CAPÍTULO OCTAVO

Programación de juegos

UN JUEGO DE LABERINTOS Y AVENTURAS

El juego que veremos ahora se llama LABERINTO y es en parte un juego de laberinto, y en parte un juego de aventuras en miniatura. La idea es empezar en la esquina inferior derecha del laberinto y trazarse un camino para llegar a la salida. Sin embargo, el mapa del laberinto no es siempre visible, y sólo puede verlo 5 veces en cada partida. A lo largo del camino se encontrará con tesoros y sufrirá peligros. Si usted no consigue el verdadero tesoro, o no sabe lo que hacer con él en el tiempo correcto, entonces pierde. Le sugiero que cuando escriba el programa repase varias veces el listado para corregir los posibles errores de transcripción. Ejecute el programa, y si éste detiene con algún error, mire la línea en que se ha producido y compárela con la que aquí tiene:

```
  2 REM **********************
  3 REM      EN EL LABERINTO
  4 REM **********************
 10 REM para ver el mapa entre
'MAP'
 25  LET F=0
 30 LET T=0
 35 GO SUB 1000
 40 LET A=0: RESTORE 110
 45 LET TRES=0
 50 LET X=31
 55 PAPER 2: INK 6: BORDER 6: CLS
 60 DIM Z$(36,6)
```

```
  65 DIM I$(6)
  70 DIM D$(1)
  75 DIM S$(1)
  80 FOR C=1 TO 36
  90 READ Z$(C)
 100 NEXT C
 110 DATA "000110","001010","000100","10
1010","000110","001010"
 120 DATA "100011","000111","001011","00
0111","001011","000001"
 130 DATA "000101","001011","000011","00
0011","010111","001000"
 140 DATA "000110","001101","011101","00
1011","000101","001010"
 150 DATA "000101","001010","000110","01
1111","001010","000011"
 160 DATA "000100","001001","000001","00
0101","001101","101001"
 170 INPUT "QUE CAMINO?(N,S,E,O)"; LINE
D$
 180 PRINT AT 0,0;"
          "
 195 IF TRES=1 AND D$="p" THEN  LET I$(1
+INT (X*6))="1"
 200 IF D$="m" THEN  GO TO 300
 210 IF d$="n" THEN  LET A=6
 220 IF D$="s" THEN  LET A=5
 230 IF D$="e" THEN  LET A=4
 240 IF D$="o" THEN  LET A=3
 250 IF X=18 AND A=4 THEN  GO TO 290
 260 GO SUB 500
 280 GO TO 170
 290 CLS : PRINT  INK 2; PAPER 6; FLASH
1;AT 10,10;"ESTAS FUERA": STOP
 300 REM MAPA: CLS
 310 PRINT AT 5,5;"*** *** ***"
 320 PRINT AT 6,5;"* * * * * *"
 330 PRINT AT 7,5;"* *** *** *"
 340 PRINT AT 8,5;"* * * * *"
 350 PRINT AT 9,5;"*** * * ***>>"
 360 PRINT AT 10,5;"  * * * *"
 370 PRINT AT 11,5;"******* ***"
 380 PRINT AT 12,5;"*     *   *"
 385 PRINT AT 13,5;"*** ***** *"
```

```
 390 PRINT AT 14,5;"  * * * * * "
 410 PRINT AT 15,3;">>*** * *****"
 415 PAUSE 0
 420 LET G=INT (X/6): LET H=X-1-(INT (X/
6))*6
 430 IF G<>(X/6) THEN  GO TO 450
 440 LET G=G-1: LET H=5
 445 PRINT AT 1,0;"X=";X;",";"G=";G;",";
"H=";H
 450 PRINT AT 5+2*G,5+2*H; OVER 1; FLASH
 1;" "
 460 PAUSE 100: LET T=T+1
 470 IF T=5 THEN  GO TO 490
 480 CLS : GO TO 170
 490 CLS : PRINT AT 10,10; FLASH 1; INK
2; PAPER 6;"PERDISTE": STOP
 500 REM ESTE CAMINO?
 510 IF Z$(X,A)="0" THEN  GO TO 540
 520 IF Z$(X,A)="1" THEN  LET X=X+(1 AND
 A=4)-(1 AND A=3)+(6 AND A=5)-(6 AND A=6
)
 525 IF Z$(X,1)="1" THEN  GO TO 600
 526 IF Z$(X,2)="1" THEN  GO TO 700
 530 RETURN
 540 PRINT AT 0,0;"NO PUEDES IR POR AQUI
!": RETURN
 550 STOP
 600 REM TESORO
 610 PRINT AT 0,0;"PUEDES VER UN"+(V$ AN
D (INT (X/6))=1)+(L$ AND (INT (X/6))=0)+
(E$ AND (INT (X/6))=6)
 620 INPUT "QUE QUIERES HACER? "; LINE S
$
 630 IF S$="p" THEN  GO TO 640
 636 PRINT AT 0,0;"NO LO TIENES
          ": RETURN
 640 PRINT AT 0,0;"AHORA SI LO TIENES
          "
 645 LET TRES=1
 650 RETURN
 699 STOP
 700 REM PROBLEMA
 710 PRINT AT 0,0;"HAY UN"+(Q$ AND (INT
(X/6))=2)+(U$ AND (INT (X/6))=3)+(M$ AND
```

```
 (INT (X/6))=4)
 720 INPUT "QUE HACES? "; LINE S$
 730 IF S$=(Y$(1) AND INT (X/6)=4)+(Y$(2
) AND INT (X/6)=3)+(Y$(3) AND INT (X/6)=
2) THEN  GO TO 800
 740 PRINT AT 0,0;"LO SIENTO....
          ": GO TO 900+(RND*3)+1
 800 IF TRES>0 THEN  GO TO 820
 810 PRINT "LO SIENTO, NO QUISO HACERLO.
...": STOP
 820 PRINT AT 0,0;"LE GUSTARIA QUE CONTI
NUARAS     ";AT 1,0;"
            "
 830 GO TO 170
 901 PRINT "TE HA GOLPEADO
   ": STOP
 902 PRINT "TE LO HA HECHO PAGAR MUY CAR
O  ": STOP
 903 PRINT "HAS CAIDO EN SU TRAMPA - ADI
OS ": STOP
1000 LET V$="DIAMANTE GIGANTE"
1010 LET L$="ELIXIR DE LA VIDA"
1020 LET E$="FRUTO DEL AMOR"
1030 LET Q$="DRAGON"
1040 LET M$="NIBELUNGO"
1050 LET U$="UNICORNIO"
1060 DIM Y$(3)
1070 LET Y$(1)="r": LET y$(2)="o": LET y
$(3)="t"
1090 RETURN
```

