**UNIVERSIDAD NACIONAL DEL LITORAL**
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS


**Tecnicatura en diseño y programación de videojuegos**

UNL VIRTUAL

# Prototipado rápido

**Unidad 2**
FlashPunk: entidades y mundos

**Docente**
Ariel Cid

# CONTENIDOS

# 1. Proyectos en FlashPunk

En el primer apunte teórico se detallaron los conceptos mínimos que se necesitaban para comenzar a abordar la materia. Desde esta semana en adelante, el contenido a desarrollar será casi en su totalidad de índole teórica. Para comenzar con esta labor, se explica la manera en que se deben crear los proyectos de FlashDevelop para que use correctamente el código de FlashPunk:

- Ir a “Project”, en la barra de herramientas.
- Hacer click en “New Project”.
- Seleccionar “AS3 project” dentro de ActionScript 3.
- Indicar nombre y ubicación del proyecto nuevo.
- Dar el OK.

Con esta acción se creará un proyecto de ActionScript 3 genérico, con sus directorios y configuraciones *default*. Luego se debe incluir la librería de FlashPunk y modificar algunas líneas de código para poder empezar a utilizarlo. Dentro del archivo FlashPunk.zip subido a la plataforma hay una carpeta llamada “net”, la cual debe colocar dentro de la carpeta “src” del propio proyecto (esto puede hacerse desde el Explorador de Windows o desde el explorador de proyecto dentro de FlashDevelop).

Para que la aplicación se convierta en un juego de FlashPunk, se deben eliminar los *import* que se crearon por defecto dentro de Main.as, y agregar la siguiente línea:

```
import net.flashpunk.Engine;
```

Luego, la clase Main debería ser una subclase de Engine, en vez de Sprite. Esa línea de código debería quedar similar al siguiente:

```
public class Main extends Engine
```

Con esto, lo único que restaría es, dentro del constructor de la clase Main, llamar al constructor de su clase base (Engine) para poder tener el motor de FlashPunk andando. Los parámetros que requiere dicho método se explican en la próxima sección. El código de la clase debería verse ahora de esta manera:

```
package
{
        import net.flashpunk.Engine;
        public class Main extends Engine
        {
                public function Main():void
                {
                        super(640, 480, 30, false);
                }
        }
}
```

Hasta aquí, al compilar el proyecto tendría que funcionar todo correctamente.

# 2. Propiedades del juego, la pantalla y la ventana

Es necesario explicar ciertas propiedades iniciales a setear en este punto. Las propiedades del constructor de Engine, llamado desde el constructor de Main, son las siguientes (en orden):

- Ancho del area de juego (640 en el ejemplo anterior).
- Alto del area de juego (480 en el ejemplo anterior).
- Cantidad de cuadros por segundo al que el juego va a correr (FPS).
- Tipo de *timestep*: fijo (*true*) o variable (*false*).

Este último parámetro es sumamente importante ya que va a regir la medida de tiempo en los juegos que se generen.

En el modo fijo el juego correrá siempre a la cantidad de cuadros por segundo que se le indique. Si se encuentra en este modo y el rendimiento empieza a mermar, lo que hará para contrarrestarlo y mantener la fluidez visual será omitir cuadros a dibujar (*frame skipping*), volviendo al juego crecientemente impreciso de manera proporcional a la cantidad de cuadros omitidos; la unidad de tiempo que se mantendrá constante en este modo serán los *frame*s.

En el otro modo, el variable, el juego tratará de correr a la cantidad de cuadros por segundo que se le indique previamente pero, si el rendimiento merma, la cantidad de cuadros por segundo bajará debido a que necesita dibujar cada uno de los *frame*s. Su unidad de tiempo serán los segundos, ya que los *frame*s variarán según la exigencia al hardware.

Al usar el Flash Player para reproducir el .swf que contiene el juego compilado, es imperativo también especificar las dimensiones que la ventana debe tener. De esta manera, se especifica debajo de los *import* qué tamaño se desea:

```
[SWF(width="640",height="480")]
```

Por último, para acceder a uno de los parámetros que rige las propiedades de la pantalla se puede importar la clase FP. Así, se pueden establecer las diversas preferencias, como color de fondo o escala del juego (si se quisiera, por ejemplo, que el juego se vea el doble de grande que su tamaño real).

```
FP.screen.color = 0x000000;
FP.screen.scale = 1;
```

# 3. Mundos

Existen dos clases primordiales en FlashPunk: los mundos (*World*) y las entidades (*Entity*), que se verán en el próximo apartado).

