

18. Scroll

Tienes muchísimo que experimentar con todo lo que hemos visto hasta ahora, sin lugar a dudas ya debes tener un buen concepto de en qué consiste el mundo de la programación.

Pero por supuesto nos sigue quedando muchísimo por aprender. Podemos tener miles de procesos distintos en pantalla, ejecutando sus instrucciones y reproduciendo los comportamientos que hemos programado para ellos, pero: ¿Que hay del mundo en el que se mueven?

El scroll es un aspecto fundamental para nuestro videojuego y Benu nos ofrece un soporte excelente para utilizarlo. A continuación veremos cuáles son todas sus posibilidades.

18.1 El vector de types scroll[]

¿Por qué hemos esperado hasta el tema 18 para hablar del scroll?

No es porque se trate de un tema muy complejo, realmente el uso de un scroll no nos supondrá una complicación adicional y por supuesto ofrecerá un aspecto mucho más sofisticado a nuestro videojuego.

Si hemos esperado hasta ahora es porque resulta muy recomendable estar al tanto de todos los conceptos que hemos visto sobre variables antes de ponernos a trabajar con el scroll. El concepto de scroll en Benu hace un sencillo uso de los vectores y de los types.

Benu nos brinda la posibilidad de tener varios scroll en pantalla. Ésto nos permitiría tener uno para el minimapa, varios para un videojuego en pantalla partida, o cualquier cosa que se nos pueda ocurrir en ese sentido.

Un scroll no es más que un gráfico que se desplaza a través de una determinada región de pantalla, y Benu tiene un type asociado para gestionar esta idea. De hecho tiene un vector de types con el que podremos gestionar hasta 10 scroll distintos.

Cada uno de esos scroll es una estructura almacenada en cada una de las posiciones de un vector llamado scroll []. Ese vector se comporta como GLOBAL, es decir, es común a todos los procesos y podemos referirnos a él en cualquier punto de nuestro código.

A continuación vamos a estudiar un poco más a fondo las diferencias que supone para nuestro videojuego el hecho de que tenga uno o más scrolls en pantalla.

18.2 Coordenadas absolutas y relativas

Hasta ahora mismo habíamos visto cómo las coordenadas x,y de cualquier proceso tenían como base la esquina superior izquierda de la pantalla.

Esa base resulta muy útil cuando trabajamos con procesos cuyo posicionamiento es relativo a la pantalla. ¿Pero qué ocurre si queremos programar un videojuego con scroll en el cual la posición de nuestros procesos no depende solamente de la pantalla, sino que depende de un gigantesco scroll.

Para que comprendas mejor de lo que estamos hablando: Piensa que la coordenada $x = 2000$ se encuentra fuera de pantalla, pero dentro de un scroll de más de 10000 píxeles de anchura sería posible que nuestro protagonista se encontrase en esa coordenada, y por supuesto debería seguir mostrándose en el centro de la pantalla.

Esto nos sugiere la idea de que podremos mantener centrada una cámara en nuestro proceso protagonista, de forma que siempre permanezca visible en pantalla, sea cual sea su posición en el scroll. Es algo que vamos a estudiar a fondo.

18.3 Los campos de scroll[]

Como hemos mencionado anteriormente, sin necesidad de haber declarado ninguna variable previamente, Bennu tiene predefinida una variable GLOBAL llama scroll que en realidad es un vector que almacena la información de 10 scroll distintos.

Obviamente por el momento tenemos suficiente con un scroll, así que por ahora trabajaremos exclusivamente con scroll [0], el primero de la lista, hasta que tengamos capacidad suficiente para sacar partido a varios scroll simultáneos.

Cada una de las componentes de ese vector, en nuestro caso la componente 0, es un type cuyos campos son los siguientes:

Campo	Tipo	Utilidad
Scroll [0] . x0	Int	Almacena la coordenada x del gráfico de scroll que actualmente se muestra en la posición $x = 0$ de la pantalla. Si hacemos uso del campo camera no tendrá mucho sentido modificar este valor, tan sólo lo consultaremos.
Scroll [0] . y0	Int	Almacena la coordenada y del gráfico de scroll que actualmente se muestra en la posición $y = 0$ de la pantalla. Si hacemos uso del campo camera no tendrá mucho sentido modificar este valor, tan sólo lo consultaremos.

