CONTENIDOS

1. Inputs

Los inputs son los eventos de entrada que podamos tener en nuestro juego, como ser el mouse, el teclado, el jostick, el touch (en caso de los dispositivos móviles o monitores táctiles) o el acelerómetro. En esta unidad sólo veremos el teclado y el mouse y dejaremos los demás como opcional al final de la cursada, por si quieren emplearlo en algún proyecto particular.

Para los eventos vamos a manejar dos clases, la clase **input** y la clase **events.**

La clase input maneja todos los eventos que ocurran en el juego, esta clase se está controlando todo el tiempo por el Core de la librería.

La clase events, en cambio, son eventos "disparados" por la clase contenedora. Si se los definimos a un actor, los disparará el actor, si se los definimos a un grupo, los disparará el grupo, si se los definimos al Game, los disparará el juego en gral, y así para cualquier clase madre contenedora.

1.1 Eventos del mouse

Lo primero que debemos conocer es que los eventos pueden involucrar a un actor o al juego. En caso de involucren al juego, sólo debemos implementarlos, pero si llegan a involucrar al actor, debemos habilitarles los eventos, de la siguiente manera:

```
actor.inputEnabled = true;
```

1.1.1 Drag and Drop

Dentro de los eventos del mouse podemos separar el *drag and drop*. Este evento, por sí sólo, ya está implementado en la librería, por lo que sólo tenemos que activarlo y nuestro actor será *dragueable*.

Ejemplo:

```
function create() {
      diamante = game.add.sprite(300, 300, 'diamante');
     //  Habilitamos los eventos
      diamante.inputEnabled = true;
      //  Habilitamos el drag
      diamante.input.enableDrag(true);
 }
```

Esto es un fragmento del ejemplo **01_Drag_drop**

Algo interesante que tiene el Drag es que cuenta con sus propios callbacks, es decir, acciones que se disparan cuando el evento se encuentra en determinado estado. Los callbacks del drag and drop que podemos verificar son:

- isDragged
- onDragStop
- onDragStart

El primero no es exactamente un callback, sino más bien un estado que podemos consultar en la función **update** para saber si nuestro actor está siendo dragueado.

Ese estado se encuentra en la clase **input** (ya que como se mencionó al principio, es un estado que debe ser controlado continuamente por el Core).

Actor.input.**isDragged**

Devolverá *true* si el objeto está siendo dragueado y *false* en caso contrario.

Los otros dos callbacks deben definirse al objeto y asignarle una función que se ejecutará cuando el objeto se dispare. Como dijimos, son eventos que se accionan directamente por el padre contenedor, así que los vamos a encontrar en la clase events.

Aquí un ejemplo de cómo definimos un callback

```
function create() {

      Hongo = game.add.sprite(200, 200, 'hongo');

      Hongo.inputEnabled = true;

      Hongo.input.enableDrag();
      //  Habilitamos el callback del DragStop y cuando se ejecute llamamos
a la función VolverInicio

      Hongo.events.onDragStop.add(VolverInicio)

}

//  Volvera a la coordenada donde se origino
function VolverInicio(Spr) {

        Spr.x = 200;
        Spr.y = 200;

}
```

En el ejemplo **02_Drag_drop_callback** pueden encontrar la implementación de este ejemplo y también el uso del estado **isDragged.**

1.1.2 Llamados a eventos del mouse sobre un actor

Para el resto de los eventos del mouse debemos manejarlos de manera similar a cómo nos manejamos para los callbacks, recordar que los eventos que implementemos serán, en definitiva, acciones disparadas por el actor al cual se lo definimos.

Primero habilitamos los inputs para el actor:

```
actor.inputEnabled = true;
```

Ahora tendremos que agregarle al actor un evento y decirle qué queremos que pase cuando ese evento ocurra.

Los eventos del mouse que podemos manejar en Phaser son:

- .onInputOver
- .onInputOut
- .onInputDown
- .onInputUp

- .onDragStart
- .onDragStop

Los mismos se activan cuando:

onInputOver: El mouse (o touch) se encuentra sobre el objeto.

onInputOut: El mouse sale del objeto

onInputDown: Cuando presionamos el botón del mouse (o touch) sobre el objeto.

onInputUp: Cuando liberamos el botón del mouse (o touch) sobre el objeto.