Aquí tiene un esquema que le indica lo que hace cada una de las partes de este programa:

LÍNEAS	COMENTARIO
25- 75	Definen las variables y dimensionan los conjuntos.
80- 160	Los datos para el laberinto se asignan a la cadena Z$.
170- 280	Se detecta la tecla pulsada.

300- 480	Dibuja el mapa en pantalla durante unos momentos.
500- 540	Detecta si la dirección respondida es posible.
600- 650	¿Qué tesoro ha encontrado?
700- 830	Un problema y su respuesta.
901- 903	Distintos finales.
1000-1090	Asignación de variables.

La clave para entender cómo funciona este programa la tenemos en las líneas de DATA de la 110 a la 160. Aquí está registrado el funcionamiento lógico del laberinto en forma de números binarios de seis bits, que le permiten saber al programa qué direcciones se pueden escoger en un punto determinado y también si hay un problema o un tesoro en esa posición. Esto se consigue dando un significado a cada uno de los seis bits:

bit 5	bit 4	biy 3	bit 2	bit 1	bit 0
tesoro	problema	oeste	este	sur	norte

Esto puede ser un poco complicado de entender a menos que se imagine que un 1 en una columna significa "sí"; así, por ejemplo, un 1 en el bit 5 significa que en esta posición hay un tesoro, un 1 en el bit 0 significa que usted puede ir en dirección norte desde aquí, etc. Por ejemplo:

> 010101 significa que se encuentra en una posición del laberinto en la que hay un problema y que desde aquí sólo puede ir al este o al norte.

El trabajo del programa consiste en ver si el bit 5 o el 4 contienen un 1, y si esto sucede ir a las rutinas de tesoro o problema respectivamente (vea las líneas 525 y 526 del programa). La otra tarea importante del programa consiste en interpretar las direcciones tecleadas en términos de los bits del 0 al 3. La manera más fácil de hacer esto es guardar los bits en una cadena y utilizar las grandes facilidades que ofrece el Spectrum para su tratamiento. Así, si

la dirección obtenida es "n" (norte), el programa lo único que hace es cargar en una variable (A) el valor 6 (ya que el bit 0, el que indica si se puede ir al norte, está en la sexta posición de la cadena) y comprobar si Z$ (X, A) es 1. Esto ocurre en las líneas 210-240 y 510-520.

