
Kara Effector 3.2:

En el **Tomo IX** seguiremos profundizando en las librerías del **Kara Effector**, ya que eso nos dará las herramientas necesarias para aumentar nuestro nivel al momento de hacer un Efecto.

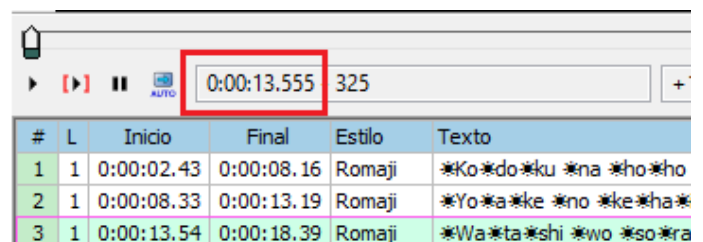
El **Kara Effector** tiene muchas funciones, pero no todas ellas son para hacer Efectos. Hay algunas funciones que nos sirven de apoyo para crear otras nuevas, también hay funciones que hacen que el **Kara Effector** pueda llevar a cabo su tarea y hay otras que nos ayudan en la generación de los Efectos.

En las librerías que veremos a continuación hay de todo tipo de funciones y para todos los gustos. Como siempre digo, a la final todo depende de qué queremos hacer para tener claro por cuál función no decidiremos o cuál es la que mejor se ajusta a nuestro proyecto.

Librería:

Funciones de Tiempo [KE]

HMS_to_ms(Time_HMS): esta función convierte un tiempo dado, de formato **HMS** (horas, minutos y segundos) a formato ms (milisegundos). El tiempo **HMS** debe ser ingresado entre comillas, ya sean comillas sencillas o dobles. Ejemplo:



The screenshot shows the Kara Effector software interface. At the top, there is a control bar with play, stop, and a time input field. The time input field is highlighted with a red box and contains the text "0:00:13.555". To the right of the time input field is a numerical input field containing "325". Below the control bar is a table with the following data:

#	L	Inicio	Final	Estilo	Texto
1	1	0:00:02.43	0:00:08.16	Romaji	*Ko*do*ku *na *ho*ho
2	1	0:00:08.33	0:00:13.19	Romaji	*Yo*a*ke *no *ke*ha*
3	1	0:00:13.54	0:00:18.39	Romaji	*Wa*ta*shi *wo *so*ra

Copiamos el tiempo que está dentro del recuadro rojo en la anterior imagen, y lo pegamos dentro de la función, sin olvidar colocarlo entre comillas:

Add Tags: Add Tags Language: Lua

```
HMS_to_ms( "0:00:13.555" )
```

Y la función convertirá ese tiempo que estaba en formato **HMS** a formato ms, cuyo resultado cuenta como un valor numérico:

☐ Comentar Romaji lead-in 2

0 0:00:02.43 0:00:08.16 0:00:05.73 0 0 0

B **I** **U** **S** **fn** **AB** **AB** **AB** **AB** ☒ Tiempo ☐ Ct

```
{Kara Effector[fx] 3.2: ABC Template \an5\pos(269.97,  
314.1113555)Ko
```

O sea que: **HMS_to_ms("0:00:13.55") = 13555 ms**

ms_to_HMS(Time_ms): esta función convierte un tiempo dado, de formato ms (milisegundos) a formato **HMS** (horas, minutos y segundos). El tiempo ms debe ser ingresado como un valor numérico, sin comillas. Ejemplo:

Add Tags: Add Tags Language: Lua

```
ms_to_HMS( 274178 )
```

Obtenemos:

☐ Comentar Romaji lead-in 2

0 0:00:02.43 0:00:08.16 0:00:05.73 0 0 0

B **I** **U** **S** **fn** **AB** **AB** **AB** **AB** ☒ Tiempo ☐ Ct

```
{Kara Effector[fx] 3.2: ABC Template \an5\pos(269.97,  
314.110:04:34.178)Ko
```

time_to_frame(time): convierte un tiempo dado, ya sea en formato **HMS** o ms, en la cantidad de "frames" (cuadros) que ocuparía ese tiempo en el vídeo que se está usando para aplicar un Efecto.

