CREANDO NUESTRAS PROPIAS ENTIDADES

APRENDE A PROGRAMAR TUS PROPIOS VIDEOJUEGOS

BOOTCAMP UTN 2019

EL PROBLEMA

A medida que nuestros personajes o entidades empiezan a hacerse más complejos deja de ser práctico mantener su código dentro de la escena y resulta mejor moverlo a una nueva clase específica.

VENTAJAS

- Código más prolijo
- Entidades reutilizables (encerramos todo lo relevante a una entidad en la clase, es más fácil llevarla a otro juego)
- Permite manejar mejor gran cantidad de entidades
- Facilita la detección y manejo de colisiones

CREAR NUESTRAS PROPIAS ENTIDADES

- Nos basamos en la clase **FlxSprite**
- Definir constructor (new)
- Redefinir método update()
- Agregar nuestras funciones (servicios)

LA CLASE HERO

```
class Hero extends FlxSprite {
  public function new(X: Float, Y: Float)
  {
    super(X, Y, "assets/images/hero.png");
    velocity.x = 100;
  }

  public override function update(elapsed: Float):Void
  {
    super.update(elapsed);
  }
}
```

CONSTRUCTOR

- Es la función que se declara con el nombre *new*
- Se invoca cuando se crea nuestra entidad (desde la escena) con la palabra *new*
- Puede recibir los parámetros que deseemos (generalmente la posición)
- **NO OLVIDAR** invocar al constructor de la clase padre (FlxSprite) **super**

EL MÉTODO UPDATE()

- Se invoca para actualizar el objeto, igual que *update()* de la escena
- Recibe como argumento el tiempo transcurrido
- Se debe declarar con *override* ya que se está redefiniendo el método de la clase padre
- **NO OLVIDAR** invocar al método *update()* de la clase padre con **super**

ATRIBUTOS

Notar que para los atributos heredados de **FlxSprite** (x, velocity, etc). no es necesario anteponer el objeto (ship.x, hero.velocity, etc) ya que estamos **dentro** del objeto, **somos el objeto**

SERVICIOS

```
class Hero extends FlxSprite {
  public function new(X: Float, Y: Float)
  {
    super(X, Y, "assets/images/hero.png");
    velocity.x = 100;
  }

  public function hit(): Void
  {
    energy = energy - 10;
  }

  var energy: Int = 100;
}
```

SERVICIOS

- Las variables que declaramos dentro de la clase (atributos) son **sólo visibles dentro de ella** a menos que se declaren como public
- Esto permite aislar el código de la entidad del de la escena

SERVICIOS

- Sólo funciones declaradas dentro de la clase (métodos) pueden utilizar estas variables
- Algunos métodos deberían ser públicos para permitir a la escena interactuar con la entidad

SERVICIOS

Además, de ésta manera agregamos **significado**, nuestra entidad ya no es un simple sprite sino que tiene métodos de propios de lo que es (por ej: *lanzarHechizo()*, *obtenerEnergia()*, etc.)