Obviamente, si Z$ (X, A) es 0 entonces no puede ir hacia esa dirección, y la línea 540 se lo indica. La línea 520 es un poco más compleja ya que utiliza **AND.** Poniendo AND después de un campo y luego una condición, se provoca que dicho campo, sólo sea utilizado en el caso de que la condición sea cierta, y no se usa si la condición es falsa. Así:

```
PRINT ("Hola" AND 2 > 3)
```

Esto no imprimirá nunca "Hola" ya que la condición es falsa. Esto es lo que se usa en la línea 520 para hacer que su posición en el laberinto sea una más o una menos de la que era (es decir, ir al este o al oeste), o bien seis más o seis menos de lo que era (lo mismo que arriba o abajo). Un procedimiento similar se utiliza en la línea 610 para decidir con qué tesoro se ha encontrado, y otra vez en la 710 para decidir con qué problema se tendrá que enfrentar.

Cuando se encuentra con un problema o un tesoro, entonces el programa le pregunta qué desea hacer. Y sólo mira a la primera letra de la respuesta. Recordará que esto se puede hacer fácilmente dimensionando una cadena de un solo carácter de longitud. La ventaja de esto es que se evita el tener que poner tantas rutinas de comprobación de respuestas, para ver si la frase o la palabra introducida era la requerida o no. Por supuesto que cualquier respuesta con la primera letra correcta será tomada como tal (esto no debería decírselo a los jugadores).

Como puede ver, los tesoros y problemas son asignados a variables, al principio del juego, para que estén disponibles cuando se necesiten. En este caso he preferido dibujar el mapa directamente con PRINT AT (líneas 310-410) donde las posibles posiciones del aventurero son 36. La posición actual en el laberinto puede ser calculada a partir de X. Como puede ver en la línea 450 simplemente hay que añadir 5 al doble del valor de X para obtener la posición y hacer un FLASH en dicho punto. La línea 60 hace que el mapa sólo sea visible durante unos 2 segundos (PAUSE

100), y también almacena en la variable T el número de veces que se ha mirado el mapa, que si sobrepasan las cinco producen la pérdida de la partida. Con toda esta información usted está capacitado para averiguar cómo funciona el programa, especialmente si lo juega unas pocas veces.

ALGUNAS OBSERVACIONES

Hubiera sido muy fácil hacer un laberinto tridimensional, simplemente añadiendo dos bits más a las cadenas de DATA para indicar arriba o abajo. Las cadenas entonces tendrían una longitud de 8. El mapa dibujado, dependería de la planta en que estuviera.

El mapa se ha dibujado con PRINT AT por dos razones, una porque es más fácil, y otra porque así el mapa aparece en el listado. Sin embargo, al componer un juego como éste, en que la naturaleza y la forma del laberinto no deben ser vistas fácilmente, es mejor disfrazar un poco toda esta información. Los asteriscos que componen el mapa pueden ser registrados en líneas DATA como la posición en que se deben imprimir. Esto hará que sea un poco más difícil de descifrar. También es posible disfrazar los distintos tesoros y problemas almacenándolos en una variable como un conjunto de códigos y para leerlos, se utilizará el comando CHR$. Métodos similares se pueden usar para disimular el lugar en que se encuentran éstos.

Obviamente en este juego el laberinto es el mismo cada vez. Puede cambiarlo dibujándolo en un papel y entrando luego los correspondientes cambios tanto en las líneas DATA como en la rutina que lo dibuja. Sin embargo, pensando un poquito, sería posible crear un laberinto aleatorio, y de modo que el programa también decida en qué lugar se encontrarán los tesoros y los problemas. Para crear un laberinto lo que tiene que hacer esencialmente es crear 36 (para uno de 6 × 6 como éste) cadenas cada una compuesta por un número binario de 6 cifras. Esto se puede hacer asignando aleatoriamente un 1 o un cero a cada bit o bien crear también aleatoriamente números entre 1 y 63 y pasarlos luego a forma binaria, según una rutina que sugerí en la primera parte del libro. El mayor problema consiste en asegurarse que cada intersección conecta al menos con

otra, y también asegurar que es posible ir desde la entrada a la salida. Si se supera este obstáculo, se puede crear un programa muy interesante.

