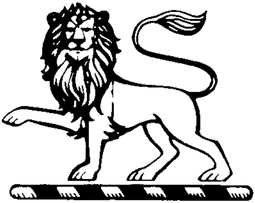
pride utilities

SUPERSPRITES

For the
AMSTRAD
CPC 464/664/6128



Unit 14, 6 Union Street,
White Hill House,
Luton, Beds. LU1 3AN

WRITTEN BY R. POOLE

SUPERSPRITES

(MODE Ø)

CPCs 464/664/6128

The program enables its user to design multi-coloured animated sprites of various sizes. Any sprites designed using the program are stored in the computers memory for future use or modification. Included in the package are a suite of machine code routines, these routines give the user control over any sprites designed from additional basic commands (R.S.X's) or they can be called from within an assembly language program (machine code). The new commands include:-

Sprite positioning
Movement in eight directions
Animation control
Collision detection
Sprite swapping

and various other commands allowing split screen effects.

The program offers facilities to save, onto a disc drive or tape recorder, a file consisting of all the available machine code routines and information on any sprites designed.

The maximum number of animated sprites that can be constructed is sixty, however each sprite can consist of up to 4 frames.

The term "frames" simply means the number of picture elements required to produce animation. From experience it has been found that four frames is usually enough to create effects such as rotating helicopter blades, walking characters and the majority of animated effects seen in todays arcade games.

The information that follows is split into three sections:-

1. The main menu selections (1 - 7).

2. Basic Commands.

3. Using the routines from machine code.

To load the program type

```
RUN"SPRITE"
```

THE MENU SELECTIONS

Selection 1: Creating a sprite.

This is simply a matter of plotting pixels within the sprite grid of the right colour. The keys used for this process are those on the Numeric Keypad. They are used to move the cursor on the sprite grid and all are direction keys with the exception of key 5. Key 5, when held down, allows the pixels to be plotted or erased depending on the mode selected (Plot or Erase). There is an on-screen indication of the mode selected represented by M=P or M=E. The mode is changed by pressing the large blue Enter key. If key 5 is not held down then the flashing cursor will skip over the sprite grid without effecting any previously plotted pixels.
A joystick, if connected, may be used in addition to the numeric keypad, the fire button acting as key 5.

KEYS USED ON THE MAIN CONSOLE

Key 1: moves the position of the pen cursor(>). The pen cursor is on the left of the screen display and points to the current pen selection.

Keys 2/3: alter the ink colours in the current pen.

Key 4: changes the colour of the Border.

Enter key: toggles the mode between Plot and Erase.

Keys Z/X: increase/decrease the size of the sprite grid selected.

Key C: used with the SHIFT key to clear any plotted pixel in the sprite grid.

Key H: will horizontally mirror the contents of the sprite grid.

Key V: will vertically mirror the contents of the sprite grid.

Key L: locks the sprite grid size selected and disables keys Z and X. Its purpose is to prevent accidental changes in grid size during the process of producing a sprite.

Key S: used with the SHIFT key to store the pattern drawn on the sprite grid in the computers memory, allocating to it the sprite number shown on the screen display. The lock (key L) must be ON otherwise no action is taken.

Key M: returns the user to the main menu aborting anything drawn in the sprite grid.

Selection 2: Modify a sprite.

The procedure used to modify a sprite is to set up the sprite number and frame number, using keys 1/2 for increasing/decreasing the sprite number and keys 3/4 to increase/decrease the frame number. After setting up the required sprite and frame number press key R. This will reconstruct the sprite and may take over a minute if it is large. The reconstruction is complete when the flashing cursor reappears at the bottom left hand corner of the sprites grid. To make alterations to the sprite use the facilities available under selection 1. After alteration the sprite may be stored as previously described, it will, however, overwrite the previous shape stored in the computers memory.

Selection 3: Select Tape or Disc.

