CONTENIDOS

1. Tweens

En este apartado veremos cómo realizar los Tweens, que son un tipo de animaciones de gran utilidad a la hora de agregar detalles a nuestros videojuegos. Internamente, los Tweens realizan una interpolación, permitiendo crear frames intermedios a dos cuadros proporcionados de una animación. Desde otro punto de vista, los Tweens también pueden verse como comportamientos personalizados aplicados a actores.

Para realizar el Tweening, debemos especificar, entre otras cuestiones, el estado final de una propiedad del objeto (tomaremos como estado inicial el actual) y el tiempo que debe durar la transición y luego Phaser calculará de forma automática los valores o estados intermedios.

Para la gestión de los Tweens vamos a utilizar la clase **tween**, cuya documentación oficial se encuentra disponible en:

docs.phaser.io/Phaser.Tween.html

1.1 Tweens individuales

Como mencionamos anteriormente, se puede aplicar un Tween a un actor. Es posible modificar cualquier propiedad asociada al actor, tal como su posición, su nivel de opacidad alpha, su escala y su ángulo.

Por ejemplo, para realizar el movimiento horizontal de un Sprite, podemos alterar la propiedad *x* del mismo de la siguiente manera:

```
image = game.add.sprite(0, game.world.centerY, 'cara');

// estado inicial de la propiedad
image.x = image.width/2;

// creamos el tween que se aplicara al objeto image
var behavior = game.add.tween(image);

// to se utiliza para configurar el tween
//los parámetros de la funcion to son:
// to(properties, duration, ease, autoStart, delay, repeat, yoyo)

behavior.to({ x: game.world.width-(image.width/2) }, 2000);

// inicia el tween
behavior.start();
```

Más en detalle, primeramente se crea un objeto tween en la variable *behavior* que se aplicará al sprite llamado *image*. Lo que deseamos es alterar la posición horizontal del sprite, y para ello utilizamos el método ***to*** pasandole como primer argumento la posición final y después el tiempo en el cuál se realizará el desplazamiento del sprite. Por último, con **start()** damos inicio al tween.

El método ***to*** de la clase **tween** posee los siguientes argumentos:

Properties: indica la propiedad a interpolar.

Duration: la duración del tween en ms (default 1000).

Ease: tipo de función *Easing Function* usada para interpolar (default Phaser.Easing.Linear.None).

AutoStart: booleano que especifica si el tween iniciará automaticamente o no (default false).

Delay: tiempo antes que inicie el tween (default 0).

Repeat: valor numérico que indica la cantidad de veces que el tween debe reiniciarse una vez completado el ciclo inicial. Es ignorado por tweens encadenados (default 0).

Yoyo: booleano que indica si el tween volverá al estado anterior en modo ping pong (elimina callback de onComplete) (default false).

De manera similar al primer ejemplo, también podemos modificar la opacidad alpha de un actor, que varía entre 0 y 1 en 2 segundos. El *autoStart* está en activo por lo que inicia automáticamente. Aquí se aplicó además el efecto denimonado *yoyo* y con repetición de mil iteraciones del tween. El efecto *yoyo* realiza un espejado deo Tween hasta volver al estado inicial, dando un sensación ida y vuelta.

```
function create() {
  . . .

  // estado inicial de alpha
  sprite.alpha = 0;

  // creamos el tween que se aplicara al objeto image
  // to(properties, duration, ease, autoStart, delay, repeat, yoyo)

game.add.tween(sprite).to( { alpha: 1 }, 2000,
Phaser.Easing.Linear.None, true, 0, 1000, true);

}
```

Para interpolar el ángulo y la escala se deberá proceder de manera análoga.

En el ejemplo **00_tween_interpoladores** que se encuentra subido se puede observar el efecto producido en la posición de un sprite al aplicar distintos valores en la función *Easing Function*, la cual expresa con qué aceleración se realizará la alteración de la propiedad a modificar (que en este caso es la posición x del sprite). La función *Easing Function* puede tomar los siguientes valores:

- Phaser.Easing.Back
- Phaser.Easing.Bounce
- Phaser.Easing.Circular
- Phaser.Easing.Cubic
- Phaser.Easing.Elastic
- Phaser.Easing.Exponential
- Phaser.Easing.Linear
- Phaser.Easing.Quadratic
- Phaser.Easing.Quartic
- Phaser.Easing.Quintic
- Phaser.Easing.Sinusoidal

En los ejemplos **00_tween**, **00_tween_alpha**, **00_tween_posicion** y **00_tween_interpoladores** se puede observar en detalle las implementaciones de tweens individuales.

1.2 Multiples Tweens

En esta sección veremos que un actor puede tener cualquier cantidad de Tweens, definidos de forma simultánea o secuencial.

1.2.1 Múltiples Tweens simultáneos

Phaser posibilita que varios Tweens se apliquen de forma simultánea a un actor.