Por último, observará que cuando se pierde por no haber tratado correctamente un problema el programa termina con una frase aleatoria de las líneas 901-903. Esto se consigue en la línea 740 donde se toma la decisión de ir a un número de línea entre 901 y 903:

```
740 PRINT AT 0,0; "Lo siento...": GOTO
900 + (RND*3) + 1
```

Este método de escoger al azar la frase que se imprimirá causa un gran efecto en juegos en los que no se puede predecir la respuesta, con lo que se hacen menos aburridos. El método se puede usar en otro tipo de aplicaciones como en el siguiente programa FRASES:

```
  2 REM     ***************
  3 REM          frases
  4 REM     ***************
  5 REM CHERI DAVIS-LANGDELL:
       1982
  6 PAPER 1: INK 5: BORDER 1: CLS
 10 REM
 20 REM
 30 REM
 40 DIM a$(5,35)
 50 DIM b$(5,35)
 60 DIM c$(5,35)
 70 DIM d$(5,35)
 80 GO SUB 1000
 90 FOR n=1 TO 5
100 PRINT AT RND*10,0;a$(n)''b$(n)''c$(
n)''d$(n)
105 PAUSE 400
106 CLS
107 BEEP .2,20
110 NEXT n
140 LET n1=1+RND*4: LET n2=1+RND*4: LET
n3=1+RND*4: LET n4=1+RND*4
150 PRINT AT RND*10,0;a$(n1)''b$(n2)''c
$(n3)''d$(n4)
```

```
 160 PAUSE 400
 170 CLS : BEEP .2,20: GO TO 140
 990 STOP
1000 REM asignando lineas
1005 RESTORE 1120
1010 FOR x=1 TO 5
1020 READ a$(x)
1030 NEXT x
1040 FOR x=1 TO 5
1050 READ b$(x)
1060 NEXT x
1070 FOR x=1 TO 5
1080 READ c$(x)
1085 NEXT x
1090 FOR x=1 TO 5
1100 READ d$(x)
1110 NEXT x
1120 DATA "Me gusta apostar","El ansioso
 vive para jugar","Quien mal anda mal ac
aba","Olvida los problemas","Ven y rie c
onmigo"
1130 DATA "La primavera trae alegria","C
on junio, el verano","La fruta esta ya m
adura","Levantate antes que el sol","Tod
avia no se ha puesto el sol"
1140 DATA "Golpe a golpe, paso a paso","
Magico es el mar","El muerto yace bajo l
a luna","Que empiece el baile!","Al pan,
 pan y al vino, vino"
1150 DATA "El vagabundo ya no llorara ma
s","Va a llover, dulce pajarito","Quien
se mueve en la noche?","Pia gaviota, pia
","Poderoso caballero"
1160 RETURN
```

Otra variación en este tema se usó en el juego del ahorcado en la primera parte. Allí utilicé la idea de almacenar las posibles palabras a adivinar en líneas DATA, cada palabra con su propio número de línea. Luego, usando el hecho de que RESTORE n coloca el puntero para READ en la línea DATA que tiene por número n, sólo tuve que escoger n aleatoria y convenientemente.

Para que el programa se vea más elaborado, queda muy

bien el poner los mensajes finales en letras de doble alto. Aquí hay un programa que lo hace:

```
    2 REM **********************
    3 REM ESCRIBE EN DOBLE ALTO
    4 REM **********************
    5 LET l=0: LET r=-1
    6 BORDER 6: PAPER 6: CLS : LET z=0
    7 PRINT "USA SHIFT '0' PARA BORRAR,
     PULSA ENTER PARA NUEVA LINEA"
    8 PAUSE 0: CLS
    9 PAUSE 0: LET r=r+1: IF r=31 THEN  G
O TO 1000
   10 LET t$=INKEY$
   15 IF t$="0" THEN  GO TO 500
   16 IF CODE INKEY$=12 THEN  GO TO 500
   17 IF CODE INKEY$=13 THEN  GO TO 1000
   20 FOR a=1 TO LEN t$
   30 FOR x=0 TO 6 STEP 2
   40 POKE USR "a"+x,PEEK (15616+(8*(CODE
 t$(a)-32))+z)
   50 POKE USR "a"+x+1,PEEK (15616+(8*(CO
DE t$(a)-32))+z)
   55 LET z=z+1
   60 NEXT x
   70 FOR x=1 TO 7 STEP 2
   80 POKE USR "b"+x-1,PEEK (15616+8*(COD
E t$(a)-32)+z)
   90 POKE USR "b"+x,PEEK (15616+8*(CODE
t$(a)-32)+z)
   95 LET z=z+1
  100 NEXT x
  101 LET z=0
  105 REM  'A' y 'B' son graficos en la s
iguiente linea
  110 PRINT AT l,r;"a";AT l+1,r;"b"
  120 NEXT a
  130 GO TO 9
  500 REM  BORRAR
  510 LET r=r-2
  520 GO TO 9
1000 REM siguiente linea
1010 LET l=l+2: LET r=-1: GO TO 9
```