The program automatically senses whether a disc drive it connected to the computer. If one is connected all loading and saving of files will use it until altered by by this selection. On screen prompts allow selection of the following combinations:

A. Disc: Saving/Loading carried out on the disc drive (if connected).

B. Tape: Saving/Loading carried out to a cassette recorder.

C. Disc in/Tape out: All files loaded from the disc drive and all files saved to the cassette recorder.

D. Disc out/Tape in: All files loaded from the cassette recorder and all files saved to the disc drive.

Selection 4: Save/Load a file.

This selection allows the user to save or load files from/to a disc drive or cassette recorder depending which is currently selected. Any file saved consists of all sprite patterns produced and the machine code routines. After a file has been saved and the computer reset or switched off there are two methods of loading a file back in. The method chosen depends on whether the user wishes to modify the sprites or work with them using the new commands. (EG perhaps to write a game).

Method A:

After switching the computer on load the Supersprites program (RUN"SPRITES) and use selection 4 on the menu to load the sprite file. When loading is complete all the facilities of the Supersprites program are available. This method is primarily used to modify sprites, however it is possible to exit the program by using selection 7 on the menu.

Method B:

After switching on the computer, type in the following commands:

```
MEMORY 19999
LOAD"FILENAME"
CALL 42472
```

The filename from cassette files is always "SPRITE.SPR". The filename on disc is that allocated to it during the saving of the file. The file type is always .SPR. The CALL instruction is necessary to log on all the additional commands. The memory command is set at 19999 to allow room for the biggest sprite file possible. However it is possible to find the actual last byte of a file as show below. It is then possible to alter the memory command to this value (-1) giving the user the maximum amount of memory to use in Basic or assembly language programs.

```
File last byte =  PEEK(39999)*256+PEEK(39998)
Length of a sprite file = 42747 - last byte.
```

Selection 5: List of all keys used.

This selection lists all the keys used by the sprite editor that are not given by on-screen prompts. A brief description of each key's function is also given.

Selection 6: Demo mode.

The demo selection from the main menu has two functions:

A. Will display any sprites that have been constructed, either stationary or moving, and in both cases showing any animated effects produced by multiple frames.

B. To load a demo .SPR file from the disc or cassette and enter a short demonstration program. The demonst-ation is in the form of a single screen game and it is written completely in BASIC using the new commands. The purpose of this demonstration is to show that it is possible to write arcade games in basic with the help of the new commands. However, it should be noted that if the routines that perform the work of the new commands were to be called from within a machine code program, a lot more could be achieved.

The object of the game is to collect all the fruit on the screen and ring the swinging bell. Good luck, because it is not easy. The arrow keys are used to move the man left or right and the up arrow key to make him jump The game may be aborted at any time by pressing key M, which will return the user to the main menu. It is possible to load this DEMO.SPR file using selection 4 on the menu, thus enabling the user to look at and modify the sprites in the file. Care should be taken when saving the modified file to disc that a different save name is used so as not to overwrite the original file.

Selection 7: Return to BASIC.

This last menu selection returns the user to BASIC after asking for confirmation. Once in BASIC the sprite editor is removed from the computers memory. However all the sprite patterns and routines are still retained giving immediate access to the facilities offered by the additional basic commands (R.S.X's). If the user should accidentally find himself in the BASIC environment prior to saving a sprite file it is possible to save the file by the following command sequence:

Enter as a direct command: PRINT PEEK(39999)*256+PEEK(39998)
(This will give the Start address)

42747-(start address) will give the length

Enter as a direct command: SAVE"FILENAME.SPR",B,start,length

BASIC COMMANDS

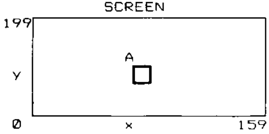
Each new basic command must be preceded by the bar symbol (¦), obtained by pressing SHIFT and "@". All the new command names can be entered in either upper or lower case letters. Several of the new commands have parameters associated with them and if these are accidentally left out the command will be ignored. If the parameters are outside the permissable range then either the command will be ignored or the parameters will be forced by the command routine to within legal limits. After the list of new commands there is a brief description of each command and any parameters associated with it.

