Me cago en el zoo de Moock

Preliminares
Realmente el título de este archivo debería ser algo como “el Gran Salto” o cómo pasar de AS.2 a AS.3 sin demasiados traumas. Pero seamos realistas, la mayoría de los que decimos que sabemos AS.2 nos hemos quedado en un triste AS.1 sabiendo que en AS.2 hay que definir el tipo de todas las variables que usemos.

Así pues este documento pretende ser, desde la modestia e ignoracia de alguien que recién nace a AS.3 un manual para que no sea tan traumático el salto. Trataré de explicar las emociones que me produce este nuevo lenguaje y cómo lo encajo con todo lo que sé de Flash de otras versiones. Para mí, AS.3, me produjo vértigo al principio (y debo dejarlo bien claro aquí). Mi primera impresión fue que “estos de Adobe la han cagado” (eso sí lo sigo pensando) porque han separado muchísimo lo que es diseño y programación dejando claramente fuera de juego a los diseñadores.

Yo soy programador. Quizá me acerque a eso que llaman diseñador/programador aunque mi fuerte no será nunca el diseño y mi originalidad dista mucho de ser original. Sí soy muy parecido a muchos aficionados a Flash y a muchos de los que lo conocemos desde que era Macromedia Flash 4.0. Creo firmemente en esa persona “multitarea” tan parecida a mí y, pese a los esfuerzos de Adobe por discriminar programadores y diseñadores, creo que en la web son dos roles que pueden ir juntos o separados, pero que nunca será bueno que alguien de una disciplina no tenga ni idea de la otra o sea incapaz de valorarla.

Es por eso el título de “Me cago en el zoo de Moock”. Porque si alguien que se aproxima a AS.3 desde el Flash que conocía, al libro de “Essential Action Script 3.0” de Colin Mook se encontrará un tocho de casi 1000 páginas de árida programación, de teoría de OOP, de patrones y demás y se preguntará: “Si yo sólo quiero crear una animación chula, ¿debo saber todo esto?” con unas ganas terrible de mandarlo todo a la mierda: a Flash, a Adobe, al propio Moock y a su puto zoo de los cojones. Resulta curioso que Adobe haya olvidado a los diseñadores; que Moock los haya olvidado es, para mí, imperdonable.

Así que mi idea es hacer un prólogo “apócrifo” del libro . Un texto que anime a leer esas 1000 páginas que nos aclarará muchísimos conceptos, cómo y por qué funcionan las cosas, qué fundamentos y qué razones hay para hacer según qué cosas. Un modesto complemento para animarnos a leer ese libro, por otra parte referencia obligada. Y, desde mi ignorancia, comentar unas impresiones sobre lo que es AS.3, cómo funciona y cómo hacer cosas que nos eran tan simples hacerlas en AS.2

Notación
Como mi idea es escribir código en AS.1 y en AS.3 lo voy a diferenciar por colores (aunque es bastante diferente el código. Así

```
Esto es AS.1
```

```
Y esto es AS.3
```

AS.3, (ese gran desconocido)
Flash ha cambiado, no hay quien lo evite: AS.3 es distinto. AS.3 se ha converido en un leguaje propio de programación y no está sólo pensado para usar en el IDE de Flash (en la ventana de acciones del Flash de toda la vida). AS.3 nace como un lenguaje propio y con tendencia a que nuestro .fla sea un .fla vacío con un montón de elementos en la biblioteca. Sigue manteniendo sus frames y sus Layers en la película principal, pero casi nos podemos hacer a la idea de que es por un cierto romanticismo. Se podría decir exagerando un poco que todas las películas van a ser un simple frame vacío.
Ello no quiere decir que los símbolos que creemos no tengan frames, pero olvidémosnos de escribir código en un símbolo salvo el socorrido

```
stop();
```

Olvidémonos de que cuando llegue a aun frame se cargue una película, olvidemonós de la "línea de tiempo", de esos "gotoAndPlay(2)".... Aún así, todavía tenemos una oportunidad que es programar en la línea de tiempo. Vamos tener en un frame una instancia de un MovieClip llamado mimc y escribir en un frame

```
mimc.onEnterFrame=function(){
   this._x++
}
```

Que, en AS.3 sería como

```
import flash.events.*
mimc.addEventListener(Event.ENTER_FRAME,onEnter_Frame,false,0,true)
function onEnter_Frame(e:Event){
     e.target.x++;
}
```

para que se mueva.