Ejemplo con tiempo en formato ms (milisegundos):

Add Tags: Add Tags Language: Lua

```
time_to_frame(2728)
```

☐ Comentar Romaji lead-in 2

0 0:00:02.43 0:00:08.16 0:00:05.73 0 0 0

B **I** **U** **S** **fn** **AB** **AB** **AB** **AB** ☒ Tiempo ☐ Ct

```
{Kara Effector[fx] 3.2: ABC Template \an5\pos(269.97,  
314.1166)Ko
```

Ejemplo con tiempo en formato **HMS**:

Add Tags: Add Tags Language: Lua

```
time_to_frame('0:01:32.486')
```

☐ Comentar Romaji lead-in 2

0 0:00:02.43 0:00:08.16 0:00:05.73 0 0 0

B **I** **U** **S** **fn** **AB** **AB** **AB** **AB** ☒ Tiempo ☐ Ct

```
{Kara Effector[fx] 3.2: ABC Template \an5\pos(269.97,  
314.112218)Ko
```

De lo anterior se concluye que 2728 ms equivalen a 66 frames (cuadros), y que en "0:01:32:486" (1 minuto, 32 segundos y 486 ms) hay 2218 frames.

frame_to_ms(frames): esta función convierte el número de la cantidad de frames a ms (milisegundos). La cantidad de frames debe ser ingresada como un valor numérico, sin las comillas, ejemplo:

Add Tags:
Add Tags Language:
Lua

frame_to_ms(150)

☐ Comentar
Romaji
lead-in
2

0
0:00:02.43
0:00:08.16
0:00:05.73
0
0
0

B
I
U
-
fn
AB
AB
AB
AB
Tiempo
Ct

{Kara Effector[fx] 3.2: ABC Template \an5\pos(269.97, 314.11) 6257} Ko

frame_to_HMS(frames): esta función convierte el número de la cantidad de frames a un tiempo en formato **HMS** (horas, minutos y segundos). La cantidad de frames debe ser ingresada como un valor numérico, sin las comillas, ejemplo:

Add Tags:
Add Tags Language:
Lua

frame_to_HMS(150)

☐ Comentar
Romaji
lead-in
2

0
0:00:02.43
0:00:08.16
0:00:05.73
0
0
0

B
I
U
-
fn
AB
AB
AB
AB
Tiempo
Ct

{Kara Effector[fx] 3.2: ABC Template \an5\pos(269.97, 314.11) 0:00:06.257} Ko

time_mid1(delay): esta función, dependiendo el tipo de Efecto (este y los siguientes ejemplos están hechos con **Template Type: Syl**, pero se puede usar para todos los tipos menos el Line y Template Line), si se usa en el tiempo de inicio de la Línea, hace que las sílabas aparezcan de forma progresiva, desde los extremos hacia en centro de la Línea, como se puede apreciar en el siguiente ejemplo:

Line Start Time =
l.start_time + time_mid1(50)

→

Kodoku

←

su kedo

→

こどく

←

な ほほ を めらす めらす けど

Esta función adicionada al tiempo de inicio de la Línea, nos sirve para hacer efectos **lead-in** (efectos de entrada). El tiempo que separa la aparición de una sílaba con respecto de la otra, es el **delay** (retraso), que para este ejemplo fue de 50 ms:

25	0	0:00:02.89	0:00:08.16	Romaji	lead-in	Effector [Fx]	*Ko
26	0	0:00:02.94	0:00:08.16	Romaji	lead-in	Effector [Fx]	*do
27	0	0:00:02.99	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ku
28	0	0:00:03.04	0:00:08.16	Romaji	lead-in	Effector [Fx]	*na
29	0	0:00:03.09	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ho
30	0	0:00:03.14	0:00:08.16	Romaji	lead-in	Effector [Fx]	*ho

Lo que muestra la imagen anterior es que la diferencia entre 0:00:02.890 y 0:00:02.940 es de 50 ms, que fue el **delay** usado para este ejemplo. Y como ya se había dicho antes, esta función también puede ser usada con los **Template Type**: Furi, Char, Translation Char y Translation Word; con resultados similares.