NEW COMMANDS

|POS,number,x coordinate,y coordinate	(sprite positioning)
|SPRITE,number,mode	(sprite control)
|SWAP,number,number	(swap one sprite for another)
|VMIRROR,number	(vertically mirror sprite pattern)
|HMIRROR,number	(horizontally mirror sprite pattern)
|HIT,number,number	(collision detection between two sprites)
|HTOP,number	(top collision with background scenery)
|HBOT,number	(bottom collision with background scenery)
|CLS	(clear the screen)
|SCORE,x coordinate,y coordinate,value	(prints a 5 digit score)
|RESET,x coordinate,y coordinate	(resets score to 5 zero´s)
|DI	(disable interrupts during sprite movement)
|EI	(enable interrupts during sprite movement)
|SPLIT,top mode,bottom mode,size	(operate screen in 2 modes)
|SPLITOFF	(turn off split screen)
|BOT	(allow basic printing on screen during split screen)
|TOP	(allow basic printing on screen during split screen)
|COL,size,PEN 0,PEN 1,PEN 2,PEN 3,BORDER	(8 colours in mode 1 only)
|COLOFF	(turn off mode 1 split colour)

|POS command

This command is used to set up the sprite starting position. The number of parameters can be any number referring to a valid sprite between 1 and 60. The "x" coordinate can be any number between 0 and 159 and the "y" coordinate any number between 0 and 199. Coordinates 0,0 refer to the bottom left corner of the screen.



Note:
Incorrect coordinates will be forced into legal coordinates by the command routine. The coordinates in the |POS command refer to the top left of the sprite as shown by point "A" in the diagram.

(|POS continued.)

The user should note that this command does not allow plotting of positions not on the screen and in practise any "x" or "y" values set at the limits will automatically have the width or height of the sprite deducted by the appropriate command routine.

This example will position sprite number 1 in the bottom left of the screen:

```
              |POS,1,0,0     (sets up the position)
              |SPRITE,1,0    (puts sprite 1 onto screen)
```

|SPRITE command

|SPRITE,number,mode:

Number - any valid sprite number between 1 and 60.
Mode - decides what action the command will perform as follows:

```
Mode 0: Place sprite on/off screen.
     1: Animate, change next frame.
     2: Move sprite down two pixels.
     3: Move sprite up two pixels.
     4: Move sprite left two pixels.
     5: Move sprite right two pixels.
     6: Move sprite up and right two pixels.
     7: Move sprite up and left two pixels.
     8: Move sprite down and right two pixels.
     9: Move sprite down and left two pixels.
```

The sprite command calls the machine code routine "wait for frame flyback" to allow flicker free movement. It is possible to remove this by adding the value 16 to the mode parameter.

This can be very useful because if several sprites are to be moved only one call to the "wait for flyback" routine may be necessary.

```
Example: A. MODE 0
            |SPRITE,1,0    (put sprite 1 onto screen)
            |SPRITE,1,4    (move sprite 1 left 2 pixels)

         B. MODE 0
            |SPRITE,1,0
            |SPRITE,1,20
```

Examples A and B are identical except that B is without frame flyback and is therefore faster than A.

|SWAP command

This command allows the user to swap one sprite for another. The new sprite taking up the number and coordinates of the old sprite.

```
Example:     |SWAP,1,2
```