Aunque lo normal no es escribir en la línea de tiempo. Lo normal es tener completamente separado la programación del .fla, lo normal es tener sólo un .fla y diversos .as

Así que olvidemonos de escribir en la línea de tiempo salvo para casos puntuales. Ordenemos un poco las cosas y ¡larga vida a AS.3!

Las cosas han cambiado

Pero no mucho en lo fundamental. Cuando nos enfrentamos a AS.3 vemos que todo son clases y mucha OOP pero no debe de asustarnos. Una clase NO es el "zoo de Moock", una clase no es algo abstracto. Una clase es un MovieClip, una clase es la clase Math que tan buenos ratos nos ha hecho pasar gracias a su randomize

```
for (var i=0;i<10;i++){
    var valor=(3+Math.floor(7*Math.randomize()))
    trace(valor)
}
```

y una clase es nuestro querido Array, nuestro loadVars, nuestro XML Object, nuestro Mouse.... Todo el mundo que haya programado en AS tiene, sino el "doctorado" en OOP, si el "currículum" de haber trabajado en OOP (ya se sabe lo que se exagera en los currículum)

Tomemos la clase más interesante de AS, la clase MovieClip (Vale, los amigos no le dábamos el tratamiento de "Clase" , e incluso le llamábamos cordialmente como "MC" –si habéis conocido a mi amiga Mari Carmen entendereís el chiste-).

Una clase no es algo más que un "algo" con propiedades, métodos y variables. Así, si teníamos un MC llamado "mimc" podríamos escribir

```
mimc._x=100          //le cambiamos el valor de una propiedad
mimc.encendido=true  //le dábamos valor a una variable
mimc.duplicateMovieClip("copia",100)  //usamos el método "duplicateMovieClip"
if (mimc.hitTest(otro)){   //si lo que devuelve el "método" hitTest es cierto
    mimc.encendido=false   //le damos valor a una variable
}
```

Ah!, me diréis, pero es que los MCs también respondían a eventos. Bueno, me he jartado de decirlo, así que no lo puedo negar, aunque sí matizar. A un MC le ocurren eventos y mediante AS podemos "capturarlos" para que cuando ocurran hacer algo con el MC. Sí lo típico que puse antes (pero como me encanta cortar y pegar...)

```
mimc.vel=5
mimc.onEnterFrame=function(){
   this._x+=this.vel
}
```

En AS.3 ocurre lo mismo: Tenemos clases también, pero las cosas cambian un poco dependiendo de si usamos ficheros .as o si programamos en la línea de tiempo.

De cualquier modo, una diferencia fundamental que nos encontramos es que

- A los MCs ya no les ocurren más eventos que los que nosotros les indiquemos, mediante addEventListener
- Una vez que decimos que nuestro MC está preparado para ese evento haremos la función que controle lo que ocurre.
- La mayoría de las "propiedades" de los MCs se llaman igual pero sin el "_" de detrás. Así la *_x* ha pasado a ser *x*, *_width* a ser *width*, *_alpha* a ser *alpha*...

En nuestro primer ejemplo de AS.3, teníamos un MC cuyo nobre de instancia es "mimc" y escribimos en la línea de tiempo:

```
import flash.events.*
mimc.addEventListener(Event.ENTER_FRAME,onEnter_Frame,false,0,true)
function onEnter_Frame(e:Event){
    e.target.x++;
}
```

El

```
import flash.events.*
```

No nos debe asustar. Al fin y al cabo no es más que, como la función addEventListener requiere como primer argumento un número según el evento que queramos añadir, pues en ese fichero está definido esos números. Vamos es algo parecido al Math.PI que usábamos

```
var grados=130
var rad=grados*Math.PI/180
```

Ya hemos dicho que debemos indicarle que cuando le ocurra un evento haga algo, pero no lo podemos hacer como hacíamos antes simplemente definiendo la función onEnterFrame, sino algo parecido a lo que hacíamos con los lísteners.

