

CREATE GIOCHI ARCADE COL VOSTRO SPECTRUM

con ampie descrizioni di oltre 30 routines e 18 fantastici programmi.

di DANIEL HAYWOOD



edizioni

Jce

CREATE GIOCHI ARCADE COL VOSTRO SPECTRUM

con ampie descrizioni di oltre 30
routines e 18 fantastici programmi.

di Daniel Haiwood

traduzione di Angelo Cattaneo



Via dei Lavoratori, 124
CINISELLO BALSAMO (MI)

Tutti i diritti sono riservati, nessuna parte di questo libro e della cassetta software allegata può essere riprodotta, posta in sistemi di archiviazione, trasmessa in qualsiasi forma o mezzo elettronico, meccanico, di fotocopiatrice, ecc., senza l'autorizzazione scritta dell'Editore.

Nel testo sono stati introdotti programmi di valore didattico. L'Editore non risponde dei possibili errori che si verificano nei listati e nei relativi risultati.

Prima edizione: INTERFACE PUBLICATIONS 1983

Pubblicato in Gran Bretagna da:

Interface Publications,

44-46 Earls Court Road,

London, W8 5BJ.

Copyright © Daniel Haywood 1983

Copyright © per l'edizione italiana: Edizioni JCE, 1984

Prima edizione: Giugno 1984

Stampato in Italia da:

Gemm Grafica S.r.l.

Via Magretti-Paderno Dugnano (MI)

INDICE

Prefazione

-Tim Hartnell

Capitolo 1

-Lettura della Tastiera

"Sketchpad"

Capitolo 2

-Grafica definita dall' utente

"Generatore di caratteri", "Olocausto"

Capitolo 3

-Suoni

Capitolo 4

-Uso degli Operatori Logici

"Bomber"

Capitolo 5

-Movimento

"Zap", "Invaders", "Snakes", "Squash!"

Capitolo 6

-Scrolling

"Meteors", "Gran Prix", "Scramble", "Slalom"

Capitolo 7

-Uso degli array

"Flea race", "Snakey", "ICBM"

Capitolo 8

-PEEK e POKE

"Circuit" parte prima e seconda, "Chomper", "Raindrops"

Capitolo 9

-Come rendere piu' interessanti i Games.

PREFAZIONE - Tim Hartnell

D' ora in poi non dovrete piu' sacrificare le cento lire in sala giochi; infatti, con l' aiuto dei programmi presentati in questo libro (sempre accompagnati dai chiarimenti tecnici del caso) potrete ottenere gli stessi identici effetti.

L' autore Daniel, nello stendere l' opera, presume che voi conosciate gia' le basi di programmazione dello Spectrum per cui procede spedito presentando i programmi piu' interessanti senza tralasciare pero' la descrizione del perche' venga seguita una certa strada anziche' un' altra.

Il contenuto didattico trova posto alla fine di ogni programma dove e' riportata la spiegazione dei vari blocchi che lo compongono. Nonostante, siamo certi che il volume verra' apprezzato principalmente per il suo donare all' utente diverse ore di divertimento, lo stesso che abbiamo provato noi nel verificare i listati per la pubblicazione.

Tim Hartnell

Tim Hartnell, oltre ad essere il fondatore del Gruppo Nazionale Inglese degli ZX Users, ha anche scritto una trentina di libri sugli ZX e su altri computers.

LETTURA DELLA TASTIERA

Il riuscire a "leggere" la tastiera, e' indispensabile per poter scrivere un programma di giochi. Questo primo capitolo spiega appunto le varie funzioni legate alla tastiera e in quale modo esse agiscano.

INKEY\$ rende il carattere del tasto premuto sottoforma di stringa esplorando la tastiera in una frazione di secondo. Così' come in altri computers, tale funzione ha una azione "continuativa", cioè' continua a leggere il carattere relativo al tasto per tutto il tempo in cui questo rimane premuto.

INKEY\$ agisce anche in combinazione con i tasti di Shift rendendo i relativi caratteri maiuscoli o i relativi simboli. CODE INKEY\$, come e' intuibile, porta il codice del carattere memorizzato da INKEY\$ il quale, nel caso non sia azionato alcun tasto, contiene una stringa vuota (ad es. " "). In questo caso CODE INKEY\$ ritorna il valore zero.