Sprite 2 becomes sprite 1 and takes up the coordinates of number 1. Sprite 1 becomes sprite 2 but remains at the same coordinates. When this command is used the sprite concerned should be removed from the screen before the swap takes place and replaced afterwards if required.

```
Example:  10 MODE 0
          20 |SPRITE,1,0              (put onto screen)
          30 FOR B=1 TO 5000:NEXT     (delay)
          40 |SPRITE,1,0              (remove from screen)
          50 |SWAP,1,2                (exchange sprites)
          60 |SPRITE,1,0              (put back onto screen)
```

|VMIRROR command

The number parameter can be any valid sprite number between 1 and 60 as previously described. The command routine vertically mirrors the specified sprite. In the following example if sprite 1 was a triangle then this routine would turn it upside down:

```
10 MODE 0
20 |SPRITE,1,0              (put onto screen)
30 FOR B=1 TO 5000:NEXT     (delay)
40 |SPRITE,1,0              (remove from screen)
50 |VMIRROR,1               (vertical mirror)
60 |SPRITE,1,0              (put onto screen)
```

|HMIRROR command

This command is identical to the |VMIRROR command except that it horizontally mirrors the specified sprite. In the following example if sprite 1 was ">" then the routine would produce "<":

```
10 MODE 0
20 |SPRITE,1,0              (put onto screen)
30 FOR B=1 TO 5000:NEXT     (delay)
40 |SPRITE,1,0              (remove from screen)
50 |HMIRROR,1               (horizontally mirror)
60 |SPRITE,1,0              (put onto screen)
```

:HIT command

This command checks for a collision between two sprites. It ignores all background scenery and only checks for collision between sprites numbered in the parameters. Information concerning a collision is stored at memory location 39894. It is possible to PEEK this location to ascertain if a collision has occurred. The routine does not allow collision detection between sprites that are not on the screen. In the following example two sprites are moved toward each other, when the collision occurs the routine will print on the screen THEY HAVE COLLIDED:

(39894=0 no collision - 39894=1 collision)

```
10 MODE 0
20 :POS,1,0,100::POS,2,100,100
30 :SPRITE,1,0::SPRITE,2,0
40 FOR B=1 TO 30::SPRITE,1,5::SPRITE,2,4
50 :HIT,1,2:IF PEEK(39894)<>0 THEN 70
60 NEXT:STOP
70 LOCATE 1,18:PRINT"THEY HAVE COLLIDED"
```

Note: The user must first construct sprites 1 and 2.

:HTOP command

This command checks for a collision between the top of the sprite and the background scenery. It does this by comparing the sprite pattern in memory with that on the screen, therefore this command will not be able to check sprites off the screen. Memory location 39894 is used to store the result of the check as follows:

39894=0 (Top of the sprite has not collided with scenery)
39894>0 (Top of sprite has collided with scenery)

The command routine treats anything that is not INK 0 as background scenery, this also includes other sprites.

:HBOT command

Identical to the :HTOP command except that it checks the bottom of the sprite for collision. The demonstration program in this package give an example of this command being used. In the example it is used to check whether the man is standing on a platform. Once again the result of the check is stored at 39894.

39894=0 (Bottom of the sprite is not on scenery)
39894>0 (Bottom of the sprite is on scenery)

:CLS command

This command must be used to clear the screen instead of the BASIC command CLS. The reason for this is that the hit detection routines must know whether a sprite is on the screen or not, to enable them to perform valid checks. The :CLS command resets flags that indicate which sprites are actually on screen. The BASIC command can be used as normal because it has been intercepted to allow the flags to be reset.

:SCORE command

This command can be used in any screen mode and even in split screen mode. The "x" and "y" coordinates are dependent on the screen mode selected as follows:

MODE 0	x = 1 to 20	y = 1 to 25
MODE 1	x = 1 to 40	y = 1 to 25
MODE 2	x = 1 to 80	y = 1 to 25

This routine will print a 5 digit score on the screen at the "x' and "y" coordinates specified. Leading zero's are printed

The parameter "value" can be any number between 1 and 65535. This value is added to the score before it is printed onto the screen. The score is a running total until it is either reset by the :RESET command or until it is reset by exceeding the routines limit of 65535. This upper limit is because the score is held in two bytes of memory at locations 39468 and 39469. The maximum number that can be held in two bytes is as follows:

Maximum = 255+(255*256) = 65535

```
Example:     :SCORE,10,1,100    (prints 00100 on screen)
             :SCORE,10,1,45     (prints 00145 on screen)
             :SCORE,10,1,1000   (prints 01145 on screen)
```

:RESET command

This command is associated with the :SCORE command. It resets the score to zero and print 5 zero's at the coordinates specified.

```
Example:     :RESET,10,1       (prints 00000 on the screen)
```

:DI command

This command disable all interrupts during sprite movement and should not be used if the Amstrad real time clock is to be maintained or if the split screen commands are to be used.

|EI command

This command enables all interrupts during sprite movement. It is only necessary to use this command if the interrupts have been disabled during sprite movement with the |DI command and only then if the user wishes the interrupts to be re-enabled.

PLEASE NOTE All the commands described so far have been designed to be used in MODE 0 except for the |SCORE command which can be used in any mode. The commands that follow are used to generate split screen operation (EG: half the screen in MODE 0 and half the screen in MODE 1 or 2). It is possible to use the sprite commands in the split screen mode providing the interrupts are not disabled and the sprites are kept within the MODE 0 section of the screen.

|SPRITE command

This command allows the user to split the screen into two modes at the same time. Any combination of the modes 0,1 or 2 can be used. The command has 3 parameters. The first two "TOP MODE" and "BOTTOM MODE" can be any mode (0/1/2). TOP mode refers to the top of the screen and BOTTOM mode to the remainder of the screen. The third parameter "SIZE" decides how many quarters of the screen display the "TOP" mode occupies. Only numbers between 1 and 3 are valid and numbers outside this range will be altered by the command routine.

```
Examples: A. |SPLIT,1,0,2
          B. |SPLIT,0,2,3
```

This will split the screen as follows:

A. The top two quarters (half of screen) in MODE 1 and the remainder in MODE 0.
B. The top three quarters in MODE 0 and the remainder of the screen in MODE 2.

The split command has two other other commands associated with it to allow BASIC printing in either section of the screen display. These commands are |TOP and |BOT.

|TOP command

This command has no parameters and is used during the split screen operation to print in the top section of the screen display.

```
Example:    |SPLIT,0,1,2:|TOP:LOCATE 5,5:PRINT"HELLO"
```

|BOT command

This command is identical to the |TOP command with the exception that it prints to the bottom section of the screen during the split screen operation.

Example: `|SPLIT,1,2,2:|BOT:LOCATE 20,20:PRINT"HELLO"`

|SPLIT command

This command turns off any split screen previously selected. The TOP mode now covering all the screen.

|COL command

This command can only be used in MODE 1. It enables the user to have eight colours on the screen at the same time, twice the normal number. The command has six parameters:

Size: This can be any number between 1 and 3 and refers to the number of quarters of the screen, starting at the top of the screen that the new pen colours occupy. The remaining quarters of the screen using the normal pen colours.

PEN 1/PEN 2/PEN 3/BORDER: These can be of any ink number between 0 and 26, the numbers representing those used by Amstrad and shown in the colour chart below.

Example: If the PEN 0 parameter was 26 then PEN 0 would contain bright white ink.

COLOUR CHART

0 = Black	9 = Green	18 = Bright Green
1 = Blue	10 = Cyan	19 = Sea Green
2 = Bright Blue	11 = Sky Blue	20 = Bright Cyan
3 = Red	12 = Yellow	21 = Lime Green
4 = Magenta	13 = White	22 = Pastel Green
5 = Mauve	14 = Pastel Blue	23 = Pastel Cyan
6 = Bright Red	15 = Orange	24 = Bright Yellow
7 = Purple	16 = Pink	25 = Pastel Yellow
8 = Bright Magenta	17 = Pastel Magenta	26 = Bright White

Example: `|COL,2,11,0,9,16,13`

Would set the colours for the top half of the screen:

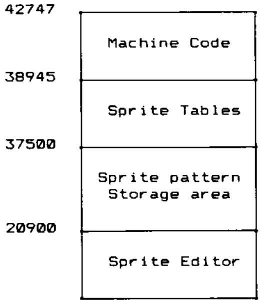
```
PEN 0 = Sky Blue     PEN 1 = Black
PEN 2 = Green        PEN 3 = Pink
         BORDER = White
```

|COLOFF command

This command turns off any previous split colour command. This command must be used if a split colour screen has already been defined before defining a second.