Comenté alguna vez sobre los lísteners que lo que ocurre es que algunos objetos no son capaces por sí solos de recibir un evento y delegan en otros objetos. Así, con un MovieClipLoader escribíamos

```
var loader=new MovieClipLoader()
var listener=new Object()
listener.onLoadInit(MovieClip:target){
   target._x=100;
}
loader.addListener(listener)
loader.loadClip("imagen.jpg",mimc)
```

Igualmente, una vez que les decimos que les ocurra un evento, podemos capturar ese evento y que haga algo cuando le ocurra.

En AS.3 tenemos un listener ya definido –por defecto- que es el que se encarga de "escucharlo todo". Por tanto no tenemos que definirlo. Y sí, cambia un poco el modo de decirle a ese listener que se mantenga a la escucha de lo que le ocurra a mimc. Si en AS.2 lo hacíamos con addListener(), en AS.3 lo hacemos con

```
mimc.addEventListener(....)
```

y si en AS ya teníamos, de algún modo cómo llamar a las funciones que recogían los eventos ya definidas (ya sabíamos que el nombre era onLoadInit), aquí le tenemos que dar nombre a la función –que es el segundo parámetro- Pero siguen estando definidos los parámetros que le llegan.

Otro modo de verlo. Imaginemos que tenemos varios MCs y queremos que algunos se muevan. Podremos hacer muchos onEnterFrames

```
mc0.onEnterFrame=function(){
     this._x++
}
mc1.onEnterFrame=function(){
     this._x++
}
mc2.onEnterFrame=function(){
     this._x++
}
```

O uno sólo

```
onEnterFrame=function(){
   for (var i=0;i<3;i++){
       var mc=this[“mc”+i]
       mc._x++
   }
}
```

ó, si tenemos un array

```
mismcs=new Array()
mismcs.push(mc0)
mismcs.push(mc1)
onEnterFrame=function(){
    for (var mc in mismcs){
           mismcs[mc]._x++
    }
}
```

Sí, ese “push” es lo que nos facilita que se muevan, en este caso los MCs mc0 y mc1, con un solo onEnterFrame. Si quisiéramos que se moviera el tercero, bastaría con añadir un

```
mismcs.push(mc2).
```

Si venimos de otros lenguajes de programación, también hay algo parecido a ese push o a ese addEventListeners

Vale, ¿por qué digo que es parecido a los lísteners? Fijaros en el modo de referirnos a la “x” del MC

```
e.target.x++;
```

Sí, no podemos usar “this”, puesto que “this” es la propia película. Si, p.e. dibujamos una línea en nuestro .fla y hubiéramos escrito

```
import flash.events.*
mimc.addEventListener(Event.ENTER_FRAME,onEnter_Frame,false,0,true)
function onEnter_Frame(e:Event){
     x++;
    //o this.x++;
}
```

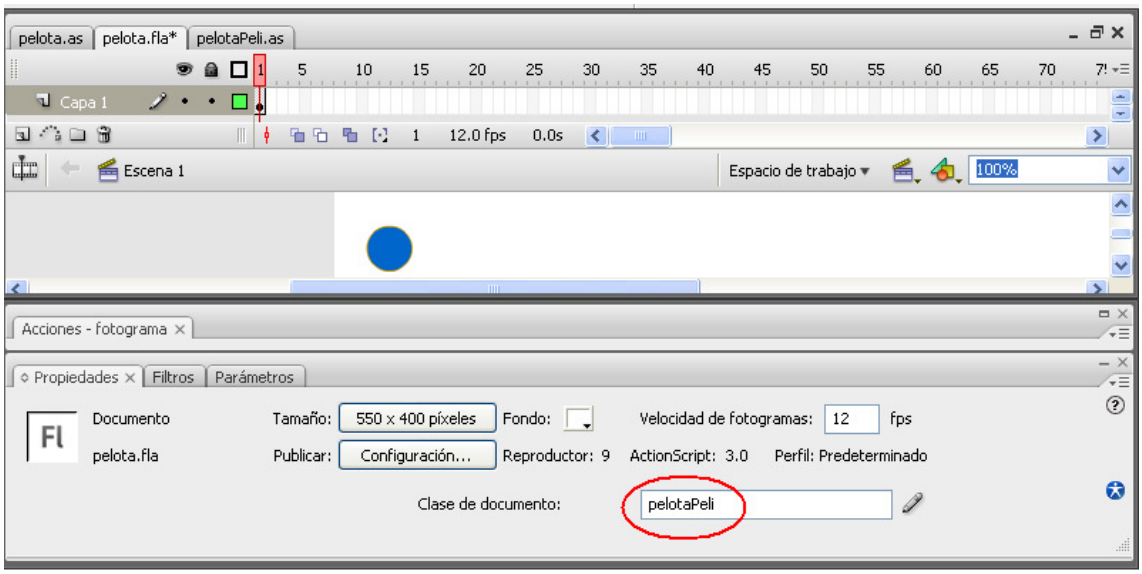
Veríamos que lo que efectivamente se mueve es la película entera.