Campo	Tipo	Utilidad
Scroll [0] . xl	Int	Almacena la coordenada x del gráfico de scroll de fondo que actualmente se muestra en la posición x = 0 de la pantalla, con cierto retraso respecto del gráfico principal de scroll. Si hacemos uso del campo camera no tendrá mucho sentido modificar este valor, tan sólo lo consultaremos.
Scroll [0] . yl	Int	Almacena la coordenada y del gráfico de scroll de fondo que actualmente se muestra en la posición y = 0 de la pantalla, con cierto retraso respecto del gráfico principal de scroll. Si hacemos uso del campo camera no tendrá mucho sentido modificar este valor, tan sólo lo consultaremos.
Scroll [0] . camera	int	Almacena el identificador del proceso que queremos que permanezca siempre en el centro de la pantalla, generalmente, el protagonista. Cuando modifiquemos las coordenadas x,y de ese proceso el scroll se desplazará. El uso de un proceso centrado sustituye al uso de los campos x0, y0, xl, yl y en general resulta más cómodo.
Scroll [0] . flags1	int	Tiene la misma utilidad que el aspecto flags de cualquier proceso en Bennu, sólo que aplicado al scroll principal.
Scroll [0] . flags2	int	Tiene la misma utilidad que el aspecto flags de cualquier proceso en Bennu, sólo que aplicado al scroll de fondo.
Scroll [0] . ratio	Int	Indica el porcentaje de avance en el desplazamiento del scroll de fondo respecto del scroll principal. Su valor por defecto es 200 y consigue una sensación de profundidad, que es lo que se desea generalmente a la hora de establecer un scroll con fondo.

18.4 Inicializar un scroll

El vector de types tiene una complejidad superior a lo que estamos acostumbrados. Afortunadamente, Bennu nos ofrece algunas funciones para inicializar un scroll sin necesidad de tener que manipular directamente los valores almacenados en scroll [].

La principal función para el tratamiento de scrolls es start_scroll (), cuya sintaxis es la siguiente:

```
int start_scroll ( int , int , int , int , int , int )
```

Observa en primer lugar que start_scroll () es una función que nos retorna un valor numérico, su utilidad es muy importante y la explicaremos al final de este tema.

En cuanto a los parámetros que recibe `start_scroll ()`, aunque casi siempre los usaremos de la misma manera, tienen la siguiente utilidad:

Parámetro	Tipo	Utilidad
1	Int	<p>Indica el índice del vector scroll [] que vamos a inicializar.</p> <p>Recuerda que disponemos de índices del 0 al 9, pero por seguir un convenio comenzaremos inicializando el scroll 0, así que en general pondremos un 0 como valor para este parámetro.</p>
2	Int	<p>Indica el fichero .fpg del que vamos a obtener los gráficos del scroll.</p> <p>Actualmente sólo tenemos un único fichero .fpg en el que se encuentran todos nuestros gráficos. Para usarlo pondremos un 0 como valor para este parámetro, así haremos uso del último fichero .fpg cargado.</p> <p>Si utilizamos varias funciones <code>load_fpg ()</code> y almacenamos su valor de retorno en una variable, podemos poner dicha variable en este parámetro y así leer los gráficos de un .fpg distinto. Esto último lo explicaremos a fondo más adelante.</p>
3	int	<p>Indica el número de gráfico dentro del fichero .fpg que queremos establecer como gráfico principal del scroll.</p> <p>Si seguimos el convenio de numeración explicado en los 2 primeros temas de este tutorial, pondremos 501 como valor para este parámetro, ya que decidimos numerar los scrolls con los valores comprendidos entre el 501 y el 700.</p>
4	int	<p>Indica el número de gráfico dentro del fichero .fpg que queremos establecer como gráfico de fondo para el scroll. Para que este gráfico sea visible es imprescindible que el gráfico de scroll principal tenga transparencia.</p> <p>Si seguimos el convenio de numeración explicado en los 2 primeros temas de este tutorial, pondremos 502 como valor para este parámetro, ya que decidimos numerar los scrolls con los valores comprendidos entre el 501 y el 700.</p>
5	int	<p>Indica la región de pantalla en la cual inicializaremos el scroll. Todavía no hemos creado regiones de pantalla y es algo que veremos más adelante, así que por el momento pondremos un 0 como valor para este parámetro. Así conseguiremos que el scroll se inicie en la región de pantalla por defecto, que es la pantalla al completo.</p>