MACHINE CODE SECTION

SUPERSPRITES MEMORY MAP



The machine code routines associated with the additional basic commands and the sprite editor are stored in memory from 38945 to 42747. These routines are position dependant and may not be moved elsewhere. The memory locations from 37500 to 38944 (labelled Sprite Tables), is storage area used by the sprite editor to store information about each sprite produced. Each sprite is allocated 24 bytes of memory, sprite number 1 starting at 37500. The information stored is as follows:

Byte 1 = x coordinate (0 to 159)
Byte 2 = y coordinate (0 to 199)
Byte 3 = Height of sprite in bytes
Byte 4 = Width of sprite in bytes
Byte 5 = Points to which frame is active
Byte 6 = Animation (0 = no - 1 = yes)
Bytes 7 to 22 contain pointers indicating where in memory the binary patterns for each frame of a sprite are stored.
Byte 23 = Total number of frames in a sprite (1 to 4).
Byte 24 = Sprite grid size used by the editor.

Sprite pattern storage area: 37499 to 20899. The amount of memory used to store sprite patterns depends on how many sprites are produced. The sprite editor allocates memory downwards from the upper limit of 37499 to the lower limit of 20899. This always leaves the programmer the maximum amount of memory possible for basic or machine code. The last byte of a sprite file can be found as previously described in Menu selection 4.

Using each command from machine code

As this section is not for beginners it will be assumed that the user understands assembly language programs written in mnemonic form. All programs listed will be in mnemonics.

Exit conditions

The exit conditions of all the commands called from machine code are as follows:

AF corrupt, HL corrupt, DE corrupt, BC corrupt, IX corrupt, IY corrupt. Alternate register set as on entry.

|SPRITE,number,mode (sprite control)

```
Table: Defw mode   (valid sprite modes 0 - 9)
       Defw number (valid sprite numbers 1 - 60)
Start: ld ix,Table
       ld a,2
Call 40766
ret
```

|POS,number,x,y (set up sprite position)

```
Table: Defw y      (0 - 199)
       Defw x      (0 - 159)
       Defw number (1 - 60)
Start: ld ix,table
       ld a,3
Call 40808
ret
```

|SWAP,number,number (swap one sprite for another)

```
Table: Defw number    (1 - 60)
       Defw number    (1 - 60)
Start: ld ix,table
       ld a,2
Call 38973
ret
```

|VMIRROR,number (Vertically mirror sprite)

```
Table: Defw number    (1 - 60)
Start: ld ix,table
Call 39895
ret
```

¦HMIRROR,number (Horizontally mirror sprite)

```
Table: Defw number     (1 - 60)
Start: ld ix,table
Call 42538
ret
```

¦HIT,number 1,number 2 (Check collision between 2 sprites)

```
Table: Defw number 2  (1 - 60)
       Defw number 1  (1 - 60)
Start: ld ix,table
       ld a,2
Call 41800
ret
```

The result is stored in memory location 39894 (0 = no collision and 1 = collision)

¦HTOP,number (check for collision at top of sprite)

```
Table: Defw number     (1 - 60)
Start: ld ix,table
       ld a,1
Call 41885
ret
```

The result is stored in memory location 39894 (0 = no collision and >0 = collision). If a collision is registered after returning from this routine then the value held in location 39894 = value of the byte where the collision was first detected.