Es más, debemos darle el cuarto argumento al addEventListener para que e.target sea una “referencia” a nuestro MC.

Usando ficheros .as

Bueno, vamos a sacar nuestro código anterior de la línea de tiempo. Para ello deberemos

- Crear una clase que relacionará nuestra película
- Hacer que nuestra película haga referencia a dicha clase



Sí, vamos a crear un fichero que se llamará pelotaPeli.as, donde quedará definida nuestra clase pelotaPeli y haremos corresponder a nuestra película.fla dicha clase.

El código queda “simplemente”

```
package {
    import flash.events.*
    public class pelotaPeli extends MovieClip {
        public function pelotaPeli() {
            mimc.addEventListener(Event.ENTER_FRAME,onEnter_Frame,false,0,true)
        }
        public function onEnter_Frame(e:Event):void{
          e.target.x+=10;
        }
    }
}
```

¡Glups!, package, extends, function pelotaPeli.... ¿qué es todo eso?

Lo del package es simplemente que queremos, de algún modo que todo forme parte de un todo más general. Al fin y cabo, el motivo de tener separados el código en fichero .as sirve para el objetivo de poder reutilizar ese código alguna vez. Es por ello útil tener los códigos “empaquetados” y ello hará fácil nombrar clases.

Luego está eso extraño de

```
public class pelotaPeli extends MovieClip
```

Bueno, ya hemos dicho que un MC no es más que un Objeto de la Clase MovieClip, nuestra película ya no es una película cualquiera. Nuestra película es una película de la clase pelotaPeli, porque se lo hemos indicado al indicarle la “Clase de documento”. Pues bien. Al definirla como clase del documento, DEBE definirse como una subclase de MovieClip. Eso es lo que nos dice el extends.

Lo bueno de las clases es que, lo primero que se hace, es “llamar” a la función de creacción de la clase. Vale, si tenemos una clase llamada “pelotaPeli”, lo primero que hace es llamar a la función “pelotaPeli”. Esa función debe existir, debe ser pública y debe definirse exactamente como está hecho. Es por ello que es en esa función cuando aprovechamos para decirle a nuestro “escuchador de todo” que añada el evento.

```
public function pelotaPeli() {
            mimc.addEventListener(Event.ENTER_FRAME,onEnter_Frame,false,0,true)
}
```

El resto aparece con lo ya conocido.

AS.3 es OOP: una vuelta de tuerca

Vemos pues que AS3, en realidad no es tan diferente a lo que conocíamos. Vale, antes escribíamos casi todo el código en el primer frame de nuestra película, ahora lo hacemos en un .as separado y le tenemos que dar eso del package y tener cuidado con el nombre de la clase. ¡Pero es SÓLO ESO!.
Antes no teníamos que añadir los eventos al “gran hermano escuchador”, ahora sí pero eso hace precisamente que sean más rápidas las películas hechas con AS.3.

Vemos que son todo ventajas y que no hay que desanimarse viendo mascotas, zoos y demás vainas...