El otro tipo de juego es el juego gráfico con movimiento. Star Gobbler y Colony son dos ejemplos de esta clase de juegos, así como lo son todos los juegos de invasores, marcianitos, etc.

Frecuentemente, estos juegos, requieren que el jugador mueva alguna cosa por la pantalla. Normalmente, esto se hace usando las flechas que están en las teclas 5 a 8 porque así ya se indica la dirección del movimiento. Aquí tenemos la rutina base:

```
10 IF INKEY$ = "6" THEN LET X = X + 1
20 IF INKEY$ = "7" THEN LET X = X - 1
30 IF INKEY$ = "5" THEN LET Y = Y - 1
40 IF INKEY$ = "8" THEN LET Y = Y + 1
50 GO TO 10
```

Aquí X representa la coordenada horizontal del objeto que está en la pantalla e Y representa a la coordenada vertical. Por supuesto que luego hay que imprimir el objeto en la pantalla, ya sea con PRINT AT o con PLOT, ya que esta rutina no contiene la impresión del objeto. Para que la figura que se imprime parezca realmente que se mueva, necesita usar alternativamente PRINT y PRINT OVER (o bien PLOT y PLOT OVER). Aquí tiene un ejemplo:

```
10 LET X = 10: LET Y = 15
20 IF INKEY$ = "7" THEN LET X = X - 1
30 IF INKEY$ = "6" THEN LET X = X + 1
40 IF INKEY$ = "8" THEN LET Y = Y + 1
50 IF INKEY$ = "5" THEN LET Y = Y - 1
60 PRINT AT X,Y;"*":PRINT AT X,Y;OVER 1;"*"
70 GOTO 20
```

Si introduce este programa y lo ejecuta, rápidamente observará dos problemas; primero que el asterisco puede exceder fácilmente los límites de la pantalla produciéndose un error ("Out of screen..."), y segundo que el asterisco se mueve muy deprisa y parpadea mucho. Estos problemas

se solucionan detectando los valores máximos de X e Y y añadiendo PAUSE entre PRINT y PRINT OVER.

```
25 IF X<0 THEN LET X = 0
35 IF X>20 THEN LET X = 20
45 IF Y>31 THEN LET Y = 31
55 IF Y<0 THEN Y = 0
60 PAUSE 2: PRINT AT X,Y;"*":PAUSE 2: PRINT
OVER1; AT X,Y ;"*"
```

Añadiendo estas líneas, se encontrará con que puede mover el asterisco hacia cualquier parte de la pantalla.

Hacer que el carácter sea una nave espacial, aunque contenga más de un cuadrado, y moverlo por la pantalla es igual de fácil. Pero recuerde que si la nave se compone de dos cuadros, tendrá que detectar cuándo se encuentra a una posición del final de la pantalla.