Lo Spectrum possiede anche un secondo comando molto usato : IN. Il vantaggio di "IN" rispetto ad "INKEY\$" e' che puo' essere usato per leggere piu' di un tasto contemporaneamente; cosa che INKEY\$ non puo' fare.

Provate a battere :

```
10 POKE 23692,-1:PRINT INKEY$:GO TO 10
```

e rilevate cosa succede premendo due tasti contemporaneamente come potrebbe capitare durante un games di invaders. La risposta e' semplice : non succede nulla in base a quanto sopra spiegato.

La funzione di INKEY\$ viene prevalentemente usata per variare le coordinate delle colonne a cui sono riferiti gli oggetti (ad esempio una base missilistica). Un sistema per effettuare dette variazioni e' quello di adottare gli statements IF...THEN, molto piu' comodi che non l'uso di operatori logici. Nella routine e' combinato anche il controllo per evitare che la base missilistica non provochi errori portandosi all' esterno dei margini dello schermo.

'p' contiene le coordinate delle colonne della base missilistica.

Versione 1: IF...THEN

```
100 IF INKEY$="8" AND p<31 THEN LET p=p+1  
110 IF INKEY$="5" AND p>0 THEN LET p=p-1
```

Versione 2: Operatori logici

```
100 LET p=p+(INKEY$="8" AND  
p<31)-(INKEY$="5" AND p>0)
```

INKEY\$ puo' anche venir impiegato per stabilire limiti di tempo. In questo caso e' sottoposto alla funzione PAUSE la quale puo' essere interrotta con la pressione di un tasto qualsiasi per poter far riprendere al programma il suo regolare svolgimento. Una forma del genere e' usata assai spesso. Eccovi un esempio

```
10  
20  
.  
.  
.  
.  
.  
5499 REM Altra partita ?  
5500 PRINT AT 5,2;"Vuoi giocare ancora ?";  
      AT 7,0;"Hai 5 secondi  
per rispondere s/n"  
5510 PAUSE 250  
5520 IF INKEY$<>"s" THEN PRINT AT 10,13;"O.K." :  
      STOP  
5530 RUN
```

In unione a GOTO, la funzione INKEY\$ sostituisce efficacemente gli statements IF...THEN come potete verificare dalla parte finale del programma che segue

```
1000 GO TO 5000  
2000 REM Programma presente  
3000 PRINT "Prog."  
4000 BEEP 1,RND*60-30  
5000  
6000 REM Ultimo uomo perso  
7000 GO TO 5500  
40000 REM Istruzioni  
50000 REM Istruzioni presenti
```



```

000000 PRINT "Istr.:"
000001 PRINT P 1,RND*60-30
000002
000003 REM Inizializzazione
000004 REM Inizializzazione presen
000005
000006 PRINT "Iniz.:"
000007 PRINT P 1,RND*60-30
000008
000009 GO TO 20
000010 REM Partita finita
000011
000012 PRINT AT 9,0;" Vuoi le istr
000013 (s/n)?";
000014 GO TO 5510-(510 AND INKEY#=
000015 )-(510 AND INKEY#="n")

```

Una particolarita' e' la sua funzione ripetitiva. Nel caso di situazioni tipo "Altra partita (s/n)?", dove "s" e "n" stanno per si o per no, INKEY\$ viene usata come risposta. Esempio

```

5550 PRINT AT 7,0;"Vuoi un'altra
5551 partita (s/n)?";
5552 GO TO 5550+(30 AND INKEY#="
5553 s")-(30 AND INKEY#="n")
5554 PRINT AT 9,2;"Spero tu ti s
5555 divertito.";AT 11,9;"Ciao per
5556 sempre"
5557 STOP
5558 REM Resto del programma....