¦HBOT,number (check for collision at bottom of sprite)

```
Table: Defw number     (1 - 60)
Start: ld ix,table
       ld a,1
Call 41925
ret
```

The result is as described for ¦HTOP.

¦CLS (clear screen)

```
Call 39096
ret
```

|SCORE,x,y,value (print 5 digit score)

```
Table: Defw value       (0 - 65535)
       Defw y           (0 - 199)
       Defw x           (0 - 159)
Start: ld ix,table
       ld a,3
Call 39365
ret
```

Note that this routine will only hold scores up to 65535 correctly.

|RESET,x,y (reset 5 digit score to zero)

```
Table: Defw y           (0 - 199)
       Defw x           (0 - 159)
Start: ld ix,table
Call 39428
ret
```

|DI (disable interrupts during sprite movement)

```
Call 39252
```

|EI (enable interrupts during sprite movement)

```
Call 39212
```

|SPLIT,top mode,bottom mode,size (split screen operation)

```
Table: Defw size          (1 - 3)
       Defw bottom mode   (0 - 2)
       Defw top mode      (0 - 2)
Start: ld ix,table
Call 41100
ret
```

|TOP (allows printing in the top section of the screen using the firmware routines)

```
Call 41314
```

|BOT (allows printing in the bottom section of the screen the firmware routines

```
Call 41238
```

|SPLITOFF (turn off any split screen)

```
Call 41335
```

|COL,size,PEN 0,PEN 1,PEN 2,PEN 3,BORDER (select split colour in MODE 1

```
Table: Defw border      (0 - 26)
       Defw PEN 3       (0 - 26)
       Defw PEN 2       (0 - 26)
       Defw PEN 1       (0 - 26)
       Defw PEN 0       (0 - 26)
       Defw size        (1 - 3)
Start: ld ix,table
       ld a,6
Call 41477
ret
```