Convierta esto en un juego de frontón, usando aquella rutina que comprobaba cuándo un objeto choca contra otro. Recordará que se usaban los comandos ATTR o SCREEN$ para conseguirlo. Esta vez usaremos ATTR, ya que el juego es en color:

```
 1 PAPER 7: CLS
 2 LET go=1: REM SQUASH
 3 CLS : PRINT AT 0,0;"VEZ=";go
 4 LET s=1: LET d=1
 5 LET g=(RND*8)+6: LET h=13
 6 LET x=14: LET y=15
 7 GO SUB 100
 8 PRINT AT x,y; INK 4;CHR$ 131+CHR$ 1 31
10 IF INKEY$="5" THEN  GO TO 40
11 IF g>14 THEN  GO TO 300
13 PRINT AT g,h;"o"
14 PAUSE 2
15 GO SUB 200
20 IF INKEY$="8" THEN  GO TO 50
30 GO TO 10
40 PRINT AT x,y;"  "
41 LET y=y-1
42 IF y<6 THEN  LET y=6
43 PRINT AT x,y; INK 4;CHR$ 131+CHR$ 1 31
```

```
  44 GO TO 10
  50 PRINT AT x,y;"  "
  51 LET y=y+1
  52 IF y>24 THEN  LET y=24
  53 PRINT AT x,y; INK 4;CHR$ 131+CHR$ 1
31
  54 GO TO 10
  70 GO TO 20
 100 BORDER 1
 110 FOR a=5 TO 26
 120 PRINT  INK 2;AT 3,a;CHR$ 140
 130 NEXT a
 140 FOR a=4 TO 14
 150 PRINT  INK 5;AT a,5;CHR$ 138;AT a,2
6;CHR$ 133
 160 NEXT a
 170 RETURN
 200 PRINT  OVER 1;AT g,h;"o"
 201 PAUSE 2
 220 IF ATTR (g-d,h-s)=61 THEN  LET s=s*
-1
 230 IF ATTR (g-d,h-s)=58 THEN  LET d=d*
-1
 240 IF ATTR (g-d,h-s)=60 THEN  LET d=d*
-1
 245 LET g=g-d: LET h=h-s
 250 RETURN
 300 LET go=go+1
 305 PAUSE 50
 310 IF go>5 THEN  GO TO 400
 320 GO TO 3
 400 CLS : PRINT AT 10,12; PAPER 2; INK
6; FLASH 1;"FIN PARTIDA"
```

En alta resolución se usa POINT para saber si un punto de la pantalla ha sido ennegrecido con PLOT o no. Aquí tenemos como actúa:

```
10 PLOT 128,88
20 PRINT POINT (128,88)
```

Si ejecuta esto, verá un pequeño punto en el centro de la pantalla, y se imprimirá un 1 indicando que la posición 128,88 ha sido PLOTeada.

Si la posición no contiene punto, entonces el resultado de POINT es 0. Un simple ejemplo en el que se utilice esto puede ser:

```
5 LET a = 1
10 PLOT 0, 10: DRAW 255, 0
20 PLOT 0, 100: DRAW 255, 0
30 LET X = 120 : LET Y = 50
40 PLOT X,Y: PLOT OVER 1;X,Y
50 LET Y = Y + a
60 IF POINT (X,Y + A) = 1 THEN LET a = a* - 1
70 GO TO 40
```

Este programa dibuja dos líneas horizontales, y hace rebotar una pelota muy pequeña entre ellas. También puede usar POINT para saber si algo está en la pantalla o no. Por ejemplo, cuando ATTR no conviene y además se usan caracteres definidos (lo que impide la utilización de SCREEN$), entonces POINT es la única solución.

Obviamente, también se puede usar la rutina de las flechas para alta resolución, haciendo que las teclas 5 a 8 muevan un punto por la pantalla.

Un programa que dibuje (tipo tele-sketch) sería muy fácil de hacer usando esto. Lo único que hay que hacer es no borrar el último punto con OVER sino esperar a que se pulse una tecla (por ejemplo, la cero por DELETE) y entonces hacer que todos los PLOTs sean OVER hasta que se pulse otra tecla que anule esto (podría ser la P por PRINT).

ACELERANDO EL MOVIMIENTO

En muchos de los juegos en los que se utiliza el movimiento, puede ser difícil que el juego se ejecute a una velocidad aceptable. En el anterior programa del frontón, había sólo dos cosas que se movían, la pelota y la raqueta. Pero si hay más de dos cosas moviéndose en la pantalla al mismo tiempo, la velocidad puede reducirse mucho.

Vamos a considerar un simple juego de invasores, en el que por comodidad he usado la letra I para representar al invasor y un asterisco para representar el disparo.

```
10 FOR a =0 TO 31
20 PRINT AT 5,a; "I";AT 5,a;"^"
30 IF INKEY$ = "7" THEN GOSUB 200
40 NEXT a
50 GO TO 10
200 FOR b = 20 TO 5 STEP - 1
210 PRINT AT b,15;"*";AT b,15;"^"
220 NEXT b
230 RETURN
```

Como puede observar, con esta rutina la letra I se para mientras se efectúa el disparo. Una manera un poco burda de arreglar esto consiste en colocar un PAUSE X entre cada movimiento de la I. X debe ser tal que la pausa producida sea más o menos igual que el tiempo que tarda en llegar el disparo. Esto se debe hacer probando cuál es el número más indicado. Una vez se ha detectado la pulsación de la tecla 7, se salta a X la subrutina del disparo haciendo que X tome el valor mínimo, de este modo parecerá que la I no se detiene al lanzarle el disparo.

