Bomber Crab

Maribel Díaz Galiano 4º IDCD A

ÍNDICE

Descripción general

El juego se desarrolla en un ambiente de playa. El jugador controla el movimiento de un cangrejo que se puede mover en horizontal, vertical o diagonal usando las teclas de dirección. El cangrejo aparece siempre en la parte inferior de la pantalla. Aparecen enemigos que se mueven hacia la parte inferior de la misma. El jugador solo puede atacar acercándose a los enemigos, no puede disparar objetos a distancia para atacar. Tiene tres vidas. Cuando el jugador muere, se acaba el juego. Hay tres niveles, a cada cual la agresividad de los enemigos aumenta. Para pasar de un nivel a otro hay que alcanzar una puntuación mínima. En el último nivel hay un jefe final. El objetivo es derrotarlo.

Controles

Movimiento - Teclas de dirección

Ataque – Ctrl

Menú pausa – Barra espaciadora

Elementos desarrollados

Jugador principal

Cangrejo

Enemigos

Se van creando en segundo plano y van apareciendo en el área de juego. Conforme va aumentando el nivel, van aumentando en cantidad y agresividad.

Gaviota: nada especial.

Estrella de mar: gira.

Erizo de mar: conforme pasa un tiempo aleatorio se pone rojo y acabado un tiempo aleatorio desaparece. Tiene un radio de visión, como un radar, si el jugador se acerca a la zona verde (ver *Detalles de implementación*), lee persigue (y lee deja de perseguir si se sale de esa zona). Si entra en la zona roja se pueden atacar entre sí (en la zona verde no).

Pulpo: jefe final. Es grande y para matarlo tienes que quitarle una determinada cantidad de golpes. Lanza objetos en la dirección del jugador.

Objetos

Coco: caen en la arena desde las palmeras. Es un coleccionable que proporciona una vida extra.

Bullet: lo lanza el jefe final. Si alcanzan al jugador, le resta una vida.

Fondo

Playa

Extras

Palmeras: de vez en cuando sueltan un coco.

Nubes: pasan por el escenario como decoración.

Detalles de implementación

Escena 0 (Menú principal)

Main Camera. La cámara principal.

Canvas. El menú principal está compuesto por un Canvas. Se usa un Panel para establecer una imagen de fondo. Se añade un botón que, cuando se pulsa, carga la primera escena. Todas las escenas que se van a usar en el juego se han cargado previamente a través de *Edit > Build Settings*.

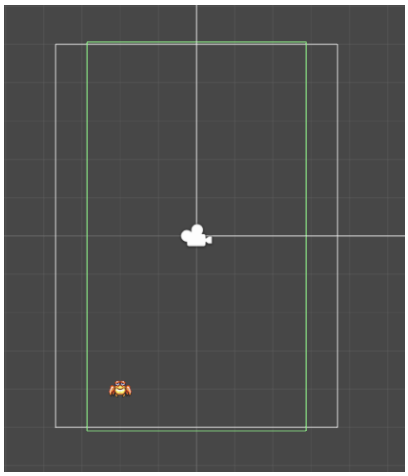
Scene Management. Es un objeto que se mantiene activo durante todo el juego, aunque se cambie de escena. Se encarga de la carga de escenas cuando es necesario, en este caso, cuando el jugador ha obtenido la puntuación necesaria para ello y de reproducir la música asignada a cada nivel. Contiene un componente `Script` con el mismo nombre `Scene Management`. Gracias al método `DontDestroyOnLoad(),` el objeto `Scene Management` no se destruye entre cambio y cambio de escena. Cuando se pulsa el botón *Start Game* del Canvas, carga la primera escena. Para comprobar si el jugador ha alcanzado la puntuación necesaria para cambiar de nivel, accede a la variable estática `Score` almacenada en la clase `PersistentData` (que contiene otros valores globales del juego, por ejemplo, si cada uno de los niveles se ha cargado ya o no, si el jugador está vivo o no o el número de vidas restantes). Al script hay que pasarle los siguientes parámetros: la puntuación necesaria para pasar al

segundo nivel, la puntuación necesaria para pasar al tercer nivel, la música de fondo del primer nivel, la música de fondo del tercer nivel y la música de la pantalla de victoria. Además, tiene un componente `AudioSource` con la música que suena en cada una de las escenas.