Ahora hay algo que me gustaba mucho a mí que era dar variables a los MCs. Bah!, no preocuparse, es que yo, en momento que podía, me ponía a “parametrizar” botones, y MCs. Vamos, el ejemplo anterior.

```
mimc.vel=5
mimc.onEnterFrame=function(){
   this._x+=this.vel
}
```

Ó digamos que tenía varios MCs en pantalla llamados “copo0”, “copo1” y “copo2” y escribía algo como

```
for (var i=0;i<3;i++){
     var mc=this[“copo”+i]
     mc.velx=3-6*Math.random()
     mc.vely=3-6*Math.random()
     mc.onEnterFrame=function(){
            this._x+=this.velx
            this._y+=this.vely
     }
}
```

ó tenía tres botones “bt0”,”bt1” y “bt2” y un MC llamado “contenedor” y escribía peliculas=new Array(“pelicula1.swf”,”pelicula2.swf”,”pelicula3.swf”)

```
for (var i=0;i<3;i++){
     var boton=this[“copo”+i]
     boton.pelicula=peliculas[i]
     boton.onRelease=function(){
           contenedor.loadMovie(this.pelicula)
     }
}
```

Vamos, daba variables distintas a un grupo de MCs o botones, y según esa variable, el “comportamiento” variaba un poco. En el primer ejemplo se mueven a distintas velocidades y en el segundo cargan distintas películas.

Pues en AS.3 seguimos con lo mismo

```
package {
    import flash.events.*;
    public class pelotaPeli extends MovieClip {
            function pelotaPeli() {
                    mimc.velx=5;
                    mimc.addEventListener(Event.ENTER_FRAME,onEnter_Frame,
                                    false,0,true);
            }
            function onEnter_Frame(e:Event):void {
                    e.target.x+=e.target.velx;
            }
    }
}
```

```
package {
    import flash.events.*;
    public class pelotaPeli extends MovieClip {
            function pelotaPeli() {
                for (var i:int=0; i<3; i++) {
                    var mc_txt:String="copo"+i.toString();
                    var mc:MovieClip=(stage.getChildAt(0) as
                                MovieClip).getChildByName(mc_txt) as MovieClip;
                    mc.velx=6-3*Math.random();
                    mc.vely=6-3*Math.random();
                    mc.addEventListener(Event.ENTER_FRAME,onEnter_Frame,
                                false,0,true);
                    }

            }
            function onEnter_Frame(e:Event):void {
                    e.target.x+=e.target.velx;
            }
    }
}
```

¡Glup! GetChildAt, getChildByName....¿dónde está nuestra querida notación de corchetes? Pues **a nuestra notación de corchetes se la ha comido el puto zoo de Moock**. Vale, nos tenemos que olvidar de ella y usar la "colección de objetos de visualización".

Hay bastante que hablar sobre esa colección de objetos de visualización. Antes, cada objeto de Flash tenía relacionado un "array con todo lo que contenía", ya fueran variables, funciones, propiedades u otros MCs.
AS.3 diferencia entre objetos visibles (displayables) -que incluye también todas las líneas, formas, etc que dibujemos-, las variables que le demos a nuestros MCs y el resto

Así, un bucle for..in sobre mimc anterior

```
package {
    import flash.events.*;
    public class pelotaPeli extends MovieClip {
            function pelotaPeli() {
                    mimc.velx=5;
                    for (var i:String in mimc){
                          trace(j,mimc[j])
                    }
            }
    }
}
```

Sólo nos devolverá un triste

```
velx 5
```

Si queremos acceder a las "propiedades" también podemos usar la notación de corchetes

```
trace(mimc["x"])
```

Pero no es lo mismo ¡con lo bonito que era el bucle for...in!