Parámetro	Tipo	Utilidad
6	Int	<p>El número introducido en este parámetro determina opciones que afectan al funcionamiento del scroll. De momento pondremos un 0 como valor para este parámetro y su comportamiento será normal, pero también podemos poner la suma de uno o más valores siguientes:</p> <ul style="list-style-type: none"> 1 - El scroll principal se repite horizontalmente. 2 - El scroll principal se repite verticalmente. 4 - El scroll de fondo se repite horizontalmente. 8 - El scroll de fondo se repite verticalmente.

Con los parámetros sugeridos en la tabla anterior pondremos en marcha un scroll por defecto, más adelante podremos modificar los valores según lo propuesto para conseguir diferentes efectos.

En cuanto al valor de retorno de `start_scroll ()`, haremos uso del mismo para determinar cuál será el proceso en el cual se centrará la cámara de nuestro scroll, como ya habíamos comentado generalmente este proceso será nuestro protagonista.

Todavía no hemos explicado a fondo el funcionamiento de la invocación de procesos en Benu. Si recuerdas funciones como `write ()` o `load_fnt ()`, eran funciones que retornaban un valor numérico con el identificador bien sea del texto o bien de la fuente de texto cargada. Con los procesos podemos proceder de la misma forma, si asignamos a una variable de tipo `int` la invocación de un proceso, almacenaremos en la variable el identificador de ese proceso.

Por ejemplo si modificamos así nuestro código:

```

GLOBAL
    int ID_protagonista;           //Declaramos una variable
BEGIN
    ...
    ID_protagonista = protagonista ( );   //Almacenamos en ella el identificador del proceso
    ...
END
    
```

Conseguiremos invocar al protagonista igual que antes y además almacenaremos en la variable `ID_protagonista` su identificador de proceso.

Si a continuación invocamos a la función `start_scroll ()`, el código quedaría así:

```
GLOBAL
    int ID_protagonista;           // Declaramos una variable
BEGIN
    ...
    ID_protagonista = protagonista ( ); // Almacenamos en ella el identificador del proceso
    start_scroll ( 0 , 0 , 501 , 502 , 0 , 0 ); // Iniciamos el scroll
    scroll [ 0 ].camera = ID_protagonista; // Centramos la cámara del scroll en el proceso
    ...
END
```

El identificador del protagonista será el que quedará centrado en la cámara del scroll que acabamos de invocar.

18.5 Establecer coordenadas relativas al scroll

A partir de ahora todas las coordenadas x,y de los procesos deberán ser relativas al scroll. Bennu nos ofrece una forma sencilla de hacerlo gracias a los aspectos básicos de todo proceso.

Existe un aspecto llamado `ctype`, es de tipo entero, y debemos realizar la siguiente asignación a este aspecto en todo proceso cuyas coordenadas deseemos que sean relativas al scroll y no a la pantalla:

```
BEGIN
    ctype = C_SCROLL;
    ...
END
```

`C_SCROLL` es un valor constante que equivale a 1, en breve explicaremos más a fondo los conceptos de valores constantes en Bennu.

En el caso de los textos tenemos una excepción, ya que las funciones `write ()` y `write_var ()` atienden siempre a coordenadas relativas a pantalla. Convertir una coordenada de proceso a coordenada de pantalla es fácil si recordamos los campos existentes en `scroll [0]`:

```
x - scroll[0].x0 //Nos devuelve la x relativa a pantalla que corresponde a un proceso cuya x es relativa a scroll.
y - scroll[0].y0 //Nos devuelve la y relativa a pantalla que corresponde a un proceso cuya y es relativa a scroll.
```