```

E' possibile adottare una tecnica simile anche per le istruzioni :

```

000000 REM Istruzioni
000001
000002 REM Inizializzazione
000003 REM (come prima):
000004
000005 ...
000006 ...
000007 REM Resto del programma....
000008 PRINT AT 9,0;"Vuoi le istru
000009 (s/n)?";
000010 GO TO 5510-(510 AND INKEY#=
000011 s)-(510 AND INKEY#="n")

```

Sfortunatamente, qui' puo' succedere che, una volta data la risposta, ad esempio "s", non si tolga abbastanza velocemente il dito dal tasto fornendo una analoga risposta, magari

errata, anche ad eventuali domande che seguano. Per evitare questo contrattempo e' sufficiente inserire la linea che segue
5590 IF INKEY\$<>" THEN GOTO 5590.

Esaminiamo ora dettagliatamente la funzione IN.

Innanzitutto IN non rende una stringa bensì un numero e poi non esplora in un sol colpo tutta la tastiera ma prende in considerazione blocchi di cinque tasti alla volta. Ogni singolo tasto sottrae un numero funzione della sua posizione nella fila dei cinque, da un secondo numero fisso caratteristico della tastiera quando non risulta azionato alcun tasto.

Prendiamo come esempio la fila coi numeri dall'1 al 5. La locazione di IN e' 63486, mentre il numero caratteristico e', di solito, 255.

Il tasto "1" sottrae 2 elevato a $0=1$; per cui, nel caso sia questo l'unico tasto azionato, il numero reso sarà $255-1=254$. Il "2" sottrae 2 elevato a $1=2$; per cui da solo rende $255-2=253$. Similmente il "3" sottrae 2 elevato a $2=4$; il "4" sottrae 2 elevato alla $3=8$ e "5" sottrae 2 elevato alla $4=16$.

Nel caso in cui ne venga premuto più di uno, si avranno le varie combinazioni. In presenza di una azione simultanea su tutti e cinque i tasti, IN 63486 renderà:

$255-16-8-4-2-1=244$.

Eccovi la serie completa delle locazioni cui si riferisce la funzione IN:

| | |
|----------|--|
| IN 63486 | : tasti 1,2,3,4,5 |
| IN 64510 | : tasti Q,W,E,R,T |
| IN 65022 | : tasti A,S,D,F,G |
| IN 65278 | : tasti Caps Shift,Z,X,C,V |
| IN 61438 | : tasti 6,7,8,9,0 |
| IN 57342 | : tasti Y,U,I,O,P |
| IN 49150 | : tasti H,J,K,L,ENTER |
| IN 32766 | : tasti B,N,M,Symbol Shift,Break/Space |

I primi quattro blocchi, si trovano nella metà sinistra della tastiera ed i relativi valori vanno considerati in ordine di importanza man mano che si va verso l'interno, per cui i tasti verso il bordo hanno valore 1 e quelli posti al centro 16. Per i rimanenti quattro blocchi, il discorso, ovviamente, si inverte.

Per essere più chiari, la cifra 1 viene sottratta alla pressione dei tasti "1","Q","p" e "0" come invece la 16 riguarda "G","5","6" e "B".

Un handicap di IN sta' nel fatto che il numero caratteristico non sempre e' 255, per cui possono venir resi anche numeri errati.

Un metodo attendibile per riportare ogni volta il valore caratteristico a 255, e' quello di inserire un BEEP prima della lettura della tastiera.

In linea di massima, e' più semplice usare INKEY\$ anche se in alcuni casi particolari, come appunto l'azionamento simultaneo di due tasti, la funzione IN si dimostra indispensabile.

Per verificare la versatilità di IN, provate a far girare il programma che segue riguardante un semplice "Sketchpad" che vi