Escena 1 (Primer nivel)

Main Camera. La cámara principal.

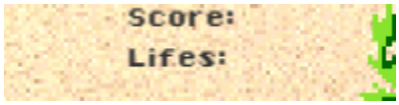
Player. El jugador principal. Tiene un `SpriteRenderer` con la imagen del personaje, un `RigidBody2D` para darle velocidad cuando nos movemos, un `BoxCollider2D` para detectar colisiones con enemigos y objetos, un `Animator` con el árbol de animaciones y un `Script` `PlayerMovement`. Este script comprueba continuamente si se han presionado las siguientes teclas: las teclas de dirección que proporcionan movimiento al jugador y la tecla Ctrl izquierdo para atacar. Hay que pasarle como parámetro la velocidad a la que queremos que se mueva el jugador.



BoundaryDestroy. Delimita el área de juego a través de su collider. Tiene asociado un script `Destroy Exiting Objects` que detecta cuándo un objeto sale de su collider (`OnTriggerExit2D()`) y lo destruye si NO es el jugador, identificado por la etiqueta `Player`. Es decir, cualquir objeto no etiquetado como `Player` que salga de su collider, es destruído. He usado esto para no acumular enemigos en la escena a medida que se van creando. Había pensado en utilizar un pool, pero quizá era demasiado para un juego tan simple. Por otra parte, al igual que el objeto `SceneManagement`, `BoundaryDestroy` también se mantiene activo entre escenas.

GameController. "Spawnea" los enemigos y los cocos que caen de las palemeras, con las propiedades especificadas. Tiene un script asociado `GameController` que se encarga de lo mencionado. Para crear las oleadas de enemigos, utiliza una corrutina, que comienza a ejecutarse cuando se carga la escena y seguirá ejecutándose en segundo plano. Lo único que hace esta corrutina es esperar los segundos especificados e ir instanciando enemigos en una posición aleatoria en la parte superior de la pantalla y asignándole una velocidad vertical hacia abajo (a modo *Space Invaders*). Los enemigos se agrupan dentro de un `GameObject` padre para mantener ordenada la escena. Los cocos también se "spawnean" por medio de una corrutina en una posición donde haya una palmera. Las palmeras se han establecido previamente a mano en el editor. El script accede al objeto padre de todos los objetos palmera, escoge uno al azar y le asigna su posición a la posición del coco.

Canvas. Contiene un texto con la puntuación y vida actuales del jugador. Como la puntuación es algo que se mantiene entre escenas, el canvas también debe hacerlo. Para esto tiene un script `Update Canvas` cuyas funciones son llamar a los métodos `DontDestroyOnLoad()` en su `Start()` y `UpdateScore()` en su `Update()` para mantener los datos actualizados.



Escena 2 (Segundo nivel)

Se pretende que la dificultad sea mayor la del nivel 1. Para ello, hay un enemigo nuevo, *Urchin*. Este no se conforma solo con caer, sino que, si el jugador entra en su radar, es perseguido.

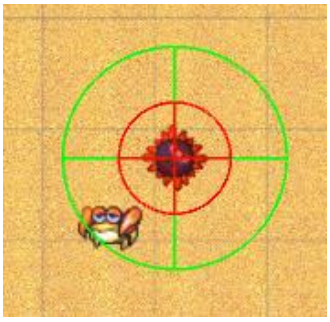
Main Camera. La cámara principal.

Player. El mismo que en la escena 1. Se crea uno nuevo cuando se cambia de escena.