Pero para acceder a los elementos interiores de un MC que sean “displayables” debemos usar la “colección de objetos de visualización”, y usar getChildAt ó getChildByName
Lo de concatenar métodos y métodos para formar algo del tipo

```
stage.getChildAt(0).getChildByName(“copo”+i)
```

No va a funcionar tan sencillo porque la colección no siempre devuelve algo del tipo esperado. Cuando queramos “forzar” a que nos devuelva algo de un tipo usaremos el operador “as”, evidentemente no podemos forzar a que algo que devuelve peras, devuelva manzanas

```
var mipera:Pera=new Pera()
var mimanzana:Manzana=mipera as Manzana  //←va a ser que no
```

y, hasta que nos acostumbremos a saber cómo están relacionadas las distintas clases va a pasar un cierto tiempo precioso que podríamos estar con nuestra pareja (por ejemplo, la podemos invitar a pasar una tarde en el zoon –de Moock, por supuesto-) mandando todo esto al diablo, pero es lo que hay

AS.3 es OOP: otra vuelta de tuerca
(Afortunadamente, una vez apretada esta tuerca, todo quedará perfectamente apretado y definido)

Hasta ahora, la cosa no difiere mucho de lo que hemos hecho a lo largo de toda la vida. De acuerdo que hemos hecho un .as para relacionarla con la película y hemos creado un paquete y nos hemos quedado sin notación de corchetes pero tenemos las herramientas para recorrernos nuestros MCs. Ahora si os digo que la tendencia es que una película sea simplemente una película vacía con un montón de objetos en la biblioteca (algo parecido a lo que hacían algunos autores attachando objetos y más objetos) me váis a decir que es una moda pasajera.

Creo que no va a ser pasajera, creo que se va a quedar con nosotros y que, además es bueno. Sí, separa un poco más las cosas, pero se clarifican. Veamos un poco el sentido de esto. Ya nos hemos dado cuenta que nuestra película no era una película cualquiera, sino una película de la clase “pelotaPeli”. Ahora nuestros MCs no van a ser unos MCs cualquieras, sino que además de proceder de símbolos distintos van a ser de clases distintas.

Imaginemos nuestro primer juego en AS3 que es un “héroe” que “dispara” a los “enemigos” (siempre me gustaron los arcades). Haremos un símbolo héroe, un símbolo enemigo y un símbolo disparo. ¿por qué limitarnos a que sean símbolos distintos y no les elevamos de categoría y hacemos que sean Clases distintas?. Vemos que tienen distintos comportamientos: Mi héroe se mueve gracias al teclado, mis enemigos se mueven tontamente y nuestros diparos acaban con ellos. Por eso ¿qué mejor que definir una clase para cada uno de ellos? Es lo que tiene la OOP, que en cuanto te acostumbras a ver Objetos, ves Clases por todas partes.

Lo que ocurre es que Flash es como mi madre, nos obliga a tener “ordenado nuestro cuarto”. NO se puede definir varias clases en el mismo fichero .as, así que tenemos que tner en cuenta que:

- Para cada clase, deberemos definir un fichero .as con el mismo nombre que la clase
- En la bilbioteca le indicarmos el nombre de la clase
- Y tendremos una clase que controle la película principal.

Así que abramos un nuevo .fla, creemos un símbolo en la biblioteca que se llame *Pelota* y en el nombre de la clase pongamos *Pelota*
Hemos dicho que nuestro .as se ha de llamas *Pelota.as* y debe ser algo parecido a

```
package {
    import flash.events.*;
    import flash.display.*;
```

```
    public class Pelota extends MovieClip {
            function Pelota() {
                    addEventListener(Event.ENTER_FRAME,onEnter_Frame,false,0,true);
            }
            function onEnter_Frame(e:Event) {
                    e.target.x++;
            }
    }
}
```

Y hagamos un .as que se llame pelotaPeli.as y asignemos ese nombre de clase a nuestra película.

Ya “tenemos los cojones pelados” con nuestra clase pelotaPeli, así que nada más crear el .as ya hemos escrito del tirón

```
package {
    import flash.display.*;

    public class pelotaPeli extends MovieClip {
            function pelotaPeli() {
            }
    }
}
```

Como sabemos, a la función pelotaPeli se la va a llamar inmediatamente, así que en esa función voy a hacer un attachMovieClip y “listo el pollo” que decía mi amigo Ernesto.

Vale, ya no existe attachMovie, ahora lo que hacemos es añadir “hijos” a nuestra película. Pero antes debemos decirle qué hijo le vamos a añadir

```
package {
    import flash.display.*;

    public class pelotaPeli extends MovieClip {
            function pelotaPeli() {
                    var pelota:Pelota=new Pelota();
                    addChild(pelota);
            }
    }
}
```

Sí, observemos que vamos a añadir un “hijo” de la clase “Pelota”, luego tenemos que “crearlo antes” con un new(). Como AS es “fuertemente tipado”, hay que indicarle que nuestra variable es del tipo “Pelota”.