abiliterà a disegnare sullo schermo. Se possedete una stampante, potrete in ogni momento dare un BREAK al programma e ricavare una copia del quadro tramite il comando COPY.

```

10000 GO SUB 10000
10010 BEEP .1,10
10020 IF E=1 THEN OVER 1
10030 PLOT INK INK;X,Y: LET a=X:
10040 DRAW SUC L'is a s u c 0 variare
10050 DRAW 1007 in 10051, 1009 in 1005,
10060 DRAW 1007 in 10051, 1009 in 1005,
10070 LET X=X+(IN 10054, 1007 IN 1005,
10080 LET Y=Y+(IN 10054, 1007 IN 1005,
10090 PLOT INK INK;X,Y: INK INK
10100 LET a=X:
10110 IF INKEY#="" THEN
10120 IF INKEY#="7" THEN
10130 PRINT AT 0,5
10140 OVER 0: GO TO 30
10150 LET X=127: LET Y=87: LET in
10160 BORDER 0: PAPER 7: INK 0: C
10170 PRINT "Movimenti: " " U-
10180 PRINT " " " D-dest
10190 PRINT " " " Combinazioni diago
10200 PRINT "Premi un tasto."
10210 IF INKEY#="" THEN GO TO 10
10220 IF INKEY#=" " THEN GO TO 106
10230 CLS
10240 PRINT FLASH 1;"DIS. "; FLASH
10250 PLOT 0,127: DRAW 191,0
10260 RETURN

```

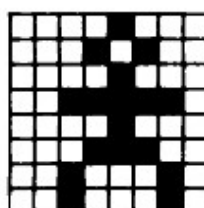
DIS. MUOV. CANG.

J C E

GRAFICA DEFINITA DALL' UTENTE

Una delle peculiarita' dello Spectrum e' la possibilita' di creare facilmente nuovi caratteri. Ogni carattere, disposto in una griglia 8*8 pixel, e' formato dalla combinazione di 64 punti ciascuno dei quali puo' essere "pieno" o "vuoto" (settato o resettato) a seconda che debba essere visibile o meno.

Considerate l' omino che segue :



Come si puo' notare, e' composto in un array (matrice) di 8*8 punti. Il primo problema da risolvere e' come inserirlo nel computer e visto che questo accetta solo numeri, vediamo di eseguire la trasformazione necessaria.

Il carattere viene definito da 8 numeri : uno riferito alla prima riga, un secondo riferito alla seconda riga e cosi' via. Il comando interessato e' BIN, forma tronca che sta per "binary", accompagnato da un numero binario formato da tanti zeri quanti sono gli spazi e da tanti 1 quanti sono i punti. Gli otto numeri dell' omino di cui sopra sono quindi :

```
I    riga - BIN 00001000
II   riga - BIN 00010100
III  riga - BIN 00001000
IV   riga - BIN 00111110
V    riga - BIN 00001000
VI   riga - BIN 00011100
VII  riga - BIN 00100010
VIII riga - BIN 00100010
```

Il manuale riporta che BIN 00000000 puo' venir sostituito da 0 il quale e', evidentemente il suo equivalente decimale, ma non dice che anche qualsiasi altro numero binario ha un suo sostituto in decimale, per cui, se preferite operare in decimale, eseguite ogni volta la conversione b/d. Se andate di fretta, non avrete da far altro che battere il comando diretto PRINT BIN seguito dal numero binario per ricevere all'istante dal computer l'equivalente decimale. Se invece avete un attimo di tempo, soffermatevi a leggere quanto segue sul come ricavarli a mano.

Considerate il decimale 3287, lo possiamo scrivere anche come sottonriportato tenendo presente che il segno "↑" sta' per "elevato alla potenza di" :

```
3*10↑3 (1000) = 3000
2*10↑2 (100)  =  200
8*10↑1 (10)   =   80
7*10↑0 (1)    =    7
```

3287

E' possibile cioe' scinderlo, moltiplicandolo termine per termine per una potenza di dieci in virtu' del fatto che tale e' la base dei numeri decimali.

Lo stesso discorso vale anche per i numeri binari la cui base e' ovviamente 2. Ad esempio :

```
1*2↑7 (128) = 128
0*2↑6 (64)  =   0
0*2↑5 (32)  =   0
1*2↑4 (16)  =  16
1*2↑3 (8)   =   8
1*2↑2 (4)   =   4
0*2↑1 (2)   =   0
1*2↑0 (1)   =   1
```