Para conseguir mayor velocidad en este tipo de juegos sin necesidad de programarlos en código máquina (véase la 5.ª parte del libro) puede colocar los caracteres directamente en el archivo de imagen mediante POKE. Esto supone enviar los caracteres que se van a imprimir directamente a la pantalla sin usar PRINT que tarda bastante más tiempo.

```
10 LET C = 0
20 LET A = 16384 + C
30 FOR B = 0 TO 7
40 POKE A, PEEK (USR "a" + B)
50 LET A = A + 32*8
60 NEXT B
70 LET A = 16384 + C
80 FOR D = 0 TO 7
90 POKE A,0
100 LET A = A + 32*8
110 NEXT D
120 LET C = C + 1
130 IF C = 31 THEN LET C = 0
140 GO TO 20
```

Ejecutando esto verá cómo la letra A se escribe en la pantalla de arriba a abajo y luego es borrada. Esto es casi ideal, pero al añadir una rutina para el disparo, el movimiento de la A se hará más lento.

El modo en que se POKEa A en la pantalla es un poco complicado debido a la extraña disposición del archivo de imagen en el Spectrum. En el apéndice, encontrará una representación del mapa de memoria del Spectrum. Allí puede ver, que el archivo de imagen, ocupa desde la posición 16384 hasta 22527. La pantalla está organizada en grupos de ocho puntos (un byte), del siguiente modo: La pantalla contiene 24 líneas (contando las dos últimas), pues bien, el fichero está dividido en tres partes, cada una de las cuales, contiene la información de un grupo de ocho líneas. Hasta aquí muy bien, pero dentro de cada parte, las primeras 32 posiciones corresponden a la parte de arriba de la primera línea, pero las 32 siguientes no corresponden al segundo byte de los caracteres que forman la primera línea sino que contiene la información de los primeros bytes de los caracteres de la segunda línea; es decir, primero van las partes de arriba de cada línea, luego la segunda hilera de puntos, y así sucesivamente. Las otras dos partes se estructuran de igual forma. Para ver lo que quiero decir, lo mejor es POKEar el fichero entero y ver el orden que sigue el Spectrum para llenar la pantalla. Si usted ha utilizado SAVE y LOAD con SCREEN$ ya sabrá de que estoy hablando. Escriba:

```
10 FOR A = 16384 TO 22527
20 POKE A, 225
30 NEXT A
```

Como consecuencia de esta distribución del archivo de imagen, resulta que los ocho bytes que forman un carácter no están seguidos sino que se encuentran a 32*8 posiciones de memoria de distancia entre ellos. Esto es lo que se tiene en cuenta en las líneas 50 y 100 del programa anterior. La primera vez la pantalla se POKEa con el carácter del gráfico definido en la tecla A y luego con un cero para borrarlo. Como puede apreciar, dada esta particular estructura es más fácil mover un carácter horizontalmente que hacerlo verticalmente.

Otra cosa más, he escogido el carácter gráfico definible A porque se puede reemplazar fácilmente por una nave, un invasor, o cualquier otra cosa.

AUTOEJECUCIÓN AUTOMÁTICA

Si usted tiene un programa cuyo listado no quiere que sea visto por el jugador, probablemente habrá intentado distintos métodos para impedirlo, como el truco del listado invisible que se dio anteriormente. Hay otra cosa que se puede hacer con respecto a esto y que además añade profesionalidad a sus programas. Es hacer que sus programas se ejecuten automáticamente después de recuperarlos del cassette mediante LOAD. Para hacerlo, se usa de nuevo LINE de esta manera:

9999 SAVE "Programa" LINE 100

Cuando haya escrito el programa escriba GOTO 9999 y el programa empezará a grabarse él mismo y cuando lo recupere con LOAD empezará a ejecutarse automáticamente desde la línea 100. Esto útil porque no borra las variables. Por supuesto, que usted puede poner cualquier número después de LINE.