Y los dos últimos ya los mencionamos anteriormente. Ahora lo que debemos hacer es definirle la función, de la misma manera que hicimos con el callback del Drag.

```
Hongo.events.onInputDown.add(clicks,this);

Hongo.events.onInputOver.add(sobre,this);

Hongo.events.onInputOut.add(salgo,this);

Hongo.events.onInputUp.add(suelto,this);
```

Donde **clicks** es una función definida por nosotros y se ejecutará cuando presionemos el botón sobre el actor Hongo. De la misma manera **sobre** es una función que se llamará cuando nos posicionemos con el mouse sobre el actor Hongo. Lo mismo con **salgo** para el *onInputOut* y **suelto** para *onInputUp.* De más está decir que los nombres de las funciones *clicks*, *sobre*, *salgo* y *suelto* fueron elegidos nosotros, pero podríamos haber puesto cualquier función aquí.

El parámetro **this** es una redundancia e indica que el actor que será llamado como parámetro en la función. Si no especificamos nada toma por defecto siempre el *this*, pero podríamos haber pasado otro actor. Por ejemplo si quisiéramos que al mover el Hongo otro actor, llamado Hongo2, cambie de color, debemos pasar como parámetro al Hongo2.

Luego nos queda definir las funciones *clicks*, *sobre*, *salgo* y *suelto*

```
function clicks(item){
    //  Se llama al onInputDown
}
function sobre(item){
    //  Se llama al onInputOver
}
function salgo(item){
    //  Se llama al onInputOut
}
function suelto(item){
    //  Se llama al onInputUp
}
```

Como habíamos mencionado, el parámetro **item** de las funciones hace referencia al parámetro **this** que colocamos en la implementación de más arriba.

Este ejemplo se encuentra implementado en **04_eventos_mouse**.

Para saber más sobre los eventos a actores, pueden revisar la documentación:

docs.phaser.io/Phaser.InputHandler.html

1.1.3 Llamados a eventos del mouse sobre el juego

No siempre queremos que el evento se llame sobre un actor, a veces necesitamos que se llame sobre el lienzo del juego en general.

Para definir un evento de mouse sobre el lienzo tenemos 2 alternativas:

Definirle una función al evento. Como lo hicimos recién, pero en este caso sería sobre el objeto **Game**

O bien, preguntar en la función **update** si se accionó un evento de teclado.

La diferencia de usar uno u otro reside en lo que necesitemos hacer. Ahora vamos a ver como implementar esto.

Primero vamos a ver cómo definirle una función a un evento. Es muy parecido a lo que hicimos antes. Esta vez no hace falta que le habilitemos los inputs al actor, ya que el objetivo del evento será el Game.

Los eventos que podemos definirle al Game son:

- onDown
- onHold
- onTap
- onUp

Los mismos se activan cuando:

onDown: Cuando el botón del ratón se presiona.

onUp: Cuando el botón del ratón se libera.

onTap: Cuando presionamos con el dedo en una pantalla táctil (equivalente al click del ratón)

onHold: Cuando mantenemos presionado con el dedo en una pantalla táctil **.**

Así le definimos una función a un evento del objeto Game

```
        game.input.onDown.add(clicks,this);

        game.input.onUp.add(suelto,this);
```

Nuevamente, **clicks** y **suelto** son dos funciones definidas por nosotros. La única diferencia aquí es que estos eventos responden al objeto *input*.

En el ejemplo **04_eventos_mouse2** pueden ver la implementación de los eventos al objeto Game.

Y como mencionamos antes, hay una segunda forma de registrar los eventos del mouse sobre el objeto Game, y es verificando en la función *update* si hubo presión del ratón. Para ello consultaremos por los estados:

game.input.mousePointer.**isUp**

game.input.mousePointer.**isDown**

Que nos dice si el botón del mouse está arriba o presionado, respectivamente. A continuación un ejemplo de cómo utilizaríamos estos estados en la función *update*

```
    function update(){

       if (game.input.mousePointer.isDown){
       // si el botón del mouse esta presionado hago esto
       }

       if (game.input.mousePointer.isUp){
       // si el botón del mouse esta liberado hago esto
       }

    }
```

En el ejemplo **04_eventos_mouse3** pueden ver el uso de los estados del mouse.

Con estos dos estados, más el evento del onInputDown en un actor, podríamos definir el evento del Drag nosotros mismos. Si les interesa ver cómo se hace, en el ejemplo **05_Drag_manual** está implementado. Sólo hace falta tener una variable que nos avise cuál es el objeto que se está moviendo.

Para saber más sobre los eventos al Game, pueden revisar la documentación:

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

1.2 Eventos del teclado

Existen dos formas de registrar teclas presionadas en Phaser y ambas son equivalentes. Podemos registrar cada tecla en particular y preguntar por su evento cuando se presione o bien podemos definir un objeto general y preguntar puntualmente si se presionó una tecla.

Ahora vamos a ver ambas formas.