157

Altro modo per ottenere la conversione, e' quello di consultare la tabella sotto riportata. I numeri binari con gli 1 nelle posizioni d' inizio sono quelli di valore piu' alto e li troverete verso la fine della lista

```
10 FOR i=0 TO 255: LET j=i
20 LET b$="00000000"
30 FOR n=7 TO 0 STEP -1: IF 2↑
n <= j THEN LET b$(8-n)="1": LET j
=j-2↑n
40 NEXT n: PRINT b$;" --- ";i
50 NEXT i
```

| | | |
|----------|-----|----|
| 00000000 | --- | 0 |
| 00000001 | --- | 1 |
| 00000010 | --- | 2 |
| 00000011 | --- | 3 |
| 00000100 | --- | 4 |
| 00000101 | --- | 5 |
| 00000110 | --- | 6 |
| 00000111 | --- | 7 |
| 00001000 | --- | 8 |
| 00001001 | --- | 9 |
| 00001010 | --- | 10 |
| 00001011 | --- | 11 |
| 00001100 | --- | 12 |

La grafica viene generata con un loop in cui il numero di ogni riga viene letto (READ) da uno statement di DATA. La routine usata piu' frequentemente e'

```

0000 REM FORM TORE 9100
0000 REM FROM 0 TO 7
0000 REM NEXT I
0000 REM USR "a"+n,a
0000 REM CURZ
0000 DATA 8,20,8,52,8,28,34,34

```

In questo specifico caso si definisce "A" con la sagoma dell'omino di prima.

Volendo definire piu' di un carattere, e' necessario apportare una modifica al programma aggiungendo le linee che seguono e inserendo i dati dalla linea 9010 in avanti. I caratteri vanno definiti dalle lettere UDG che vanno dalla "A" alla "U".

```

9010 FOR i=1 TO n: REM ['n' e' il numero dei
      caratteri da definire]
      .
      .
9040 POKE USR CHR$(i+143)+n,a
      .
9060 NEXT i

```

La subroutine usata d' ora in avanti da questo libro e'

```

0000 FOR i=1 TO n
0000 FOR n=0 TO 7
0000 REM USR CHR$ (i+143)+n,a
0000 REM NEXT I
0000 REM CURZ
0000 DATA n1,n2,n3,n4,n5,n6,n7,n
0000 DATA n1,n2,n3,n4,n5,n6,n7,n

```

Per rendere piu' veloce e piu' agevole la vostra programmazione, ecco una lista delle sagome grafiche usate piu' correntemente con i relativi numeri di DATA.

Per primi, gli invaders "statici", che non devono girarsi, saltare o camminare ma solo fare scena.

DATA 129,90,60,90,60,102,153,195

DATA 129,66,60,219,165,24,36,195

DATA 195,165,24,60,90,36,24,231

DATA 129,102,60,255,90,255,129,231

DATA 60,126,219,219,255,126,66,102

Per gli invaders in movimento scrivere prima una versione e quindi l'altra presentandole alternativamente :

DATA 60,126,235,235,255,126,36,231

DATA 60,126,215,215,255,126,66,102

DATA 66,36,189,173,255,60,36,231

DATA 36,36,60,247,189,189,36,36

DATA 129,189,189,173,239,60,36,231

DATA 231,165,189,181,52,255,129,231

DATA 0,24,60,90,126,24,36,90

DATA 60,90,126,60,24,36,90,0

DATA 198,34,20,42,28,34,17,230

DATA 99,68,40,84,56,68,136,103

Ora gli "alieni", ben conosciuti dagli appassionati di giochi spaziali. Con questo sistema l'effetto si crea presentando un carattere alternato ad altri due :

DATA 60,126,255,170,170,255,126,60

A sinistra :

DATA 3,7,15,10,10,15,7,3

A destra :

DATA 192,224,240,160,160,240,224,192

Altro metodo, e' quello di stampare gli alieni (tutti lunghi non piu' di due caratteri) di seguito nell'ordine. Si ottiene un effetto assai realistico specialmente a velocita' sostenute.