A mayor **delay**, mayor será el tiempo en que aparezca una sílaba (carácter o palabra, dependiendo del Template Type) con respecto de la inmediatamente anterior. O sea que uno decide el retraso de la función dependiendo del resultado que queremos obtener.

La función **time_mid1(delay)** usada en el tiempo final de la Línea, nos ayuda a hacer efectos **lead-out** (de salida) y hace que las sílabas desaparezcan desde los extremos hacia el centro de la línea, ejemplo:

Line End Time =
l.end_time + time_mid1(50)

→

doku na hoho wo nurasu nurasu ke

←

よあけのけはいがしずかにみちて

O sea que la función **time_mid1(delay)** nos sirve para hacer efectos, tanto **lead-in** como **lead-out**, dependiendo en qué tiempo de la Línea la usemos: **Line Start Time** o **Line End Time**, tiempo de inicio y final, respectivamente.

time_mid2(delay): esta función, dependiendo el tipo de Efecto, si se usa en el tiempo de inicio de la Línea, hace que las sílabas aparezcan de forma progresiva, desde el centro hacia los extremos de la Línea. Es la función opuesta a **time_mid1**, como se puede apreciar en el siguiente ejemplo:

Line Start Time =



También se puede usar esta función en los Template Type: Furi, Char, Translation Char y Translation Word

La función **time_mid2(delay)** usada en el tiempo final de la Línea, nos ayuda a hacer efectos **lead-out** (de salida) y hace que las sílabas desaparezcan desde el centro hacia los extremos de la Línea. Ejemplo:

Line End Time =



time_li(delay): esta función es similar a las dos funciones anteriormente vistas. Esta función se puede sumar o restar al tiempo de inicio de la Línea y por eso su nombre: **time_li (time lead-in)**. Y al igual que las dos anteriores funciones, se puede usar en los **Template Type**: Syl, Furi, Char, Template Char y Translation Word. Para este ejemplo usaré **Template Type**: Syl

Modo suma: hace que las sílabas aparezcan de forma progresiva de izquierda a derecha en la Línea.

Line Start Time =



Modo resta: hace que las sílabas aparezcan de forma progresiva de derecha a izquierda en la Línea.

Line Start Time =



time_lo(delay): es similar a **time_li**. Esta función se puede sumar o restar al tiempo final de la Línea y por eso su nombre: **time_lo (time lead-out)**. Y al igual que las anteriores funciones, se puede usar en los Template Type: Syl, Furi, Char, Template Char y Translation Word.

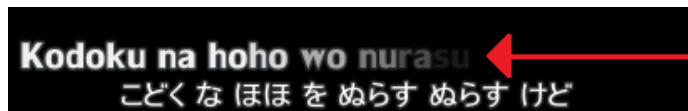
Modo suma: hace que las sílabas desaparezcan de forma progresiva de izquierda a derecha en la Línea.

Line End Time =



Modo resta: hace que las sílabas desaparezcan de forma progresiva de derecha a izquierda en la Línea.

Line End Time =



Con esta función culmina la librería de las funciones de tiempo del Kara Effector y da paso a la próxima librería.