Primero vamos a ver cómo definir una tecla particular, vamos a poner el ejemplo de la tecla UP (flechita arriba):

```
    // Necesitamos si o si definir una variable porque luego
preguntaremos por ella en la función update

    var arKey;

    // En la función preload la definimos
    function preload() {

        arKey = game.input.keyboard.addKey(Phaser.Keyboard.UP);

    }


    // Y luego en el update por el estado
    function update() {
        // preguntamos por el estado

        if (arKey.isDown){

        }
    }
```

La otra manera de registrar un evento de teclado es básicamente idéntica, sólo que definiremos un único objeto que responderá a todo el teclado por lo que sólo tendremos que preguntar por la tecla de nuestro interés:

```
    // De la misma manera, necesitamos si o si definir una variable
porque luego preguntaremos por ella en la función update
    var cursors;

    // En la función preload la definimos
    function preload() {

        cursors = game.input.keyboard.createCursorKeys();

    }

    // Y luego en el update preguntamos por el estado de alguna de
las teclas
    function update() {

        // preguntamos por el estado
        if (cursors.up.isDown){

        }
    }
```

Utilizar uno u otra manera es indistinto.

En los ejemplos **06_movimiento_de_sprite_con_teclado** y **06_movimiento_de_sprite_con_teclado2** podrán ver la utilización de ambas maneras de registrar un evento de teclado.

Para entender mejor el registro de eventos del teclado pueden consultar la documentación:

docs.phaser.io/Phaser.Keyboard.html

2. Animaciones

En la guía anterior vimos como animar spritesheets. Sin embargo no dijimos nada de qué hacer con un spritesheet irregular.

En un spritesheet como el siguiente no hay problemas porque todos los frames son iguales:

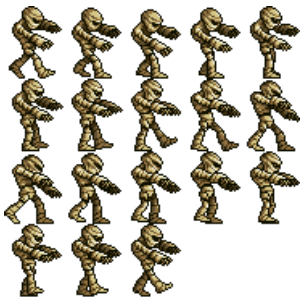

Figura 1: *Spritesheet regular*

Si lo quisiéramos dividir podríamos poner cada frame en un marco igual para todos los elementos del spritesheet:

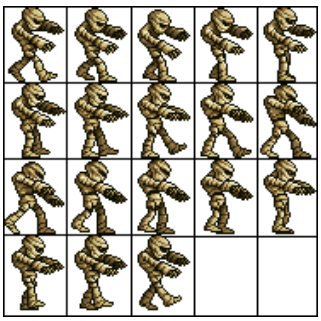

Figura 2*: División ficticia Spritesheet regular*

Sin embargo ¿Qué ocurre con un spritesheet como este?

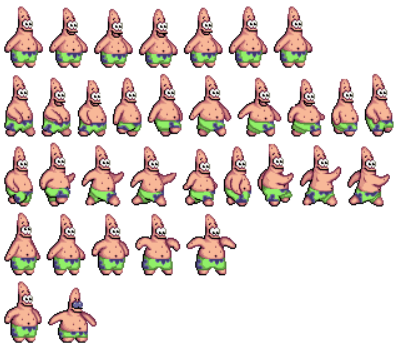

Figura 3: *Spritesheet irregular de Patricio Estrella*

Si quisiéramos encontrar un tamaño común para dividir en incluir a todos los frames no podríamos, porque cualquier tamaño que tomemos siempre habrá frames que calzarán bien pero otros que estarán cortados.

Para poder utilizar un spritesheet así necesitamos otro archivo denominado "atlas". El atlas es un archivo (por lo general JSON o XML) que nos dice donde se encuentra cada frame, a qué animación pertenece y qué índice lleva en la animación.

Para ver como se genera ese archivo vean el tutorial de "como_crear_atlas" (también en la carpeta de esta semana), aquí sólo los usaremos.

Para crear una animación de un spritesheet irregular debemos implementar un atlas mediante la función:

game.load.atlas

Que recibe por parámetros el archivo imagen del spritesheet y el archivo de atlas.

Para el spritesheet anterior la llamada la instanciación del atlas sería:

```
     function preload(){

    game.load.atlas('atlas',
'../../assets/sprites/patricio/spritesheet.png',
'../../assets/sprites/patricio/sprites.json');
    }
```

El primer parámetro es el id con el que luego recuperaremos el atlas, el segundo parámetro es la imagen de spritesheet y el tercero es un archivo Json que tiene la información del spritesheet.

Ahora nos queda crear el sprite como vimos en la unidad anterior.

```
var Patricio;

function create(){

        //cargamos a patricio
        Patricio = game.add.sprite(200, 200, 'atlas');
}
```

Como ya vimos, los primeros 2 parámetros dan la posición de *x* e *y* y el 3ro es la imagen que tiene como textura, esta vez asignaremos el atlas.