Primo "alieno"

Sinistra : DATA 63,127,255,73,73,255,127,63
Destra : DATA 252,254,255,36,36,255,254,252

Secondo "alieno":

sinistra : DATA 63,127,255,36,36,255,127,63
Destra : DATA 252,254,255,146,146,255,254,252

Terzo "alieno"

Sinistra DATA 63,127,255,146,146,255,127,63
Destra DATA 252,254,255,73,73,255,254,252

Basi missilistiche

DATA 24,24,24,60,126,255,219,219

DATA 0,24,24,24,255,255,255,255

DATA 24,24,24,60,255,255,255,0

Sinistra DATA 1,1,7,9,31,127,255,255
Destra DATA 0,0,192,32,240,252,254,254

Vediamo qualche astronave

Sinistra : DATA 0,252,32,33,18,127,15,1
Destra : DATA 0,0,240,8,4,228,255,248

Sinistra : DATA 31,32,79,255,255,79,32,31
Destra : DATA 0,0,60,235,235,60,0,0

Sinistra : DATA 0,248,32,16,15,63,15,3
Destra : DATA 0,0,0,224,16,8,207,248

Alcuni asteroidi assomigliano ad astronavi. Sono formati da due disegni uno per la sagoma orizzontale-verticale, l' altro per quella diagonale.

Rivolto verso nord-ovest :

DATA 192,176,76,67,44,40,16,16

Rivolto verso ovest :

DATA 0,7,25,98,132,98,25,7,

Rivolto verso sud-ovest :

DATA 16,16,40,44,67,76,176,192

Rivolto verso sud :

DATA 198,170,146,68,68,40,40,16

Rivolto verso sud-est :

DATA 8,8,12,44,194,50,13,3

Rivolto a est :
DATA 0,224,152,70,33,70,152,224
Rivolto a nord-est :
DATA 3,13,50,194,44,12,8,8
Rivolto a nord :
DATA 16,40,40,68,68,146,170,198

I prossimi grafici sono aerei.

Un tipo di Jet
Sinistra : DATA 0,0,224,112,120,255,60,15
Destra : DATA 0,0,0,0,224,248,30,248

Un modello normale :
Sinistra : DATA 192,192,227,158,127,0,0,0
Destra : DATA 0,228,244,60,244,36,80,32

Un secondo tipo di Jet
Sinistra : DATA 14,199,224,255,127,1,0,0
Destra : DATA 0,0,96,248,252,224,240,112

Un modello particolare quale usato nel programma "Olocausto".

Sinistra : DATA 192,224,240,248,143,255,15,1
Centrale : DATA 0,7,9,17,255,255,224,255
Destra : DATA 0,192,32,16,254,255,30,248

Le prossime otto file di dati si riferiscono ad un aereo inclinato in ognuna dello otto direzioni della rosa dei venti.

Verso nord :
DATA 8,8,28,62,127,8,8,28

Verso nord-est :
DATA 1,2,124,60,28,172,68,32

Verso est :
DATA 16,24,156,255,156,24,16,0

Verso sud-est :
DATA 32,68,172,28,60,124,2,1

Verso sud :
DATA 28,8,8,127,62,28,8,8

Verso sud-ovest :
DATA 4,34,53,56,60,62,64,128

Verso nord-ovest :
DATA 128,64,62,60,56,53,34,4

Se preferite stare con i piedi per terra, eccovi una sequenza di carri armati e di navi.

Sinistra DATA 0,16,31,31,63,127,63,0
Destra DATA 0,0,252,0,240,248,240,0

Sinistra DATA 0,0,63,0,15,31,15,0
Destra DATA 0,8,248,248,252,254,252,0

DATA 0,0,64,63,112,126,255,126
DATA 0,0,20,252,14,126,255,126

Nave da guerra rivolta a sinistra

Sinistra DATA 0,0,2,115,23,255,126,63
Destra DATA 192,192,224,239,236,255,255,254

ora rivolta a destra

Sinistra DATA 3,3,7,247,55,255,255,127
Destra DATA 0,0,64,204,232,255,254,252

Ecco ora i dati per ottenere un carro armato direzionato in vari modi.