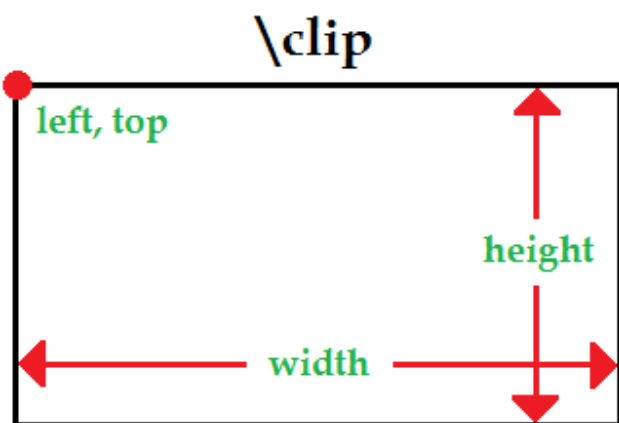
Librería: tag [KE]

Esta librería contiene una serie de funciones enfocadas en los Tags que usamos a la hora de hacer Efectos.

tag.clip(left, top, width, height, mode): crea uno o más clip's rectangulares dependiendo del **loop** asignado, con posición y medidas específicas. Los cinco parámetros a ingresar en la función son opcionales, ya que cada uno de ellos tiene valor por default en caso de ser necesario.

left y top son las coordenadas 'x' y 'y' respectivamente del punto de origen del clip, que es el punto superior izquierdo del rectángulo que lo generará.

width y height son el ancho y el alto del rectángulo que generará al clip, respectivamente:



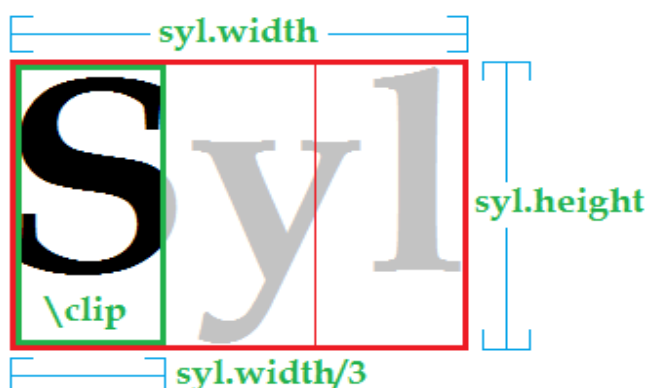
Veamos algunos ejemplos para empezar a aclarar los conceptos hasta acá vistos:

```
Add Tags: Add Tags Language: Lua
tag.clip( syl.left, syl.top, syl.width/3, syl.height )
```

- **left** = syl.left
- **top** = syl.top
- **width** = syl.width/3
- **height** = syl.height



Solo queda visible un tercio del ancho de la sílaba, ya que para el ejemplo se usó **syl.width/3** como el ancho del clip:



Si nuestro efecto es un **Template Type: Syl**, y queremos que toda la sílaba sea totalmente visible dentro del clip, debemos usar los siguientes valores:

```
Add Tags: Add Tags Language: Lua
tag.clip( syl.left, syl.top, syl.width, syl.height )
```

Los anteriores valores también son los valores por default en el **Template Type: Syl**, o sea que lo podemos usar con el mismo resultado de la siguiente forma:

```
Add Tags: Add Tags Language: Lua
tag.clip( )
```

Es decir, que los valores por default que usará la función dependen del Template Type.

Para Template Type: **Line** y **Translation Line**, los valores por default de **tag.clip()** son: line.left, line.top, line.width y line.height

Para el Template Type: **Translation Word**, los valores por default de **tag.clip()** son: word.left, word.top, word.width y word.height

Para el Template Type: **Syl**, los valores por default de **tag.clip()** son: syl.left, syl.top, syl.width y syl.height

Para el Template Type: **furi**, los valores por default de **tag.clip()** son: furi.left, furi.top, furi.width y furi.height

Y para Template Type: **Char** y **Translation Char**, los valores por default de **tag.clip()** son: char.left, char.top, char.width y char.height

Hasta este punto pareciera que la función **tag.clip** no tiene nada de especial, o al menos que no tiene algo distinto a lo que el tag “\clip” podría hacer por sí solo, pero es aquí en donde la celda de texto “loop” entra en escena y nos muestra las ventajas de la función, dependiendo de los resultados que deseemos en nuestros efectos.