GameController. El mismo que en la escena 1. Se crea uno nuevo cuando se cambia de escena. La razón es por si queremos cambiar algún aspecto del tablero y los enemigos que se crean. En este caso, es sobre todo por la segunda razón.

Urchin Enemy. Contiene un script que establece el radar. Hay dos fronteras: una (la verde) a partir de la cual el enemigo detecta al jugador y entonces le persigue, pero no le puede hacer daño (ni el jugador al enemigo) y otra (la roja) a partir de la cual el enemigo le persigue y le puede hacer daño (y el jugador al jugador). La detección del jugador se hace con `Physics2D.OverlapCircle()`. Además, se usa el paquete `iTween` para dar el efecto de cambiar el color de forma intermitente. Para establecer el color, compruebo si la distancia al jugador es menor que una distancia especificada. Se utiliza el método `OnDrawGizmos()` para visualizar el radar en la pestaña de escena.

Jugador fuera del alcance del enemigo



Jugador dentro del alcance del enemigo (zona verde)

Escena 3 (Escena final)

Se pretende que la dificultad sea mayor que en nivel 2. Para ello, hay un enemigo nuevo, `Final Boss`. Este es de un tamaño mayor (más difícil de esquivar) y lanza objetos en la dirección del jugador. Para matarlo, hacen falta más golpes de lo normal.

Main Camera. La cámara principal.

Player. El mismo que en los anteriores niveles.

GameController. El mismo que en los anteriores niveles.

SeaShore. El fondo del juego. Tiene un `SpriteRenderer` con la imagen que queremos que tenga y un Animator con la animación.

FinalBoss. El jefe final. Tiene un `SpriteRenderer` con la imagen que queremos que tenga, un `Animator` con la animación, un `BoxCollider2D` para detectar colisiones tanto con el borde del área de juego (que tienen la etiqueta `Boundary`) como con el jugador. Tiene también un script `Destroy By Contact`: si estás chocando con un enemigo y estás atacando (la tecla Ctrl está siendo pulsada), el enemigo muere y desaparece. Si chocas con un enemigo y NO estás atacando, pierdes una vida. Cuando pierdes el número máximo de vidas, mueres y vuelves al menú principal del juego. Tiene un `Weapon Script` que se encarga de gestionar el ataque del boss.

Escena 4 (Escena GameOver)

Contiene un canvas con una imagen de fondo. Se carga cuando el jugador muere. Se barajó la opción de hacer que cuando el jugador muriera se volviera al menú inicial, pero había problemas con la duplicación de objetos `DontDestroyOnLoad()` y se decidió que, simplemente, acabara el juego.

Escena 5 (Escena You Win!)

Contiene un canvas con una imagen de fondo. Se carga cuando el jugador derrota al boss.