Verso nord :
DATA 8,73,73,93,127,93,65,65

Verso nord-est :
DATA 33,66,188,60,60,29,2,4

Verso est :
DATA 254,16,56,63,56,16,254,0

Verso sud-est :
DATA 4,2,29,60,60,188,66,33

Verso sud :
DATA 65,65,93,127,93,73,73,8

Verso sud-ovest :
DATA 32,64,184,60,60,61,66,132

Verso ovest :
DATA 127,8,28,252,28,8,127,0

Verso nord-ovest :
DATA 132,66,61,60,60,184,64,32

Concludendo la selezione dei semoventi terrestri ecco qualche auto :

In alto a sinistra : DATA 31,31,4,5,55,59,55,6
In alto a destra : DATA 248,248,32,160,236,220,236,96

In basso a sinistra: DATA 6,6,7,19,31,19,1,7
In basso a destra : DATA 96,96,224,200,248,200,128,224

Sinistra : DATA 224,159,119,239,239,119,159,224
Destra : DATA 28,242,190,191,191,190,242,28

DATA 0,66,189,239,239,189,66,0

DATA 102,153,239,223,223,239,153,102

Lanciamo le bombe

DATA 32,16,160,92,30,31,15,7

DATA 54,28,8,28,62,62,28,8

Qui' di seguito trovate alcuni "pacman" disegnati in modo che ognuno dei primi quattro si alterni col quinto in funzione della propria direzione.

Pacman rivolto a destra
DATA 60,127,252,240,240,252,127,60

Pacman rivolto in su'
DATA 66,66,231,231,255,255,126,60

Pacman rivolto a sinistra
DATA 60,254,63,15,15,63,254,60

Pacman rivolto in giu'
DATA 60,126,255,255,231,231,66,60

Pacman con la bocca chiusa
DATA 60,126,255,255,255,255,126,60

Spettro
DATA 56,124,214,214,254,254,170,170

Ora un po' di frutta

Fragola : DATA 24,82,247,255,255,126,60,24
Lampone : DATA 4,8,8,86,171,213,106,60
Ciliegia : DATA 8,8,20,20,34,99,243,96
Pesca : DATA 44,110,231,247,247,247,102,44
Banana : DATA 2,3,7,14,30,124,248,0
Prugna : DATA 8,16,24,60,124,62,60,24
Mela : DATA 24,82,255,255,255,255,126,36

Per concludere la parte della grafica definita dall'utente, presentiamo una serie alternativa di numeri riferiti a forme spesso usate in fasi particolari dei giochi TV.

Un esempio e' il punteggio, inserito in una variabile di stringa con STR\$ e quindi scelto per mezzo di un loop tra i vari caratteri, l'ultimo dei quali determinato da LEN. Il codice (CODE) di ciascun carattere viene determinato aggiungendo il numero corrispondente alla differenza tra il CODE del digit interessato e il CODE della grafica stessa definita dall'utente.

Ma poiche' gli esempi valgono da sempre piu' delle parole battete la routine che segue. La variabile "S" contiene il punteggio mentre gli user-graphic da 0 a 9 derivano dalle lettere da A e J impostate in "graphic mode". La variabile S# contiene il punteggio da ricavare dalla serie di numeri.

```
4000 LET S$=STR$ S
4010 FOR I=1 TO LEN S$
4020 LET S$(I)=CHR$ (CODE S$(I) +
95)
4030 NEXT I
4040 RETURN
```

Eccovi i grafici :

```
0 DATA 0,60,66,66,98,98,98,60
1 DATA 0,8,8,8,12,12,12,12,
2 DATA 0,124,2,2,60,96,96,62
3 DATA 0,124,2,2,60,6,6,126
4 DATA 0,64,96,100,100,126,4,4
5 DATA 0,62,64,64,60,6,6,126
6 DATA 0,62,64,64,124,70,70,126
7 DATA 0,120,8,8,12,12,12,12
8 DATA 0,60,66,66,60,70,70,60
9 DATA 0,60,66,66,60,6,6,126
```