|COLOFF (turn off any split colour mode)

```
Call 41460
```

Demonstration Program - Basic Listing

A listing of the demonstration program appears on page 18. This demonstration is contained within the SuperSprites BASIC program from lines 650 to 830. The listing has been renumbered and reformatted it to suit this instruction book.

```
10 CLEAR:CALL 42689:¦DI:CALL 47944:BORDER 3:MODE 0
20 PEN 1:DRAW 639,0:FOR a%=1 TO 5:LOCATE 16,13+a%:PRINT CHR$(
151);CHR$(157):NEXT:LOCATE 3,13:FOR a%=1 TO 13:PRINT CHR$(254
);:NEXT:LOCATE 3,6
30 FOR a%=1 TO 5:PRINT CHR$(254);:NEXT:LOCATE 10,6:FOR a%=1 T
O 7:PRINT CHR$(254);:NEXT:LOCATE 18,6:PRINT CHR$(254);CHR$(25
4);CHR$(254):LOCATE 6,19:FOR a%=1 TO 15:PRINT CHR$(254);:NEXT
:FOR a%=1 TO 5:LOCATE 9,20+a%:PRINT CHR$(151);CHR$(157):NEXT
40 ¦POS,8,8,199:¦SPRITE,8,0:¦POS,7,146,180:¦SPRITE,7,0:¦POS,4
,128,175:¦SPRITE,4,0:¦POS,1,120,72:¦SPRITE,1,0:¦POS,2,0,0:¦SP
RITE,2,0:¦POS,3,40,120:¦SPRITE,3,0:¦POS,5,20,136:¦SPRITE,5,0
50 ¦POS,12,30,98:¦SPRITE,12,0:¦POS,13,12,156:¦SPRITE,13,0:¦PO
S,11,140,50:¦SPRITE,11,0:¦POS,14,60,160:¦POS,9,150,120:¦SPRIT
E,9,0:¦POS,10,120,199:¦SPRITE,10,0
60 EVERY 4,3 GOSUB 230:EVERY 3,0 GOSUB 260:EVERY 40,1 GOSUB 3
30:EVERY 25,2 GOSUB 280
70 EI:IF g%=1 THEN 180
80 IF g%=2 THEN 160
90 IF INKEY(1)=0 THEN GOSUB 120 ELSE IF INKEY(8)=0 THEN GOSUB
140
100 IF INKEY(38)=0 THEN STOP
110 IF INKEY(0)=0 THEN 200 ELSE DI:¦HBOT,2:IF PEEK(39894)>0 T
HEN 70 ELSE 160
120 IF k%=1 THEN ¦SPRITE,2,0:¦HMIRROR,2:¦SPRITE,2,0:k%=0
130 ¦SPRITE,2,21:¦SPRITE,2,17:RETURN
140 IF k%=0 THEN ¦SPRITE,2,0:¦HMIRROR,2:¦SPRITE,2,0:k%=1
150 ¦SPRITE,2,20:¦SPRITE,2,17:RETURN
160 EI:LOCATE 1,1:PRINT CHR$(7):FOR a%=1 TO 25:¦SPRITE,2,18:N
EXT:DI:¦CLS:IF k%=1 THEN ¦HMIRROR,2
170 CLEAR:EI:GOTO 10
180 DI:¦CLS:LOCATE 6,10:PRINT"WELL DONE.":FOR a%=1 TO 5000:NE
XT:CALL 42105:STOP
190 DI:CALL 42105:CALL 42698:STOP
200 EI:¦SPRITE,2,0:¦SWAP,2,6:¦SPRITE,2,0:FOR a%=1 TO 16:¦SPRI
TE,2,19:GOSUB 220:NEXT:GOSUB 350:FOR a%=1 TO 16:FOR b%=1 TO 2
5:NEXT:¦SPRITE,2,18:GOSUB 220:¦HBOT,2:IF PEEK(39894)>0 THEN 2
10 ELSE NEXT:¦SPRITE,2,0:¦SWAP,2,6:¦SPRITE,2,0:GOTO 160
210 ¦SPRITE,2,0:¦SWAP,2,6:¦SPRITE,2,0:GOTO 70
220 t%=(t%+1)MOD 5:IF t%=4 THEN ¦SPRITE,2,17:RETURN ELSE RETU
RN
230 DI:z%=(z%+1)MOD 70:IF z%<35 THEN ¦SPRITE,1,20:¦SPRITE,3,2
1:¦SPRITE,4,18 ELSE ¦SPRITE,1,21:¦SPRITE,3,20:¦SPRITE,4,19
240 y%=(y%+1)MOD 3:IF y%=2 THEN ¦SPRITE,1,17:¦SPRITE,3,17
250 EI:RETURN
260 DI:x%=(x%+1)MOD 80:IF x%<40 THEN ¦SPRITE,5,21 ELSE ¦SPRIT
E,5,20
270 EI:RETURN
280 DI:¦HIT,2,1:IF PEEK(39894)>0 THEN 320
290 ¦HIT,2,3:IF PEEK(39894)>0 THEN 320
300 IF f%=0 THEN ¦HIT,2,4:IF PEEK(39894)>0 THEN 320
310 EI:RETURN
320 g%=2:LOCATE 1,1:PRINT CHR$(7):EI:RETURN
330 DI:¦SPRITE,14,0:¦SPRITE,7,17:SOUND 2,5,5,7,8:¦HIT,2,7:IF
PEEK(39894)>0 THEN g%=1
340 EI:RETURN
350 FOR c%=8 TO 13:¦HIT,2,c%:IF PEEK(39894)>0 THEN 360 ELSE
NEXT:RETURN
360 DI:¦SPRITE,c%,0:EI:h%=h%+1::LOCATE 1,1:PRINT CHR$(7):IF
h%=6 THEN f%=1:RETURN ELSE RETURN
370 LOCATE 1,1:PRINT CHR$(7):GOTO 310
```