GRAFICOS Y ANIMACION EN BASIC (I)

Los gráficos constituyen, sin duda, el ingrediente fundamental de un buen juego. Incluso quien no domine el código máquina, puede realizar magnificos gráficos en Basic. En este articulo os vamos a explicar como. Veremos tambien como realizar animaciones sin utilizar para nada el código máquina.

Jesús Alonso Rodriguez

En el Spectrum hay varias formas de realizar gráficos en pantalla. La más sencilla tal vez sea con el empleo de las sentencias PLOT, DRAW y CIRCLE. Se pueden hacer dibujos muy vistosos, pero resulta poco adecuado para crear personajes y objetos en juegos. Tambien es posible emplear los caracteres gráficos pre-definidos (códigos 128 al 143) aunque no brindan muchas posibilidades. Por último, nos queda la opción de redefinir todo el juego de caracteres del ordenador.

Los juegos comerciales suelen estar escritos en código máquina y emplear rutinas de animación de "sprites" para conseguir los efectos de movimiento en pantalla. Sin duda, es la mejor forma de hacerlo, pero eso no quiere decir que sea imposible conseguir buenos gráficos en Basic e, incluso, hacer animaciones. En este artículo os lo vamos a demostrar. Prometemos no emplear ni una sola instrucción en código máquina.

Empecemos por lo facil

La forma más facil de hacer gráficos es definirlos en los UDGs ("User Defined Grafics" o Gráficos Definidos por el Usuario) que para eso están. Todos los caracteres que utiliza el Spectrum están guardados en la ROM, excepto los que tienen códigos 144 al 164 (ambos inclusive) cuya definición está almacenada en RAM y puede, por tanto, ser alterada. Cuando encendemos el ordenador, se copian las primeras letras mayúsculas (de la "A" a la "U") en la zona de UDGs; por eso, si tecleamos una "A" en modo gráfico obtendremos una "A" mayúscula. Una vez redefinido el caractér, veremos la imagen que hayamos creado. Ahora, vamos a ver como se definen los UDGs.

Empieza por coger un papel cuadriculado. Traza en él un cuadrado que tenga 8 cuadros de alto y 8 de ancho. En esta reticula de 64 cuadros es donde tenemos que definir nuestro caractér. Cada uno de los 64 cuadros puede estar en blanco o en negro; un cuadro en blanco lo representamos por "0" y uno en negro por "1". En principio todos los cuadros están en blanco, así que podemos representar la reticula de la siguiente forma:

```
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```


Es decir, con 64 ceros. A cada línea horizontal de ceros la vamos a llamar "scan" y a cada cero (o uno) lo vamos a llamar "pixel". Cada pixel queda, por tanto, representado por un bit (digito binario que puede valer "0" ó "1") y cada scan queda representado por un byte (conjunto de 8 bits). A estas alturas, seguro que todos los lectores sabeis que en cada posición de memoria del Spectrum cabe un byte (8 bits), así que cada caractér nos cabe en 8 posiciones de memoria consecutivas. Luego veremos donde están, de momento, vamos a intentar definir alguna imagen en nuestra reticula.

Mira el siguiente conjunto de ceros y unos, y trasládalo al papel cuadriculado que has usado antes, de forma que cada uno sea un cuadrito relleno de tinta (o lapiz) y cada cero sea un cuadrito en blanco:

```
00000000
00000000
00011000
00111100
00111100
10111101
01111110
00000000
```

Si ahora miras el papel desde cierta distancia, verás algo parecido a un sombrero. Ahora, vamos a ver como metemos el sombrero en el ordenador. Vamos a meterlo en el gráfico de la "A". No sabemos la dirección de memoria donde almacenarlo, pero el ordenador nos lo puede decir. Teclea:

```
PRINT USR "A"
```

El número que obtienes es la dirección de memoria donde está almacenada la definición del gráfico correspondiente a la "A", es decir, del caracter con código 144. En realidad, un caracter ocupa 8 posiciones de memoria, así que estará almacenado en esta y en las siete siguientes. Vamos a sustituir la "A" por nuestro sombrero. Teclea el siguiente programa:

```
10 POKE USR "A",BIN 00000000
20 POKE USR "A"+1,BIN 00000000
30 POKE USR "A"+2,BIN 00011000
40 POKE USR "A"+3,BIN 00111100
50 POKE USR "A"+4,BIN 00111100
60 POKE USR "A"+5,BIN 10111101
70 POKE USR "A"+6,BIN 01111110
80 POKE USR "A"+7,BIN 00000000
90 PRINT CHR$ 144
```

Al ejecutar el programa, verás que te aparece en pantalla un sombrerito. Hemos empleado un monton de líneas para generar un simple gráfico, pero lo hemos conseguido y eso es lo importante. Por supuesto, hay otras formas de introducir los gráficos en el ordenador, pero hemos dicho que ibamos a empezar por lo facil.

Ahora vamos a hacer algo un poco más dificil, vamos a convertir cada grupo de 8 ceros y unos (cada scan) a un solo número expresado en decimal. Para ello, a cada bit le damos un

valor y sumamos el valor de todos los que estén a "1". Los valores de los bits, empezando por la izquierda, son: 128, 64, 32, 16, 8, 4, 2 y 1. Los dos primeros scans son, logicamente, 0. El tercer scan es: 16+8=24. El cuarto y quinto son: 32+16+8+4=60. el sexto es: 128+32+16+8+4+1=189 el séptimo es: 64+32+16+8+4+2=126 y el último es cero. Por tanto, nuestro UDG puede quedar representado por los números: 0, 0, 24, 60, 60, 189, 126 y 0. Al ordenador le dá igual que le digamos BIN 10111101 ó que le digamos 189; para él es lo mismo, así que podríamos repetir el programa anterior cambiando todos los BIN por su valor correspondiente, pero aspiramos a ser buenos programadores, así que haremos algo mejor, vamos a emplear un bucle para no repetir instrucciones:

```
10 FOR i=0 TO 7
20 READ a
30 POKE USR "A"+i,a
40 NEXT i
50 DATA 0,0,24,60,60,189,126,0
60 PRINT CHR$ 144
```

De momento nos hemos ahorrado 3 líneas y el programa queda bastante más presentable. Resulta que las direcciones que ocupa el gráfico correspondiente a la "B" están a continuación de las que ocupa el de la "A", así que podemos definir dos gráficos de una sola vez:

```
10 FOR i=0 TO 15: READ a
20 POKE USR "A"+i,a: NEXT i
30 DATA 0,0,24,60,60,189,126,0
40 DATA 8,28,28,28,62,127,127,73
50 PRINT CHR$ 144;CHR$ 145
```

Hemos definido los gráficos de la "A" y la "B" en un solo bucle (esperamos que el lector ponga su mejor voluntad para ver un pequeño cohete en el gráfico de la "B"). En realidad, la definición de todos los UDGs se encuentra en direcciones consecutivas por lo que es posible definirlos todos con un solo bucle; recuerda que el número de iteraciones del bucle ha de ser igual al número de gráficos multiplicado por ocho.

Es un auténtico "peñazo" tener que ir sumando los valores de los bits que componen cada caracter para hallar los 8 números que lo representan. Vamos a buscar una forma más facil de hacerlo. Agrupemos los bits de cuatro en cuatro. A cada grupo de cuatro bits le corresponderá un número del "0" al "9" ó una letra de la "A" a la "F". De esta forma, cada scan quedará representado por dos caractéres. La correspondencia entre los cuatro bits y el carctér correspondiente viene dada por la siguiente tabla:

```
0000 = 0   1000 = 8
0001 = 1   1001 = 9
0010 = 2   1010 = A
0011 = 3   1011 = B
0100 = 4   1100 = C
0101 = 5   1101 = D
0110 = 6   1110 = E
0111 = 7   1111 = F
```


Podemos representar nuestro primer caractér como:

```
00000000 = 00
00000000 = 00
00011000 = 18
00111100 = 3C
00111100 = 3C
10111101 = BD
01111110 = 7E
00000000 = 00
```

Y el segundo como:

```
00001000 = 08
00011100 = 1C
00011100 = 1C
00011100 = 1C
00111110 = 3E
01111111 = 7F
01111111 = 7F
01001001 = 49
```

Si ponemos los datos todos seguidos, el primer caracter sería:

```
0000183C3CBD7E00
```

Y el segundo:

```
081C1C1C3E7F7F49
```

Lo cual se parece bastante al formato empleado en nuestro Cargador Universal de Código Máquina. Seguro que los más adelantados ya se han dado cuenta de que lo que estamos haciendo es representar en hexadecimal los bytes que definen cada caracter. La rutina para cargarlos se complica un poco, pero puedes copiarla en un papel y utilizarla en tus programas cada vez que la necesites:

```
 10 DEF FN a(a$,n)=16*(CODE a$(
n)-48-7*(a$(n)>"9"))+(CODE a$(n+
1)-48-7*(a$(n+1)>"9"))
 20 LET d=USR "A"
 30 FOR i=1 TO 2: READ a$
 40 FOR n=1 TO LEN a$-1 STEP 2
 50 LET a=FN a(a$,n): POKE d,a
 60 LET d=d+1: NEXT n: NEXT i
 70 DATA "0000183C3CBD7E00"
 80 DATA "081C1C1C3E7F7F49"
```

Creemos que este método es el mejor porque simplifica la conversión del UDG diseñado sobre el papel a números para introducir en el ordenador. Sin embargo, te hemos explicado los tres métodos posibles para que emplees el que más te guste.

Por último, nos queda decir que en los modelos de 128K (Plus 2, Plus 3, etc.) hay dos UDGs de menos, ya que los códigos 163 y 164 corresponden a los comandos SPECTRUM y PLAY por lo que hay 19 UDGs en lugar de 21.

Todos los gráficos que necesites

Ya nos parece estar oyendo decir a alguien que "vaya m... de juego se puede hacer con solo 19 gráficos". El buen programador no se conforma con cualquier cosa, y menos el que se dedica a programar juegos, así que vamos a ver la forma de tener todos los gráficos que nos dé la real gana.

Antes hemos hallado la direción donde el ordenador almacena los UDGs tecleando: PRINT USR "A". La respuesta es distinta en un modelo de 48K y en uno de 128K. En el primero será: 65368 y en el segundo será: 65384. El ordenador utiliza la últimas posiciones (por encima de RAMTOP) para almacenar la definición de los UDGs. Pero ¿cómo sabe el ordenador donde están almacenados los UDGs?. Muy sencillo, porque lo tiene apuntado en una de las famosas variables del Sistema, concretamente en la que se llama UDG y está situada en las direcciones: 23675 y 23676; así que si tecleamos:

PRINT PEEK 23675+256*PEEK 23676

obtendremos el mismo resultado que con: PRINT USR "A". Nos viene muy bien que la dirección de los UDGs esté en una variable del Sistema, porque así podemos cambiar su contenido y colocar los UDGs en otra parte (siempre han de estar por encima de RAMTOP, así que tendremos que bajar RAMTOP con CLEAR). Pero puestos a tenerlos en cualquier parte, ¿por qué no generar varios bancos de UDGs y seleccionar uno u otro según lo que queramos imprimir?; al fin y al cabo, el ordenador no tarda nada en cambiar de banco y así podemos tener todos los UDGs que queramos.

En un 48K cada banco de UDGs ocupa 168 bytes (8*21) mientras que en un 128K ocupa 152 bytes (8*19). Vamos a hacer los cálculos de las direcciones para tener 5 bancos de UDGs en un 128K, lo que hace un total de 5*19=95 UDGs. Quien disponga de un modelo de 48K, no tiene más que adaptar los cálculos para que el espacio entre direcciones sea de 168 en lugar de 152.

Cinco bancos por 152 es igual a 760 bytes; 65536-760=64776 que será la dirección del primer banco. La RAMTOP deberá estar, por tanto, en 64775 así que empezamos por hacer:

CLEAR 64775

Las direcciones de cada banco serán:

1º = 64776
2º = 64928
3º = 65080
4º = 65232
5º = 65384

Ahora tenemos que escribir una rutina que nos POKEe en la variable UDG la dirección de cada banco que seleccionemos. Puede ser una rutina a la que se entre con una variable llamada "banco" conteniendo un número de "1" a "5" y retorne tras POKEar en UDG la dirección de ese banco; es decir, tras seleccionarlo. La rutina puede ser:


```
1000 REM SELE-BANCO
1010 LET db=152*banco+64624
1020 POKE 23676,INT (db/256)
1030 POKE 23675,db-256*PEEK 23676
1040 RETURN
```

El procedimiento para definir los UDGs será:

```
xxxx FOR b=1 TO 5
xxxx LET banco=b
xxxx [Define un banco de UDGs]
xxxx NEXT b
xxxx DATA [primer banco]
xxxx DATA [segundo banco]
xxxx DATA [tercer banco]
xxxx DATA [cuarto banco]
xxxx DATA [quinto banco]
```

Antes de imprimir cada UDG hay que asegurarse de que está seleccionado el banco correspondiente. Para ello, se puede hacer una llamada a la rutina "SELE-BANCO" con la variable "banco" conteniendo el número de banco a seleccionar.

En la segunda parte de este artículo, te vamos a contar como tener 96 gráficos en un solo banco, ó varios bancos de 96 gráficos cada uno. Además, vamos a ver la forma de hacer una animación ¡¡totalmente en Basic!!. De momento, y para que le des un poco a la tecla, hemos preparado un pequeño programa que te va a permitir analizar la forma en que está definido cada caracter del Spectrum. Lo hemos hecho cortito para que no te canses de teclear, pero te aseguramos que los resultados son sorprendentes. Se llama "ANALIZA-CHAR" y tienes el listado en la Figura-1. Su manejo es tán sencillo que no hace falta explicarlo, teclea el programa y ya nos contaras...

-.-.-.-.-