Ahora nos queda crear la animación, asignarla a nuestro personaje y ejecutarla. Para hacer ello tenemos que conocer un poco cómo está constituido el archivo Json del atlas.

El archivo Json se encuentra definido de esta manera:

```
{"frames":[
 {"filename":"idle_00","frame":{"x":1,"y":2,"w":31,"h"
 {"filename":"idle_05","frame":{"x":181,"y":2,"w":31,"
 {"filename":"idle_06","frame":{"x":217,"y":2,"w":31,"
 {"filename":"idle_01","frame":{"x":37,"y":2,"w":31,"l
 {"filename":"idle_04","frame":{"x":145,"y":2,"w":31,"
 {"filename":"idle_02","frame":{"x":73,"y":2,"w":31,"l
 {"filename":"idle_03","frame":{"x":109,"y":2,"w":31,"
 {"filename":"walk_04","frame":{"x":116,"y":56,"w":28
 {"filename":"walk_09","frame":{"x":290,"y":56,"w":23
 {"filename":"walk_00","frame":{"x":1,"y":57,"w":25,"l
 {"filename":"walk_05","frame":{"x":149,"y":57,"w":34
 {"filename":"walk_01","frame":{"x":31,"y":59,"w":24,"
 {"filename":"walk_03","frame":{"x":88,"y":59,"w":23,"
 {"filename":"walk_06","frame":{"x":188,"y":59,"w":35
 {"filename":"walk_07","frame":{"x":228,"y":60,"w":30
 {"filename":"walk_08","frame":{"x":263,"y":60,"w":22
 {"filename":"walk_02","frame":{"x":60,"y":61,"w":23,"
 {"filename":"run_01","frame":{"x":29,"y":111,"w":29,"
 {"filename":"run_06","frame":{"x":204,"y":112,"w":29
 {"filename":"run_02","frame":{"x":63,"y":113,"w":34,"
 {"filename":"run_07","frame":{"x":238,"y":113,"w":33
 {"filename":"run_00","frame":{"x":1,"y":114,"w":23,"l
 {"filename":"run_03","frame":{"x":102,"y":114,"w":37
 {"filename":"run_05","frame":{"x":177,"y":114,"w":22
 {"filename":"run_08","frame":{"x":276,"y":114,"w":34
 {"filename":"run_04","frame":{"x":144,"y":116,"w":28
```


Figura 4*: Fragmento del atlas JSON del spritesheet de Patricio*

Cada línea corresponde a un frame distinto, el Core utiliza estos datos para saber dónde se encuentra cada frame. A nosotros el único dato que nos interesa es el segundo, donde dice: idle_00, idle_05, idle_06, idle_01, idle_04, idle_02, idle_03, walk_04, walk_09, etc. No importa que no se encuentre en orden. De hecho, cada línea con el filename *idle_xx* corresponde a un frame de la primera línea del spritesheet de Patricio, donde el 00 es el primero, el 01 el segundo y así hasta el 06.

Resumiendo, la animación **idle** contiene 7 frames que van del 00 al 06. Con esta información vamos a crear la animación del idle (en reposo).

```
var _idle;

function create(){

   //creamos la animación idle
   _idle = Phaser.Animation.generateFrameNames('idle_', 0, 6, '', 2);
}
```

Los parámetros de la función generateFrameNames son:

_ Cadena de texto que corresponde al filename de la animación del archivo Json: **"idle_"**

_ Primer índice de la animación en el archivo Json: **00** (no importa el orden en el archivo, aquí especificaremos el primer frame de la animación).

_ Último índice de la animación en el archivo Json: **06**

_ Cadena de caracteres al final de la secuencia del filename. En este caso no hay ninguna, si en el archivo Json la secuencia fuera 'idle_00_abc', 'idle_01_abc', 'idle_02_abc', etc. Este parámetro sería "**_abc**".

_ Cantidad de cifras para numerar los índices, en este caso hay 2 cifras idle_**XX**. Si la cadena fuera idle_**001**, idle_**002**, etc. Este parámetro sería 3.

Con esto hemos creado la animación, ahora sólo hay que asignarla al sprite y ejecutarla.

```
//Le agregamos la animación al sprite

Patricio.animations.add('idle',_idle , 10, true);

//Y la ejecutamos

Patricio.animations.play('idle');
```

Los parámetros de la función `Patricio.animations.add` son:

_ Id que le damos a la animación (para después identificarla).

_ Variable que le hace referencia.

_ FPS

_ Animación cíclica (true o false)

En el ejemplo **02_MultipleSprite** está el ejemplo completo de la animación de Patricio.