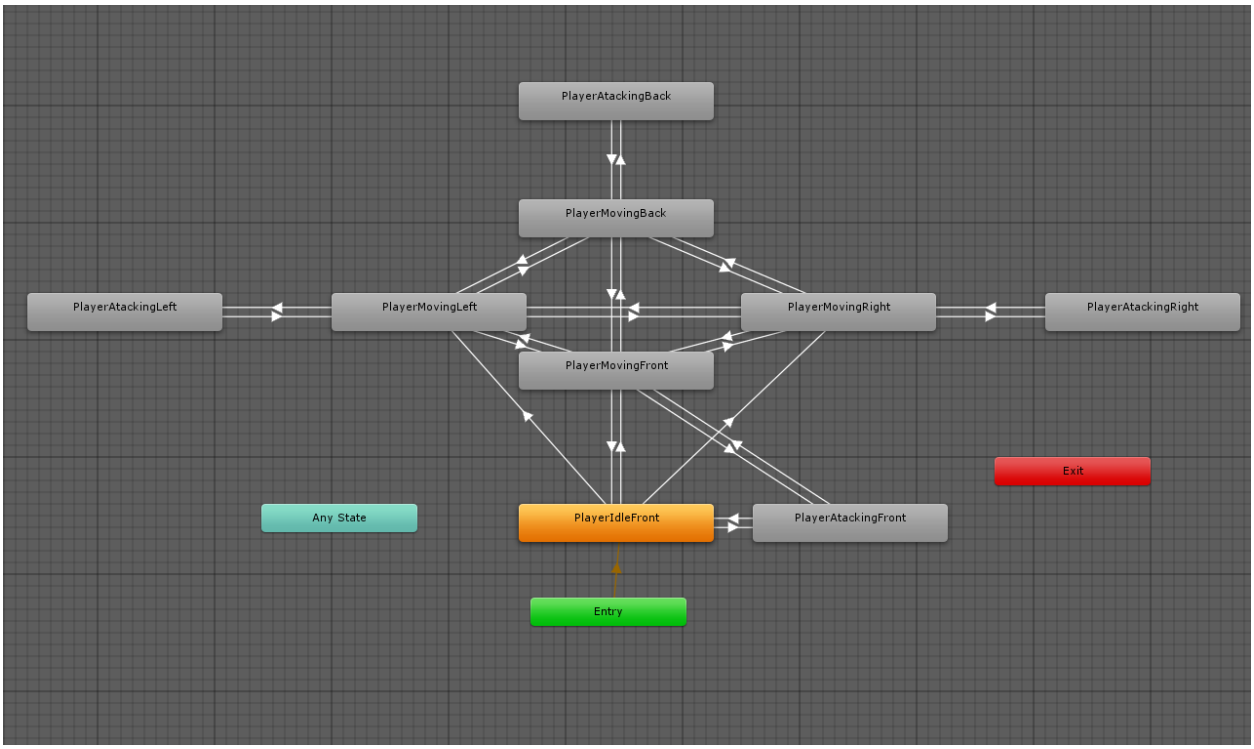
Menú de pausa

Un canvas deshabilitado hasta que presionas la tecla espacio. Entonces se habilita y aparece un menú de pausa con un botón para continuar jugando. Para pausar el juego pone `Time.timeScale` a 0, y para continuar jugando pone `Time.timeScale` a 1 de nuevo. El objeto existe durante todo el juego.

Detalles de otros objetos

Árbol de animaciones. El jugador tiene animaciones para cuando se mueve hacia arriba, abajo, izquierda o derecha. Además, también tiene para cuando está parado mirando hacia arriba, abajo,

izquierda o derecha. También tiene para cuando está atacando hacia arriba, abajo, izquierda o derecha. A continuación, una captura del árbol de animaciones:



Bullet (Prefab). Tiene un `SpriteRenderer` para que tenga la imagen que queramos, un `RigidBody2D` cinético para darle velocidad. Además, tiene un script `Destroy After Time` que destruye el objeto tras un tiempo especificado y un script `Destroy By Contact`, descrito en el objeto `Final Boss` de la Escena 3.

RandomRotation (Script). Hace que el enemigo Estrella de Mar gire aleatoriamente.

Layers (Script). Contiene un mapeo de las capas existentes a número. Para trabajar con ellas con más comodidad en código.

BackgroundScrolling (Script). Hay un *parallax scrolling* entre el fondo de la arena, el jugador y los enemigos, las palmeras y las nubes. Los primeros objetos se mueven más lentamente porque están más "lejos" del jugador mientras que las últimas lo hacen más rápido porque están más "cerca" del jugador. Lo que hace es darle movimiento al objeto fondo modificando su posición sobre el vector `Up` a una determinada velocidad. La clave está en que al comienzo del movimiento guarda la posición desde la que partió, y cuando ha avanzado una distancia determinada (normalmente la longitud en Y del fondo, de forma que todo el fondo haya recorrido toda la pantalla), vuelva a la posición de inicio y comience de nuevo el mismo recorrido anterior.