Por lo tanto, es factible que por ejemplo un objeto cambie su nivel de opacidad alpha mientras se desplaza. En el siguiente código (**01_tween_simultaneo**) se encuentra implementada esta funcionalidad:

```
// creamos el tween que se aplicara al objeto image
var behavior = game.add.tween(image);



// sprite con 2 tweens simultaneous

behavior.to({ x: 700, alpha: 1 }, 3000, Phaser.Easing.Linear.None,
true, 0, 1, true);
```

La configuración del Tween anterior hará que el actor se desplace hasta 700 en el eje *x* y su nivel de opacidad alpha llegue a 1, ambas cosas en 3 segundos. El *autoStart* está en activo por lo que inicia automáticamente. El Tween tendrá una repetición y aquí también se aplicó el efecto yoyo.

1.2.2 Multiples tweens secuenciales

Los tweens se pueden concatenar para obtener distintos efectos.

En el siguiente ejemplo (**01_tween**) el actor se define con cuatro Tweens concatenados en loop, que permiten desplazar el sprite en un circuito predefinido.

```
var behavior = game.add.tween(image);

// concatenamos 4 tweens, y los fijamos en loop

behavior.to({ x: 700 }, 3000, Phaser.Easing.Linear.None)
        .to({ y: 400 }, 1000, Phaser.Easing.Linear.None)
        .to({ x: 100 }, 1000, Phaser.Easing.Linear.None)
        .to({ y: 200 }, 1000, Phaser.Easing.Linear.None)
        .loop() // hace loop de una cadena de tweens
        .start(); // inicia
```

El actor se desplazará por un rectángulo con la diagonal en 100,200 y 700,400. Es importante observar que al primer tramo lo recorrerá en 3 segundos, mientras que al resto en 1 segundo cada uno. Además, se utiliza el método **loop()** para hacer iterar indefinidamente cuando se tiene una cadena de Tweens.

En los ejemplos **02_tween_alpha_loop** y **03_tween** se pueden ver otras implementaciones de Tweens secuenciales. En **04_tween** se incorporaron Tweens simultáneos y secuenciales.

1.3 Callbacks de tweens

Es importante destacar que la clase Tween posee sus propios callbacks, es decir, acciones que se disparan cuando el Tween se encuentra en determinado estado. Los callbacks del tween son:

- onStart: se activa cuando el tween inicia, es decir cuando la propiedad a interpolar está en su estado inicial.
- onLoop: se activa cuando el tween se encuentra en los pasos intermedios de la interpolación.
- onComplete: se activa cuando el tween finalizó, estando la propiedad a interpolar su estado final.

Los tres callbacks deben definirse al objeto Tween y asignarle una función que se ejecutará cuando el evento se dispare.

Aquí un ejemplo de cómo definimos un callback:

```
function create() {

  ball = game.add.sprite(200, 200, 'ball');

  ball.scale.setTo(1.0, 1.0);

  tween = game.add.tween(ball.scale).to(escala, duracion, easing,
autostart,delay,repetir,yoyo);

  // habilitamos el callback del onStart y cuando se ejecute
llamamos a la funcion empieza

  tween.onStart.add(empieza, this);

  // cuando hace un ciclo
  tween.onLoop.add(loop, this);

  // cuando termina el tween
  tween.onComplete.add(complete, this);


}
```

En el ejemplo **05_estados** se puede encontrar la implementación completa de este ejemplo**.**

1.4 Tweens Data

Phaser permite realizar una simulación del Tween entre la cantidad intermedia de fotogramas definidos, basándose en los valores de configuración dada en *Tween.to*, generando una matriz con los valores de los objetos muestreados, interpolados desde el valor inicial al final (dependiendo de la función *Easing Function* definida).

Por ejemplo, si deseáramos modificar el color del fondo cambiando los valores de los canales rojo, verde y azul haciendo que vayan de 0 a 255 en un lapso de tiempo de 10 segundos para 60 fps, podríamos crear un Tween genérico que nos cree la matriz de valores interpolados la cual luego podríamos usar para realizar nuestra animación.

```
// Creamos a parte los datos que vamos a interpolar
//Los canales red, green y blue
var tweenData = { r: 0, g:0, b:0};

// con make lo declaramos, pero no lo instanciamos
tween = game.make.tween(tweenData).to( { r: 255, g:255, b:255},
duracion, easing);
```

Luego debemos generar el objeto *data* en base a los fps que deseamos:

```
var fps = 60;
data = tween.generateData(fps);
```

En *data* nos devolverá un arreglo con los valores muestreados, interpolados desde el valor inicial 0 al final 255. Es decir que disponemos de un arreglo de 600 valores (10seg * 60fps = 600).

Luego para acceder a un elemento interpolado debemos referenciarlo, considerando que cada posición del arreglo tiene el valor que tomar ese elemento en ese punto de la interpolación, por ejemplo en **data[200].r** tenemos el valor de r (red) en la interpolación 200.

