Técnica del Scroll Vertical en Basic Tower

Cada fase es una cadena de texto que mide 60 filas de alto, por 13 caracteres de ancho. Al final de cada fila, introducimos un RetornoDeCarro, para pasar a la fila siguiente al imprimir la cadena.

Esto nos da un tamaño de 60 *(13+1) = 840 bytes para la cadena de la fase completa.

Esto es cierto, suponiendo que la fase es monocromo, y no hay que incorporar códigos de color.

Entonces, para dibujar el frame de más arriba, empezando en la fila 1, y con un alto de 20 filas, tenemos qque apuntar la cadena z$ (que es el frame que se imprime) al inicio de la fila 1, y asignarle el tamaño que ocupan esas 20 filas , esto es 20*(13+1) = 280 bytes.

Si queremos hacer scroll hacia debajo de una fila, tenemos que apuntar la cadena z$ al inicio de la segunda fila, y asignarle el tamaño de las filas 2 a la 21, otros 280 bytes.

Y si seguimos bajando, sucesivamente hasta la ultima fila, el frame z$ apuntaría al inicio de la fila 41, con un alto de 20 filas de tamaño 280 bytes.

En Basic del spectrum no hay una manera directa de usar punteros. Entonces, para decirle al interprete que una cadena empieza en una dirección de memoria determinada, y ocupa un tamaño concreto, necesitamos hacer uso de DefAdd. Creamos una estructura como en la intruccion DEF FN, en la que se le puede especificar la dirección de inicio de unos datos, y el tamaño que tienen esos datos.

En BasicTower uso 15 variables DefAdd, y es la variable z$ la que apunta al frame con el stringque se imprimirá en pantalla, y para especificar la dirección de inicio hay que pokear las posiciones 38930+4 y +5, y para asignar el tamaño del frame hay que pokearlo en 38930+6 y +7.

Después hacer PRINT at 0,0;z$; y se pinta instantáneamente el frame.(líneas 141 o 167 es donde se hace la magia).

Si metemos códigos de color, fácilmente se puede duplicar el tamaño de cada frame, o incluso más, según la complejidad de cada fila.
Entonces, para intenter reducir la cantidad de bytes a imprimir, utilizo esta optimización:

Mediante un preproceso offline de cada fase, compruebo si la fila a imprimir ha cambiado con respecto al frame anterior, y solo la imprimo hasta el carácter que ha cambiado.

Imagina que hay 5 filas de ladrillos representando un muro alto, desde la fila 10 a la 14. Al hacer scroll, en la fila 9 tendremos que pintar los ladrillos, pero en la 10,11,12 y 13 ya

hay dibujados esos ladrillos, así que podemos saltarnos esas filas, imprimiendo solo el RetornoDeCarro, ahorrando muchísimos bytes, lo que se traduce en velocidad de impresión.

El mencionado preproceso reduce el tamaño de las 60 filas de la cadena con el mapa de la fase a un máximo de 768bytes, y cada frame de 20 lineas aproximadamente ocupa unos 300bytes incluyendo códigos de color.

Otra pequeña optimización es que los 4 pokes que establecen la dirección de inicio de la cadena y el tamaño del frame, los sustituyo por una asignación let también DefAdd que copia rápidamente esos 4 valores. (línea 141).

Como en cada iteración del bucle (línea 110) se imprime la cadena z$ con el frame actual, conseguimos la sensación de animación de los fondos cambiando el banco de UDGs cada vez, variando entre los 4 disponibles.

Otras variables DefAdd que uso son:
3 (i$, y$ y f$) para la cadena que pinta el fantasma,
5 (c$, k$ y g$) para las cadenas de las gárgolas,
3 (p$, q$ y r$) para copiar los 4 bytes de dirección y tamaño del frame,
1 (e$) para la cadena que pinta el personaje y restaura el fondo en su posición anterior
2 (o$ y d$) para copiar toda la info de la fase actual a la zona de trabajo