A continuación veremos las distintas combinaciones que podemos hacer con la función **tag.clip** a partir de los valores que usemos en la celda de texto “**loop**”. Ejemplo:

- **loop**: 4
- Template Type: **Translation Word**

loop = 4

Add Tags: Add Tags Language: Lua

tag.clip()

La función **tag.clip** creará 4 clip's horizontales dentro de rectángulo, según los parámetros que hayamos ingresado a la función, en este ejemplo está por default y como es un Template Type: **Translation Word**, entonces tiene las dimensiones exactas de cada una de las palabras en cada una de las líneas:



Con solo colocar cualquier valor en la celda de texto “**loop**” obtendremos tantos **clip's horizontales** como los necesitemos. Para el siguiente ejemplo veremos cómo obtener clip's verticales con la función **tag.clip**

- **loop**: 1, 3
- Template Type: **Char**

loop = 1, 3

Add Tags: Add Tags Language: Lua

tag.clip()



Lo que hace el loop usado de ese modo es que la función **tag.clip** cree **clip's verticales** dentro del rectángulo hecho por las medidas ingresadas y como es un **Template Type**: Char y usamos los valores por default de la función, hará que cada carácter quede dentro de los tres clip's creados.

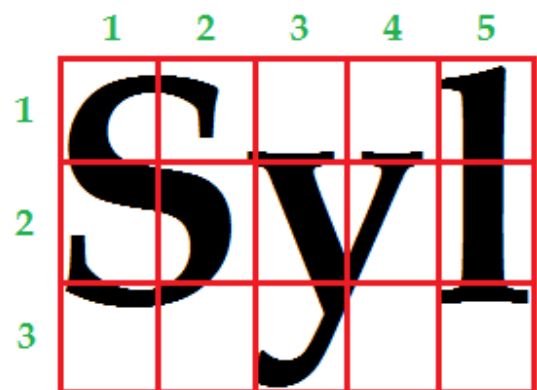
En el próximo ejemplo veremos la forma de hacer **clip's reticulares** (como en forma de grilla o cuadrícula):

- **loop**: 3, 5
- Template Type: **Syl**

loop = 3, 5

Add Tags: Add Tags Language: Lua

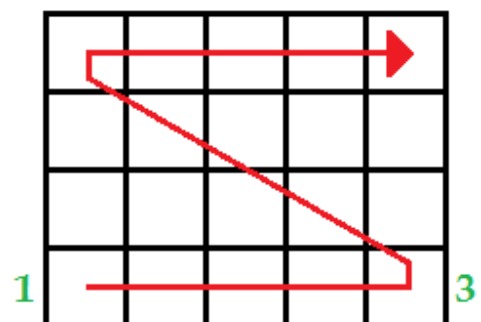
tag.clip()



La retícula creada por la función **tag.clip** será de 3 clip's horizontales por 5 verticales y la cantidad total del **loop** será: $3 \times 5 = 15$

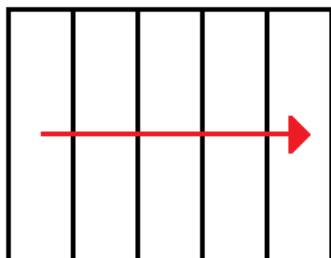
El parámetro **mode** en la función **tag.clip** es un número que asignamos con el fin de determinar el orden de los clip's. Son **8 modes** y los veremos a cada uno de ellos:

Modo: 13

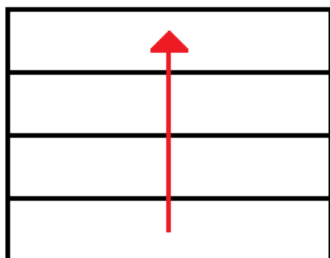


- **modo 13:** la imagen anterior muestra el orden de los clip's en el **modo 13** de un **tag.clip** reticulado. En el caso de los clip's verticales, el **modo 13** hace que el primer clip sea el del extremo izquierdo y el último el del extremo derecho. Para los clip's horizontales el primero sería el del extremo inferior y el último el del extremo superior:

Modo: 13



Modo: 13



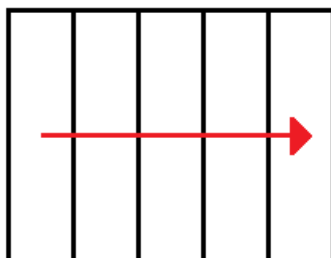
- **modo 17:** para un **tag.clip** reticulado, el orden de los clip's es como el de la siguiente imagen, es decir, el primer clip es el de la esquina inferior izquierda y el último es el de la esquina superior derecha, siguiendo la trayectoria de la gráfica:

Modo: 17

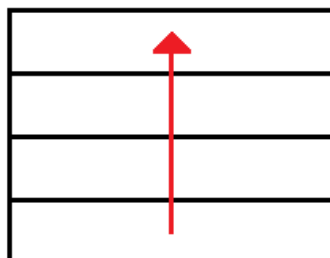


En el caso de los clip's verticales, el **modo 17** hace que el primer clip sea el del extremo izquierdo y el último el del extremo derecho. Para los clip's horizontales el primero sería el del extremo inferior y el último el del extremo superior:

Modo: 17

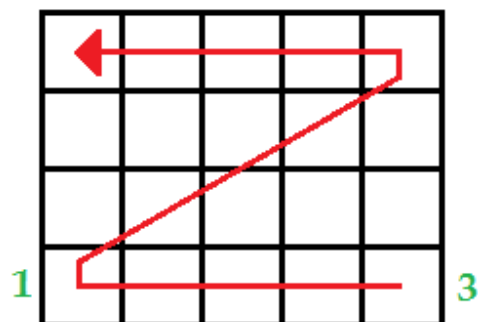


Modo: 17



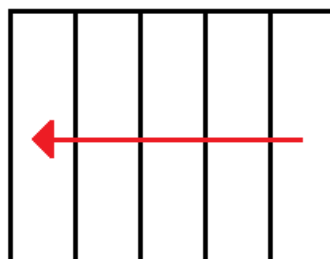
- **modo 31:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina inferior derecha y el último es el de la esquina superior izquierda, siguiendo la trayectoria de la gráfica:

Modo: 31



En el caso de los clip's verticales, el **modo 31** hace que el primer clip sea el del extremo derecho y el último el del extremo izquierdo. Para los clip's horizontales el primero sería el del extremo inferior y el último el del extremo superior:

Modo: 31

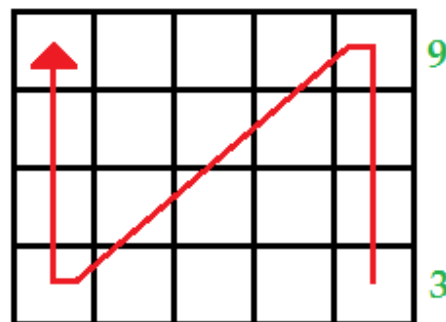


Modo: 31



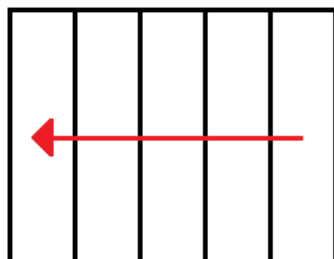
- **modo 39:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina inferior derecha y el último es el de la esquina superior izquierda, siguiendo la trayectoria de la gráfica:

Modo: 39

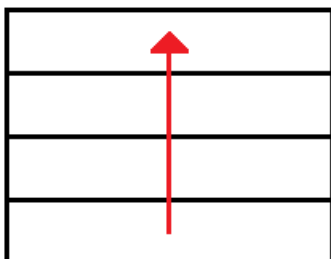


En el caso de los clip's verticales, el **modo 39** hace que el primer clip sea el del extremo derecho y el último el del extremo izquierdo. Para los clip's horizontales el primero sería el del extremo inferior y el último el del extremo superior:

Modo: 39

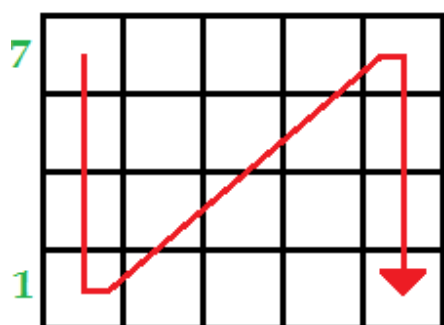


Modo: 39



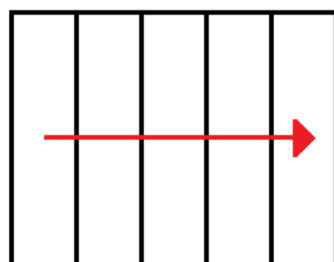
- **modo 71:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina superior izquierda y el último es el de la esquina inferior derecha, siguiendo la trayectoria de la gráfica:

Modo: 71

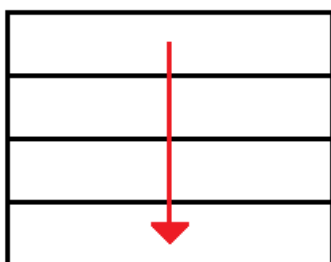


En el caso de los clip's verticales, el **modo 71** hace que el primer clip sea el del extremo izquierdo y el último el del extremo derecho. Para los clip's horizontales el primero sería el del extremo superior y el último el del extremo inferior:

Modo: 71



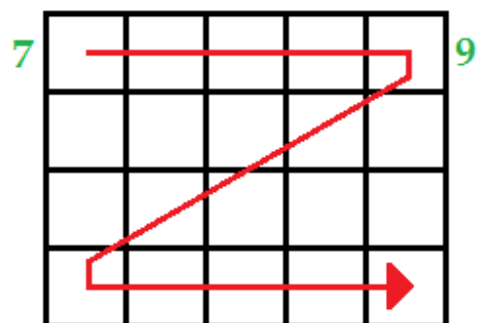
Modo: 71



- **modo 79:** es el modo por default de la función. para un **tag.clip** reticulado, el orden de los clip's

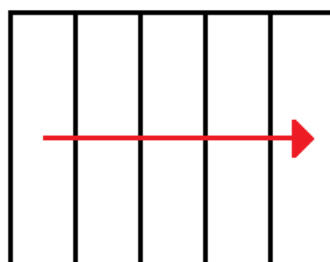
sería el siguiente; el primer clip es el de la esquina superior izquierda y el último es el de la esquina inferior derecha, siguiendo la trayectoria de la gráfica:

Modo: 79

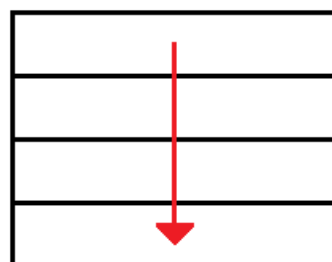


En el caso de los clip's verticales, el **modo 79** hace que el primer clip sea el del extremo izquierdo y el último el del extremo derecho. Para los clip's horizontales el primero sería el del extremo superior y el último el del extremo inferior:

Modo: 79

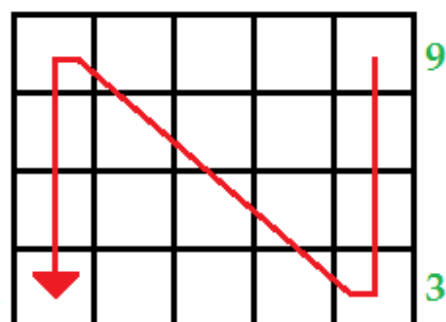


Modo: 79



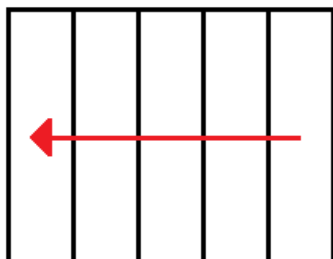
- **modo 93:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina superior derecha y el último es el de la esquina inferior izquierda, siguiendo la trayectoria de la gráfica:

Modo: 93

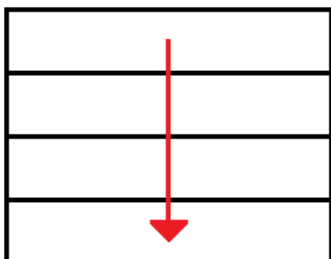


En el caso de los clip's verticales, el **mode 93** hace que el primer clip sea el del extremo derecho y el último el del extremo izquierdo. Para los clip's horizontales el primero sería el del extremo superior y el último el del extremo inferior:

Modo: 93

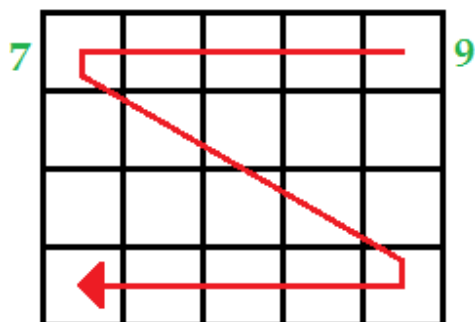


Modo: 93



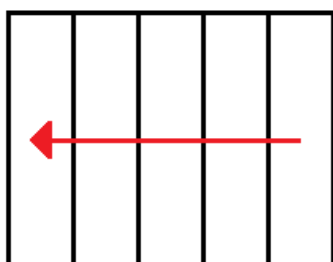
- **mode 97:** para un **tag.clip** reticulado, el orden de los clip's sería; el primer clip es el de la esquina superior derecha y el último es el de la esquina inferior izquierda, siguiendo la trayectoria de la gráfica:

Modo: 97

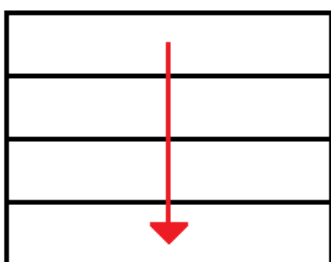


En el caso de los clip's verticales, el **mode 97** hace que el primer clip sea el del extremo derecho y el último el del extremo izquierdo. Para los clip's horizontales el primero sería el del extremo superior y el último el del extremo inferior:

Modo: 97



Modo: 97



Y el **mode 97** sería el último de ellos. Así que hay para elegir según sea nuestra necesidad en un Efecto.

Recordemos que el número **mode** lo escribiremos dentro de la función **tag.clip** como un valor numérico, ejemplos:

```
tag.clip( fx.pos_l, fx.pos_t, syl.width, syl.height, 31 )
```

```
tag.clip( fx.pos_l, fx.pos_t, char.width, char.height, 97 )
```

```
tag.clip( fx.pos_l - 50, fx.pos_t, l.width + 100, l.height, 17 )
```

Es todo por el momento y damos por terminado el **Tomo IX**. En el **Tomo X** del **Kara Effector** continuaremos no solo con la función **tag.clip** sino también con el resto de la Librería "**tag**", ya que vale la pena dedicarle más de tiempo y espacio al estudio de estas funciones y todos los efectos que podemos hacer con ellas.

Intenten poner en práctica todos los ejemplos vistos en este **Tomo** y no olviden descargar última actualización disponible del **Kara Effector 3.2** y visitarnos en el **Blog Oficial** lo mismo que en los canales de **YouTube** para descargar los nuevos Efectos o dejar algún comentario, exponer alguna duda o hacer alguna sugerencia.