World es una clase que organiza las cosas a dibujar en pantalla de una forma simple y práctica ya que sólo permite tener un World activo en pantalla, y actualiza el estado de todas las entidades que se encuentran dentro de él en cada *frame*. Se pueden crear cuanto World se quiera pero sólo uno está activo en un mismo momento; un World puede ser el menú principal, otro un nivel de juego, etcétera. Entonces, es siempre necesario tener un World activo donde el jugador esté interactuando, por lo que se debe crear (por lo menos) una clase que herede de ésta.

```
package
{
        import net.flashpunk.World;
        public class GameWorld extends World
```

```
        {
                public function GameWorld() {}
        }
}
```

Al crear un Main, es preciso setear inicialmente un World como activo.

```
public function Main():void
{
        super(640, 480, 30, false);

        FP.world = new GameWorld();
}
```

La asignación de World activo puede ser hecha también dentro del método *init* de la clase Engine, que se ejecuta una sola vez, la primera, cuando se pone en funcionamiento.

```
override public function init():void
{
        FP.world = new GameWorld();
}
```

# 4. Entidades

La clase Entity sirve para crear objetos que realicen acciones dentro del World donde se juega. Una Entity puede ser un personaje, un enemigo, un ítem, los elementos de interfaz, entre otros. Necesitan, además de un comportamiento propio, estar asignadas a un World activo para que se dibujen en pantalla y se actualicen cuadro a cuadro. Para esto se crea una nueva clase que herede de Entity, y se la asigna al World que se creó en el apartado anterior.

```
package
{
        import net.flashpunk.Entity;
        public class Player extends Entity
        {
                public function Player() {}
        }
}
```

Luego, hay que añadir la clase a algún mundo; se agrega la siguiente línea dentro del constructor de GameWorld:

```
this.add(new Player());
```

Convendría, en caso de que efectivamente sea esta entidad la que el jugador controlará, agregarla como un atributo del GameWorld para poder acceder a dicho atributo y modificar sus propiedades. El código quedaría entonces algo así:

```
public class GameWorld extends World
{
        protected var player:Player;
        public function GameWorld()
        {
```

```
                this.player = new Player();
                this.add(player);
        }
}
```

Tanto las Entity como los World poseen un método llamado *update* que contiene todo el código a ejecutar cuadro a cuadro para actualizar su estado. Para especificar el comportamiento específico de las nuevas clases se debe sobreescribir dicho método.

```
override public function update():void { super.update() }
```

Con esto, el mundo y la entidad que contiene se actualizan en cada *frame*. Restaría darle una representación gráfica a dicha entidad; para esto se usa una imagen llamada "player.png" que habrá sido creada previamente en el directorio. Además se crea una instancia de la clase Image (que habrá que importar también) para poder manipular los atributos de dicho PNG (color, alpha, rotación u otros).

```
[Embed(source="player.png")] private const PLAYER_IMG:Class;
private var playerImage:Image;
```

Esta línea contendrá la dirección física (relativa al directorio "src") y será atributo de Player. Esta implementación se muestra sólo para ejemplificar su uso; en un proyecto lo suficientemente grande quizá convenga disponer de una clase o archivo externo que contenga todas las direcciones físicas de los *assets* a utilizar. Se creará entonces desde esta dirección una imagen nueva para asignar a la entidad dentro del constructor de la misma (importando la clase Image), debido a que el atributo "graphic" de cada entidad es de otra clase (justamente, la clase Graphic) que no tiene acceso a todos estos parámetros.

```
this.playerImage = new Image(PLAYER_IMG);
this.graphic = playerImage;
```

Esta imagen será dibujada ahora en pantalla en cada *frame* pero en el origen de coordenadas (0,0). Desde el llamado a su constructor habrá que especificar dónde se creará, y se elegirá el centro de la pantalla.

```
this.player = new Player(FP.screen.width/2, FP.screen.height/2);
```

El constructor de Player también deberá cambiar para reflejar esta necesidad. El código del constructor quedaría entonces así:

```
public function Player(px:Number = 0, py:Number = 0)
{
        this.playerImage = new Image(PLAYER_IMG);
        this.graphic = playerImage;
        this.x = px;
        this.y = py;
}
```

Con esto, la entidad ya estará dibujándose en las coordenadas que se le dictaron. Restaría darle un comportamiento a dicha entidad, lo cual será el tema central de la Unidad 3.