Vale, queremos que nuestra pelota vaya más rápido, así que podríamos pensar en escribir algo como

```
package {
    import flash.events.*;
    import flash.display.*;
    public class Pelota extends MovieClip {
            function Pelota() {
                    addEventListener(Event.ENTER_FRAME,onEnter_Frame,false,0,true);
            }
            function onEnter_Frame(e:Event) {
                    e.target.x+=e.target.velx;
            }
    }
}
```

```
package {
    import flash.display.*;

    public class pelotaPeli extends MovieClip {
            function pelotaPeli() {
            var pelota:Pelota=new Pelota();
```

```
                pelota.velx=5;
                addChild(pelota);
            }
    }
}
```

¡oh! Error :"acceso a una propiedad posiblemente no definida"

Vale, es un defecto de la OOP, si no declaramos una variable, no la podemos usar. Puesto que la variable es de la clase Pelota, debemos definirla.

```
package {
    import flash.events.*;
    import flash.display.*;
    public class Pelota extends MovieClip {
            public var velx:int;
            function Pelota() {
                addEventListener(Event.ENTER_FRAME,onEnter_Frame,false,0,true);
            }
            function onEnter_Frame(e:Event) {
                e.target.x+=e.target.velx;
            }
    }
}
```

La hemos hecho "pública" para que podamos cambiar el valor de la velocidad de cualquier "pelota" en la Clase pelotaPeli.

Ahora, si queremos, podríamos hacer que nuestra pelotaPeli dibujara multitud de pelotas

```
package {
    import flash.display.*;

    public class pelotaPeli extends MovieClip {
            function pelotaPeli() {
                for (var i:int=0;i<30;i++){
                var pelota:Pelota=new Pelota();
                pelota.velx=Math.floor(10*Math.random())+1;
                pelota.y=350*Math.random();
                addChild(pelota);
                }
            }
    }
}
```

Y si no es por la pequeña patada que le acabo de dar a la OOP este primer esbozo estaría completo.

Hagamos las variables privadas o el motivo por el que en AS desaparecieron los "_" de las propiedades de los MovieClip

En OOP se tiene a la tendencia a que todas las variables sean "privadas". Entonces, ¿cómo accedemos a una propiedad de nuestra "pelota"?. Pues no queda más que hacerse una función

```
package {
    import flash.display.*;

    public class pelotaPeli extends MovieClip {
            function pelotaPeli() {
                for (var i:int=0;i<30;i++){
                var pelota:Pelota=new Pelota();
                pelota.init(Math.floor(10*Math.random())+1);
                pelota.y=350*Math.random();
                addChild(pelota);
                }
            }
    }
```

```
}
```

```
package {
	import flash.events.*;
	import flash.display.*;
	public class Pelota extends MovieClip {
		private velx:int;
		function Pelota() {
			addEventListener(Event.ENTER_FRAME,onEnter_Frame,false,0,true);
		}
		public function init(velx:int){
			this.velx=velx;
		}
		private function onEnter_Frame(e:Event) {
			e.target.x+=e.target.velx;
		}
	}
}
```

Vale, como mi teclado ya tiene las teclas de los paréntesis gastadas, en AS.3 existen unos métodos especiales que no se llaman con función. Los métodos “get y set”. Si definimos unas funciones como

```
		public function set _velx(valor:int){
			velx=valor;
		}
		public function get _velx(){
			return (velx)
		}
```

ya podemos usar la notación de punto para darles valor

```
for (var i:int=0;i<30;i++){
		var pelota:Pelota=new Pelota();
		pelota._velx=(Math.floor(10*Math.random())+1);
		pelota.y=350*Math.random();
		addChild(pelota);
	}
```

Tareas para mañana

- Acabar de leer el libro del puto zoo de Moock
- Ver cómo se hace un preloader en AS.3
- Hacer un visor de imágenes
- Retomar mi arcade “mata enemigos”
- Dormir más