Para el ejemplo, es posible usar estos valores muestreados para asignárselos al color de fondo del fondo en cada frame.

```
function update(){
  game.stage.backgroundColor = color;
}
```

Donde la variable *color* está compuesta por *data[index].r* , *data[index].g* y *data[index].b.*

En el ejemplo **06_data** se puede observar en detalle la implementación del Tween Data.

2. Timers

Los Timers son pausas que nos permiten controlar los tiempos de nuestros objetos. Estas pausas no hacen nada realmente, pero es posible agregarles funciones que se disparen en algún evento que estas ejecuten.

Antes de hablar sobre los Timer hay que conocer la diferencia entre el objeto Phaser.Time y el objeto Phaser.Timer. El primero es el objeto interno que controla el tiempo del juego, de él podemos obtener información sobre los fps que estamos ejecutando u otras variables del Core de la Librería. De este objeto se desprende el objeto Timer, de hecho, el objeto Timer es un evento del objeto Time. El objeto Timer es lo que describimos al principio, un evento que permite ejecutar pausas para controlar las distintas acciones que queramos tener en nuestro juego.

Controlar un Timer es muy sencillo, aquí veremos las situaciones más comunes, aunque se pueden ver más usos en la documentación de la librería:

http://phaser.io/docs/Phaser.Timer.html

Si así lo desean, pueden también ver la documentación del objeto Time:

http://phaser.io/docs/Phaser.Time.html

2.1 Definir un Timer

Para crear un Timer debemos hacerlo desde el estado *add*, allí definiremos su duración y qué función ejecutará cuando llegue al tiempo especificado:

```
game.time.events.add(duracion, funcionCallBack, callbackContext,
arguments);
```

Como mencionamos antes, el Timer es un evento del objeto Time. Lo que hacemos es agregarle un comportamiento al evento. Los parámetros que recibe son:

- duracion: El tiempo en milisegundos que trascurrirá hasta que se ejecute la función
- funcionCallBack: Es la función que se llamará cuando trascurra ese tiempo. Esta función debemos definirla nosotros
- callbackContext: Es el objeto al cual hace referencia, al menos que queramos definir un timer para un actor en particular, aquí siempre irá this, ya que hace referencia al objeto Game mismo.
- arguments: Son los argumentos para la función del callback, deben definirse como objeto. Si la función no recibe parámetros, este argumento se puede omitir.

Veamos un ejemplo aplicado:

```
var Timer = game.time.events.add(4000, CambiaColor, this);
```

Aquí definimos una pausa de 4 segundos (4000 milisegundos) que trascurrirán desde que se instanció el Timer (luego veremos que podemos controlar el inicio y corte del Timer) y cumplido este tiempo se ejecutará la función CambiaColor.

En el ejemplo **00_time_creacion** pueden ver un ejemplo simple de creación de Timer.

2.2 Loop y repeate

Al Timer lo podemos definir para que se ejecute indefinidamente, o bien, decirle que se repita X veces. Para definir un loop utilizamos la misma sintaxis que antes, pero ahora llamamos a la función *loop*

```
var Timer = game.time.events.loop(4000, CambiaColor, this);
```

En este caso, ejecutará la función CambiaColor indefinidamente cada una pausa de 4 segundos.

Si en cambio quisiéramos que se ejecute sólo 10 veces, lo que debemos hacer es crear el evento con la función *repeate*:

```
var Timer = game.time.events.repeate(4000, 10, CambiaColor, this);
```

Aquí el segundo argumento es el que dice cuántas veces se ejecutará la llamada a la función, siempre cada 4 segundos de delay.

En los ejemplos **01_time_loop** y **03_time_repeate** pueden ver ejemplos del uso de estos dos modos.

2.3 Pause, Resume y Remove

Y por último, una característica importante de los Timers es la posibilidad de controlar su llamada. Con la función Pause podremos detener un Timer infinito o que aun se esté repitiendo. Para ello debemos conservar una referencia del evento mediante una variable, para luego poder invocar a a Pause o a Resume.

```
var timer = game.time.events;

timer.loop(4000, CambiaColor, this);

timer.pause();
```

Aquí lo que estamos haciendo es detener la ejecución a continuación de la instanciación, luego podremos reanudar el loop mediante la llamada a la función *resume*

```
timer.resume();
```

Y volver a darle pausa:

```
timer.pause();
```

En el ejemplo **04_pause_resume** pueden ver el uso de estas dos funciones, el control se realiza cliqueando con el mouse

También es posible eliminar por completo el evento, para ello podemos utilizar la función *remove*, para hacerlo debemos tener una referencia al Timer. Notar que en el ejemplo anterior conservamos una referencia al evento en general y no al Timer en particular, para referenciar el Timer nuestra variable debe igualar la instanciación, como se muestra a continuación:

```
var Timer = game.time.events.loop(4000, CambiaColor, this);
```

Luego podemos eliminar el Timer mediante la llamada a la función *remove.*

```
game.time.events.remove(Timer);
```

Pueden encontrar un ejemplo completo del *remove* en el directorio: **02_time_remover**