier objeto, se le reduce la vida al jugador, si colisiona con un enemigo, le quita la vida al enemigo también. Si solo tiene una vida, el enemigo muere.
- **Gestión de la puntuación** En el OnDestroy() guarda la puntuación "Highscore" en memoria para la próxima partida.

4 Scripts de propósito específico

4.1 Controlador de Mapa

Este script simplemente desplaza el mapa de fondo a una velocidad gestionada por el usuario de forma continua.

4.2 Gestión de vida

Esta script gestiona la vida de cada gameobject al que se le añade. Tiene un método IsAlive() que uso para saber si debería ser destruido un GameObject. Tiene funcionalidad de animator ya hecha aunque no lo he usado.

4.3 Intermitent

Gestiona con una corrutina la aparición y desaparición del cartel "Insert coin" del menú de inicio.

4.4 layers

Script que facilita la implementación de layers en el código. Al conocer posteriormente la matriz de colisiones 2D del editor de Unity, no lo he usado den todos lados, sino que solamente en las partes necesarias.

4.5 Main Menu

Simplemente carga el nivel 1 si se ha presionado una tecla.

4.6 Pause Menu

Controla el menú de pausa. Puede llevar al jugador al menú de inicio, salir del juego o resumir el juego.

4.7 Balas

Si colisionan con los enemigos, les quitan una vida y se eliminan las balas. Si no colisionan con nada, se eliminan a los 5 segundos.

4.8 PoolManager

Uso este script para hacer que los enemigos aparezcan en el mapa. Es un singleton de forma que solo puede haber uno a la vez mientras el juego está en ejecución. Cuando quiero añadir un gameObject mediante el PoolManager, llamo a sus métodos load y spawn. En realidad no sería necesario el PoolManager para hacer este juego. Además, creo que se podría hacer mejor de lo que lo estoy haciendo.

4.9 SceneLoader

Este es un script que empecé a hacer para facilitar el cambio de escenas en el juego pero no he acabado de retocarlo, por eso solo lo uso en la escena de la muerte. Lo que quería hacer era gestionar el cambio de escenas directamente desde el editor. Esto si lo consigo con el script, por tanto, lo he dejado ahí.

4.10 ScoreManager

Mantiene actualizados los campos de puntuación. Si el jugador está batiendo el record de puntuación, esta se actualiza con la puntuación de la partida. Si el juego se ha ejecutado en la máquina antes, ha guardado una puntuación máxima, que recoge y usa para las próximas partidas.

4.11 Singleton

Una clase para implementar el patrón de diseño Singleton. Sirve para que solo pueda existir una instancia de un objeto a la vez.

4.12 WaveManager

Este script gestiona las oleadas de los enemigos. Los dos enemigos más básicos (Steer y Basic) están controlados por probabilidades de aleatoriedad. Si el número aleatorio es menor al indicado por el usuario, se instancia un enemigo. El boss necesita que pase cierto tiempo, ya que es demasiado complicado batirlo tantas veces. El juego está diseñado para ser un solo nivel y que el jugador juegue hasta que muera.

5 Editor

Durante la realización de la práctica, he tenido que hacer tanto scripts como acciones mediante el editor.

5.1 Organización de carpetas

Las carpetas están organizadas según tipos de datos.

5.2 Layer collision matrix

Esta matriz indica al editor que layers van a poder colisionar. De esta forma, no tengo que detectarlo mediante código. Esto hace que el código sea bastante mas sencillo.

5.3 Animator

El animator del player decide a que estado de animación se va a dirigir la animación.

5.4 audio

Contiene los audios de todo el juego. Para que un audio se reproduzca, se necesita un audioListener en la escena, un audioSource. He puesto una canción en cada escena del juego y efectos de sonido para el jugador y para el boss.

5.5 Escenas

En este juego existen 3 escenas. Una de inicio del juego, otra del nivel, en la que juega el jugador y otra para la muerte. Todas estas escenas tienen interacción entre si de forma que puedes pasar de una escena a la otra sin reiniciar el juego. Básicamente como funcionaría un juego de verdad.

6 Conclusión

Durante el desarrollo del juego, he aprendido la API de Unity y creo que conozco el motor lo suficientemente bien como para continuar aprendiendo de forma autodidacta.