```
  10 REM *** ANALIZA-CHAR ***
  20 INPUT "CODIGO?:";a
  30 IF a=0 THEN GO TO 9999
  40 IF a<32 OR a>164 THEN GO TO
 20
  50 CLS : PRINT AT 0,0; INK 7;
PAPER 7;CHR$ a;
  60 FOR i=56 TO 120 STEP 8: PLO
T 64,i: DRAW 64,0: NEXT i
  70 FOR i=64 TO 128 STEP 8: PLO
T i,56: DRAW 0,64: NEXT i
  80 FOR y=0 TO 7: LET b=0: FOR
x=0 TO 7
  90 PRINT AT y+7,x+17;"0";: IF
POINT (x,175-y)=1 THEN PRINT AT
y+7,x+8;CHR$ 143;AT y+7,x+17;"1"
;: LET b=b+2↑(7-x)
 100 NEXT x: PRINT AT y+7,25;"="
;b;: NEXT y
 110 PRINT AT 16,10-1*(a>99);a;"
= ";CHR$ a: GO TO 20
```

GRAFICOS Y ANIMACION EN BASIC (II)

Cumpliendo la promesa que hicimos en la primera parte de este articulo, vamos a ver la forma de definir todos los gráficos que queramos. Tambien veremos una animación realizada totalmente en Basic. Al igual que la vez anterior, prometemos no emplear ni una sola instrucción en código máquina.

Jesús Alonso Rodriguez

Quien piense que para hacer buenos programas es imprescindible saber programar en código máquina, está en un error. Para hacer buenos programas solo hace falta ser un buen programador. Es conveniente aprender código máquina porque es más divertido que el Basic, y se pueden hacer más cosas, pero no es imprescindible. En este articulo utilizaremos algunos trucos de programación avanzada en Basic y conseguiremos efectos bastante curiosos. Suponemos que quien lo lea conoce, al menos, todas las sentencias de Basic. Si alguna sentencia en particular te crea problemas, consúltala en el capítulo correspondiente del manual.

En la primera parte vimos como definir UDGs (Gráficos Definidos por el Usuario) y como tener varios bancos de 19 (ó 21) UDGs cada uno. Dijimos tambien que la definición del juego de caracteres del Spectrum estaba guardada en ROM (excepto los UDGs). Sería estupendo poder re-definir todos los caracteres del ordenador, y no solo los UDGs. De esta forma, podríamos definir un banco de 96 gráficos que se imprimieran en lugar de los caracteres normales y seleccionar este banco cuando fuéramos a imprimir gráficos, pudiendo volver al font de caracteres original cuando haya que imprimir texto... lástima que los caracteres estén definidos en ROM.

Sin embargo, te vamos a dar una buena noticia: ¡¡SI es posible re-definir todos los caracteres del ordenador!! y además, se puede hacer en Basic y de una forma bastante facil. Ahora vamos a ver cómo:

Re-definir todos los caracteres

Como vimos en la primera parte, la dirección de memoria donde están definidos los UDGs se encuentra almacenada en una variable del Sistema que se llama "UDG" y cuya dirección es 23675. Esto nos permite colocar los UDGs en otra parte de la memoria (alterando el contenido de esta variable) ó definir varios bancos de UDGs seleccionando en cada caso el que necesitemos.

Pues otro tanto ocurre con la dirección donde están definidos el resto de los caracteres (excepto los códigos 128 al 143 que no tienen definición, sino que se generan por programa). Existe una variable del Sistema denominada "CHARS" cuya dirección es 23606 y que contiene la dirección donde están definidos los caracteres con códigos 32 al 127 (ambos inclusive). En principio, esta variable apunta a una dirección de ROM, pero podemos alterar su contenido para que apunte a una zona de RAM donde nosotros hayamos definido nuestro propio font de caracteres (que pueden ser letras o gráfcos).

Existe una particularidad con la variable CHARS que la hace ser distinta de UDG. En realidad, esta variable no contiene la dirección del font de caracteres, sino el resultado de restar 256 a esta dirección. ¿Por qué?. Muy sencillo, si multiplicamos el código de un caractér por 8 y le sumamos el contenido de CHARS, el resultado será la dirección donde está definido ese caractér. Pero el código del primer caractér no es cero, sino 32 y precisamente, 32 por 8 es igual a 256; por tanto, el contenido de CHARS ha de ser 256 números más bajo que la dirección del primer caractér. Vamos a comprobarlo. Empecemos por leer el contenido de CHARS:

```
PRINT PEEK 23606+256*PEEK 23607
```

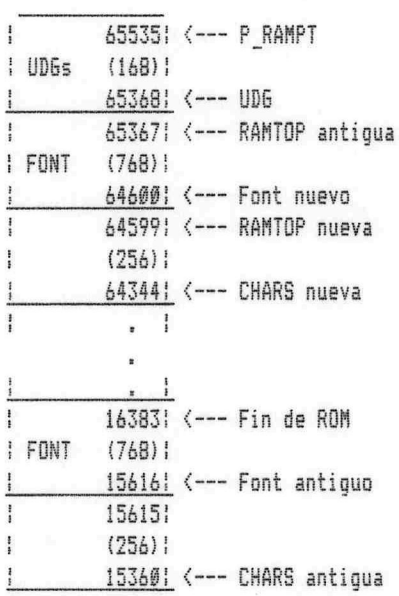
El resultado ha de ser: 15360. Si le sumamos 256, obtenemos: 15616 (3D00h) y esta es, precisamente, la dirección a partir de la que está definido el font de caracteres. Hay 96 caracteres y cada uno ocupa 8 bytes, por lo que el font completo ocupa 96*8=768 bytes. Si sumamos 768 a 15616 obtenemos 16384 que es la primera dirección de RAM. Por tanto, la definición de los caracteres ocupa las 768 últimas direcciones de la ROM (en los modelos de 128K, cuando decimos "la ROM" hay que entender que nos referimos a la ROM de ejecución, es decir, la que contiene el intérprete de Basic y está paginada durante la ejecución de un programa).

Si queremos construir otro font de caracteres (o un banco de gráficos) no tenemos más que ubicarlo en una dirección de RAM, definirlo de la misma forma que los UDGs (sin olvidar que esta vez son 96 caracteres) y direccionarlo alterando el contenido de CHARS para que sea el resultado de restar 256 a la dirección donde hemos almacenado nuestro font. Tambien podemos definir solo parte de los caracteres copiando todos los caracteres desde ROM y redefiniendo solo aquellos que nos interesen. Pero como ejemplo, vamos a hacer algo aún más divertido: vamos a copiar en RAM los caracteres de la ROM, pero haciéndoles una pequeña alteración para que el ordenador imprima con letra cursiva (tambien llamada: "itálica").

Para conseguir letra cursiva, hay que hacer que cada letra se incline a la derecha. Para ello, desplazamos a la derecha los tres "scans" superiores de su definición y a la izquierda los tres inferiores, dejando los dos centrales como están. Vimos en la primera parte que cada scan quedaba definido por un número en decimal; pues bien, si dividimos ese número por dos (tomando solo la parte entera) el scan queda desplazado a la derecha; si lo multiplicamos por 2 (en caso de que sea mayor de 255 hay que restarle 256) el scan queda desplazado a la izquierda. Para entender por qué ocurre esto es necesario tener unos ciertos conocimientos de matemáticas, así que mejor os lo creeis y nos ahorramos explicarlo (de verdad, la explicación es un rollo y nos ocuparía varias páginas).

Para este experimento, los que tengan un 128K deberán pasar a modo 48K, de esta forma, las explicaciones que demos serán utilizables por todos los lectores.

Vamos a colocar nuestro nuevo font inmediatamente debajo de los UDGs, así que tendremos que bajar RAMTOP. El siguiente cuadro nos ayudará a localizar todas las direcciones. Es interesante hacer un cuadro como este cada vez que se quiera distribuir la memoria:



Tenemos que bajar la RAMTOP a 64599 y almacenar en CHARS la dirección 64344. Asimismo, tendremos que mover 768 bytes desde la dirección 15616 hasta 64600; pero los vamos a mover en grupos de 8 y por el camino, desplazaremos a la derecha (dividir por 2) los tres primeros de cada grupo y a la izquierda (multiplicar por 2) los tres últimos. Veamos el programa:

```
 10 CLEAR 64599
 20 FOR i=15616 TO 16376 STEP 8
 30 POKE i+48984,PEEK i/2
 40 POKE i+48985,PEEK (i+1)/2
 50 POKE i+48986,PEEK (i+2)/2
 60 POKE i+48987,PEEK (i+3)
 70 POKE i+48988,PEEK (i+4)
 80 FOR j=5 TO 7
 90 LET a=PEEK (i+j)*2
100 LET a=a-256*(a>255)
110 POKE i+48984+j,a
120 NEXT j: NEXT i
130 POKE 23607,INT (64344/256)
140 POKE 23606,64344-256*PEEK 2
3607
150 LIST
```

Teclea y ejecuta el programa. Si no has cometido ningún error, al cabo de unos segundos verás el listado en letra itálica. A partir de este momento, todo lo que escribas en el ordenador saldrá en itálica. Para volver a la letra normal teclea:

```
POKE 23607,60: POKE 23606,0
```

Tecléalo como una sola línea, si intentas hacerlo en dos veces (un POKE de cada vez) te volverás loco (adivina por qué). Para volver a activar la letra itálica basta con que hagas: GOTO 130. Comprobarás que ambos tipos de letra pueden convivir en pantalla sin ningún problema.

Creemos que la única instrucción que vale la pena explicar de este programa es la contenida en la línea 100.

Es equivalente a:

```
100 IF a>255 THEN LET a=a-256
```

con la única diferencia de que se ejecuta con mayor rapidez. El valor del paréntesis será "1" si "a" es mayor de 255 y "0" si es menor ó igual, con lo cual, solo se restará 256 si "a" es mayor de 255. La posibilidad de emplear operadores lógicos de esta forma es una particularidad del Basic del Spectrum (no lo intentes hacer en otros ordenadores, que no funciona) que nos permite ahorrar mucho tiempo de ejecución, ya que el evaluador de expresiones es bastante más rápido que el intérprete. Cuando escribamos el programa de animación, haremos un grán uso de esta técnica ya que al animar figuras, la velocidad es muy importante.

Igual que hicimos en la primera parte con los UDGs, puedes definir un banco de gráficos para alternarlos con los caracteres normales y tener 96 diferentes gráficos en un solo banco. Tambien puedes definir varios bancos de gráficos para tener más de 96 gráficos distintos; para ello, solo tienes que modificar el programa "SELE-BANCO" de la primera parte, para que trabaje con bancos de 768 bytes; no olvides que lo que hay que almacenar en CHARS es un número 256 unidades menor que la dirección del banco. Por cierto, en la primera parte, nos olvidamos de poner un GOSUB 1000 despues de la linea: "xxxx LET banco=b" del procedimiento para definir los UDGs; el error es tan trivial que estamos seguros de que ya os habíais dado cuenta.

Probablemente, os pregunteis que para qué se quieren tantos gráficos; bien, ahora lo vais a ver.

Gráficos más grandes

En los programas comerciales, habrás visto que los gráficos suelen ser más grandes que 8 por 8 pixels. Tambien es posible conseguir este efecto en Basic. Para ello no hay más que agrupar los caracteres de varios en varios. Por ejemplo, supongamos que queremos conseguir caracteres de 24 pixels de ancho por 16 de alto; en ese caso, nos bastará con agrupar 6 caracteres. Cada vez que vayamos a imprimir el gráfico en las coordenadas "l,c" lo haremos de la siguiente forma:

```
PRINT AT l,c;"ABC"; AT l+1,c;"DEF"
```

Hemos supuesto que el gráfico está formado por las letras "ABC" y "DEF". En este caso, "c" no puede pasar de 29 ni "l" de 20 ya que el gráfico se saldría de la pantalla.

Muchos lectores se preguntan cómo es posible que los programas comerciales tengan tantas pantallas; si cada una ocupa 6 Ks, no cabrían en la memoria. Bien, todo tiene truco; en este caso el truco está en definir cada pantalla como una combinación de gráficos grandes; por ejemplo, de 4 por 4 caracteres (32 por 32 pixels). En cada pantalla caben 40 de estos gráficos (5 por 8 en 32 columnas y 20 líneas) por lo que cada pantalla puede quedar definida por 40 bytes; ¡imaginate si te caben pantallas!. Vamos a ver una forma de hacerlo:

Cada gráfico nos ocupa 16 caracteres, por tanto, en cada banco nos caben 6 gráficos (16*6=96). Podemos utilizar un código interno para cada gráfico; supongamos que nuestro programa utiliza 18 gráficos, por lo que necesitaremos 3 bancos; el

código interno puede ir desde "1" hasta "18". Cada pantalla estará almacenada como 40 números del 1 al 18. Para hacer corresponder cada código interno con sus caracteres, utilizamos una matriz de caracteres que tenga 18 elementos de 17 caracteres cada uno: DIM A$(18,17). El código interno será el subíndice que nos seleccionará un determinado elemento de la matriz. Dentro de cada elemento, el primer caractér será el número de banco y los 16 restantes serán las letras de ese banco que definen nuestro gráfico. Ya sabemos que las tenemos que imprimir de 4 en 4. Una matriz de ejemplo podría ser:

```
A$(1)="1 !"#$%&'()*+,-./"
A$(2)="10123456789:;<=>?"
A$(3)="1@ABCDEFGHIJKLMNO"
A$(4)="1PQRSTUVWXYZ[\]^_"
...
A$(7)="2 !"#$%&'()*+,-./"
A$(8)="20123456789:;<=>?"
...
A$(13)="3 !"#$%&'()*+,-./"
A$(14)="30123456789:;<=>?"
...
```

Aunque lo lógico es que se complique más ya que un mismo caractér puede formar parte de la definición de varios gráficos; siempre que sea dentro del mismo banco. Una rutina de ejemplo para imprimir el gráfico con código interno "a" en las coordenadas "l,c" podría ser la siguiente (suponemos que la rutina "SELE-BANCO" que nos selecciona un determinado banco está en la línea 1000):

```
100 LET banco=VAL A$(a,1)
110 GOSUB 1000
120 PRINT AT l,c;A$(a,2 TO 5);
AT l+1,c;A$(a,6 TO 9); AT l+2,c;
A$(a,10 TO 13); AT l+3,c;A$(a,14
TO );
130 RETURN
```

En este caso, "c" no puede pasar de 28 ni "l" de 18. Por este procedimiento, puedes tener todos los gráficos que quieras, del tamaño que quieras y con el número de pantallas que quieras; creemos que vale la pena el pequeño esfuerzo adicional de programación que ello exije.

Cada pantalla se puede almacenar en 40 bytes de memoria por encima de RAMTOP. Las pantallas pueden estar numeradas de "1" en adelante. La dirección base de cada pantalla se puede hallar restando "1" a su código, multiplicando por "40" y sumando el resultado a la dirección base de la primera de ellas. Sin embargo, este procedimiento no es Basic estrictamente. Otra posibilidad es que cada pantalla sea un elemento de una matriz que tenga tantos elementos como pantallas haya, cada uno de los cuales tenga 40 caracteres de longitud. Es interesante utilizar una matriz de caracteres ya que ocupa mucha menos memoria que las de números.

Cada caractér de cada elemento de la matriz contiene el código interno de un gráfico. Este código puede ir de "1" a "18"; por tanto, utilizaremos los números del "1" al "9" y las

letras de la "A" a la "I". Supongamos que la matriz de pantallas se llama P$ la rutina para imprimir la pantalla cuyo número es "p" podría ser:

```
200 FOR c=0 TO 28 STEP 4
210 FOR l=0 TO 16 STEP 4
220 LET a=CODE P$(p,l*2+c/4+1)
230 LET a=a-48-7*(a>57)
240 IF a<>0 THEN GOSUB 100
250 NEXT l: NEXT c: RETURN
```

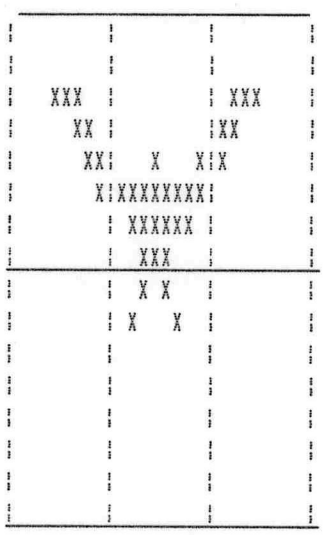
Fíjate que utilizamos la rutina vista anteriormente (GOSUB 100) para imprimir cada gráfico, y que entramos a ella 40 veces con los valores de "l", "c" y "a" correspondientes a cada uno de los gráficos que forman una pantalla. En la línea 240 hemos previsto que se pueda utilizar el "0" para no imprimir ningún caractér.

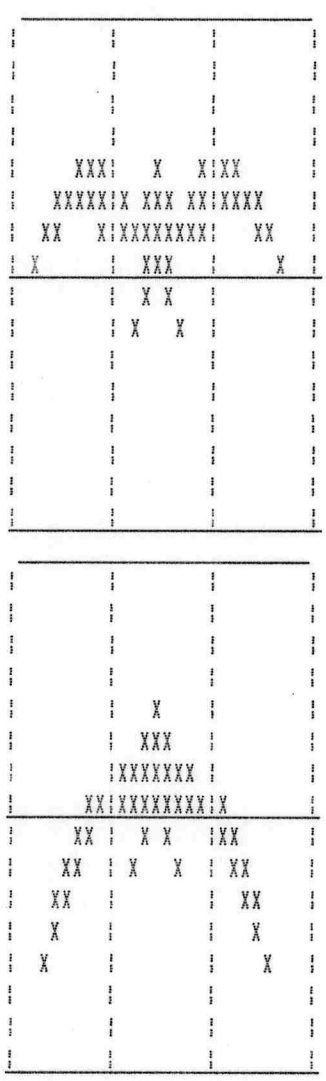
A continuación y como prometimos, vamos a ver un ejemplo de animación realizado totalmente en Basic. Quienes afirman que el Basic no vale para programar juegos, que se preparen:

Animación en Basic

Para no complicar demasiado el ejemplo y que sea compatible con cualquier modelo de Spectrum, hemos empleado los UDGs correspondientes a las letras de la "A" a la "R". Definimos una figura de un pájaro volando, y diseñamos tres fases del movimiento de las alas. La primera es con las alas arriba, la segunda es con ellas en la posición intermedia y la tercera con ellas abajo. Para realizar una animación medianamente creible son necesarias, al menos, cuatro fases del movimiento. En este caso, sin embargo, se trata de un movimiento cíclico en el que la fase intermedia se repite en una cadencia: 1,2,3,2,1,2,3... con lo que tres fases son suficientes.

Cada una de las fases está definida en 6 UDGs (cada gráfico tiene 3 UDGs de ancho y 2 de alto). Se han definido algunos UDGs repetidos con el fin de que este mismo elemplo se pueda aplicar a cualquier gráfico. La definición de los UDGs es la siguiente (hemos representado un pixel activado con una "X" y uno desactivado con un espacio en blanco):





En el LISTADO 1 se puede var el programa que genera el gráfico y lo anima. Como siempre, las letras que aparecen subrayadas deberán teclearse en modo gráfico (por ejemplo, si en el listado aparece: ABC deben pulsarse las siguientes teclas:

[Caps Shift] + [9], [A], [B], [C], [Caps Shift] + [9]

En los modelos con teclado en castellano, la tecla "Caps Shift" está etiquetada cómo "MAYUS." ó "MAYUSCULAS". Donde hemos puesto un signo "mas" (+) entre las teclas, significa que ambas deben pulsarse a la vez. Al teclear el programa, en la pantalla no debe salir el gráfico ni la letra subrayada; en su lugar, saldrá la letra mayúscula correspondiente; sin embargo, una vez ejecutado el programa, las letras habrán sido sustituidas por los gráficos correspondientes.

Vamos a estudiar el programa línea a línea: A partir de la línea 1000 está la definición de los gráficos en hexadecimal, es decir, de la tercera de las formas explicadas en la primera parte de este artículo. Las líneas 100 a 160 leen los DATAs que definen los gráficos y los cargan en los UDGs correspondientes (este proceso dura unos 5 segundos). En la línea 180 definimos la matriz G$ que contiene las fases del movimiento en el orden: 1,2,3,2. Cuando variemos el subindice de esta matriz en el orden

1,2,3,4,1,2,3,4... las fases del movimiento variarán en el orden: 1,2,3,2,1,2,3,2... La línea 210 inicializa las variables correspondientes a la posición actual del pájaro ("li" y "co"), a la posición anterior ("lia" y "coa") y a la fase en curso ("si"). Las líneas 220, 230 y 240 constituyen el bucle de animación; hemos puesto especial cuidado en que todas las operaciones que se realicen en estas líneas tomen el menor tiempo posible de ejecución.

En la línea 220 borramos el gráfico de la posición anterior ("lia,coa") y lo imprimimos en la nueva ("li,co") inmediatamente para reducir el parpadeo al mínimo; a continuación, asignamos las nuevas coordenadas a las antiguas. En la línea 230 se genera un número aleatorio comprendido entre "0" y "4" (ambos inclusive) y se ajustan las nuevas coordenadas conforme a ese número: si es "0", el pájaro se moverá a la izquierda; si es "1", a la derecha; si es 2 se queda donde está; si es "3" se moverá hacia arriba y si es "4" hacia abajo. Para gestionar este movimiento hacemos uso de operadores lógicos que son mucho más rápidos que las sentencias del tipo IF ... THEN. Al incrementar o decrementar las coordenadas, se tiene en cuenta que el pájaro no puede salirse de la pantalla; para ello, "li" no puede ser menor de "0" ni mayor de "20" y "co" no puede ser menor de "0" ni mayor de "29". En la línea 240 se incrementa el subíndice de la matriz G$ que indica la fase del movimiento y si ya fuera "4" se le hace ser "1".

El LISTADO 2 es una variante de este mismo programa en donde el pájaro no se mueve de forma aleatoria, sino en función de las teclas de cursor que se pulsen. Para ello solo ha sido necesario modificar la línea 230; el resto de las líneas son iguales que en el caso anterior.

Esperamos que estos dos programas y las restantes ideas propuestas sivan de ejemplo para que las incorpores en tus programas. A partir de ahora, nadie va a poder quejarse por no saber cómo hacer gráficos en el Spectrum.

----- O -----