USANDO SCREEN$

Si ha gastado casi toda la memoria disponible, en un programa largo y ve que necesita más espacio, piense que puede usar SAVE y LOAD con SCREEN$ para almacenar en cinta pantallas llenas de información, que a lo mejor sólo se necesita para explicar el juego al principio, o que se puede indicar al jugador que haga LOAD cuando sea necesario. Esto proporcionará gran cantidad de espacio.

UN RELOJ

Muchos de los juegos que usted quiere escribir serían mejores si pudiera disponer de un reloj para que controlara que las cosas se hacen en un tiempo determinado, etc. El Spectrum dispone de un reloj interno, que ocupa tres bytes en la variable del sistema **FRAMES.**

FRAMES se encuentra en las posiciones de memoria 23672, 23673, y 23674. La menor de ellas, la 23673, empieza a cero, y se incrementa en 1 cada dos centésimas de segundo. Cuando alcanza el valor 255 vuelve a cero otra vez y pone un 1 en 23673. La siguiente vez, pone un 2 en 23673, y cuando ésta llega a 255 se pone a cero y coloca un 1 en 23674.

Esto tiene un pequeño problema y es que al leerse las posiciones secuencialmente puede ocurrir que 23674 dé cero, pero esté a punto de cambiar a 1 y cuando se lee 23673, entonces 23674 ya esté a 1 y por lo tanto 23673 sea 0, obteniéndose una respuesta muy baja. Esto se puede evitar leyendo dos veces y tomando la más alta. Esta expresión da el tiempo, en segundos, que ha transcurrido desde que estaban los tres a cero (en principio desde que conectó su Spectrum a la red), pero usted puede ponerlos a cero cuando quiera usando POKE:

$$[(65536*A + 256*B + C)/50]$$

En esta expresión, A es PEEK 23674, B es PEEK 23673, y C es PEEK 23672. Los minutos se obtienen dividiendo por 60, y las horas se obtienen también muy fácilmente. En cualquier momento puede colocar el reloj a cero o bien a una hora determinada. Para ponerlo a las 9.00 usted deberá calcular: 9*60*60*50 = 1620000. Dividiendo esto por 65536 y tomanto la parte entera (con INT) se obtiene el número que debe "POKEar" en 23674, el resto de la división anterior, se divide por 256 y obtiene lo que debe "POKEar" en 23673, y el resto de esta última división en 23672. Es decir:

$$1630000 = 65536*24 + 256*184 + 32$$

Y usted hará: POKE 23674, 24: POKE 23673, 184: POKE 23672, 32

CÓMO HACER QUE ESPERE

En muchos juegos, usted querrá que el jugador se tome su propio tiempo para tomar una decisión, o incluso para leer algo. Una manera muy común de detener el programa

en espera del jugador, es esperar a que éste pulse una tecla. Por ejemplo:

```
300 IF INKEY$ = ""THEN GOTO 300
```

Sin embargo, el Spectrum parece tener problemas al trabajar con INKEY$. Una prueba de esto es el ejemplo que sigue, en el que el programa no espera a que usted pulse una tecla para seguir:

```
10 IF INKEY$ = ""THEN GOTO 10
20 PRINT "AQUÍ EMPIEZA EL PROGRAMA"
```

Esto se debe a que cuando usted pulsa RUN y luego ENTER, este último, es tomado por INKEY$ con lo cual se deja de cumplir la condición. Esto se arregla fácilmente añadiendo:

```
5 PAUSE 2
```

Recuerde este hecho en cualquier programa que utilice INKEY$. Sin embargo, tal como apuntó el escritor del libro del ZX81, puede usarse una manera mucho más corta para esperar a que se pulse una tecla, que también funciona en el Spectrum:

```
10 PRINT "PULSE UNA TECLA PARA EMPEZAR"
20 PAUSE 4E 4
30 PRINT "EMPIEZA EL PROGRAMA"
```

Lo bueno de esto es que 4E4 suena casi como "para siempre" (for ever) y puede ser fácilmente recordado. Sin embargo, por casualidad descubrí un modo más corto de hacerlo y más fácil de recordar. Simplemente es:

```
20 PAUSE 0
```

En el ZX81, este comando no espera nada, pero en el Spectrum, se espera indefinidamente, hasta que se pulsa una tecla. La ventaja sobre el otro es que 4E4 no es mucho tiempo si la persona tiene que leer algo, entonces PAUSE 0 es la solución ideal.