Dopo aver esaminato alcuni possibili caratteri, vediamo il programma sotto riportato che vi permettera' di effettuare la definizione per conto vostro :

```
9500 PRINT TAB 7;"Generatore di
caratteri"
9510 PRINT "per creare grafici
definiti dall'utente su una matr
ice di 8x8"
```

```

1000  REM *****
1010  REM *****
1020  REM *****
1030  REM *****
1040  REM *****
1050  REM *****
1060  REM *****
1070  REM *****
1080  REM *****
1090  REM *****
1100  REM *****
1110  REM *****
1120  REM *****
1130  REM *****
1140  REM *****
1150  REM *****
1160  REM *****
1170  REM *****
1180  REM *****
1190  REM *****
1200  REM *****
1210  REM *****
1220  REM *****
1230  REM *****
1240  REM *****
1250  REM *****
1260  REM *****
1270  REM *****
1280  REM *****
1290  REM *****
1300  REM *****
1310  REM *****
1320  REM *****
1330  REM *****
1340  REM *****
1350  REM *****
1360  REM *****
1370  REM *****
1380  REM *****
1390  REM *****
1400  REM *****
1410  REM *****
1420  REM *****
1430  REM *****
1440  REM *****
1450  REM *****
1460  REM *****
1470  REM *****
1480  REM *****
1490  REM *****
1500  REM *****
1510  REM *****
1520  REM *****
1530  REM *****
1540  REM *****
1550  REM *****
1560  REM *****
1570  REM *****
1580  REM *****
1590  REM *****
1600  REM *****
1610  REM *****
1620  REM *****
1630  REM *****
1640  REM *****
1650  REM *****
1660  REM *****
1670  REM *****
1680  REM *****
1690  REM *****
1700  REM *****
1710  REM *****
1720  REM *****
1730  REM *****
1740  REM *****
1750  REM *****
1760  REM *****
1770  REM *****
1780  REM *****
1790  REM *****
1800  REM *****
1810  REM *****
1820  REM *****
1830  REM *****
1840  REM *****
1850  REM *****
1860  REM *****
1870  REM *****
1880  REM *****
1890  REM *****
1900  REM *****
1910  REM *****
1920  REM *****
1930  REM *****
1940  REM *****
1950  REM *****
1960  REM *****
1970  REM *****
1980  REM *****
1990  REM *****
2000  REM *****

```


| | |
|-----------|--|
| Note | |
| 10 | Inizio |
| 20 | Loop per aereo |
| 30 | Predisporre l' aereo successivo |
| 40-70 | Loop per muovere l' aereo |
| 80-100 | Aereo successivo |
| 170 | Fine della gara |
| 3000-3190 | Predisporre la corsa dell'aereo successivo |
| 4000-4020 | Disegna il suolo |
| 4030 | Posizione della casa |
| 4100-4190 | Disegna lo schermo |
| 4500-4510 | Sgancia la bomba |
| 4520 | Esplosione |
| 4530-4540 | Setta le variabili di "yes" e "bomb" |
| 5000-5490 | Inizializzazione |
| 9000-9030 | Grafica definita dall' utente |
| 9100-9260 | Dati per la grafica. |

SLONI

Per produrre i suoni lo Spectrum usa il comando BEEP. Il formato dello statement e' BEEP d,n in cui "d" e' il valore numerico della durata della nota e "n" la sua tonalita'. Il principale vantaggio di tale comando e' l'estrema facilita' d'uso. Pur essendo costante, la qualita' del suono, permette la composizione di effetti assai gradevoli. Il maggior svantaggio e' che, durante l'esecuzione dei suoni, risulta inibita ogni altra operazione con conseguente rallentamento dell'intero programma. Per tale motivo, il suo uso viene ristretto in listati relativamente lunghi.

BEEP puo' venire usato in concomitanza con gli statement di READ e DATA per comporre motivetti coi quali accompagnare i games.

Provate quelli che seguono

| | | | |
|--|---|-----------------------------------|--|
| | - | 60000001 | |
| | - | 80+000000 | |
| | - | 940001-0 | |
| | - | 00000000 | |
| | - | DZBX7X | |
| | - | CIMMCOM | |
| | - | +XXIXD | |
| | - | D-HO +C | |
| | - | . -ad=H | |
| | - | JN CCHE | |
| | . | r r w t e y | |
| | . | w t e , n o t e | |
| | . | , 1 , 2 , . 1 , 2 , . 2 , 4 , . 4 | |

E' altresì possibile ottenere ottimi risultati impiegando dei loop FOR...NEXT come nei due esempi qui' di seguito :

```

000000 ZONE 10 TO -10 STEP -.5
000000 TIME 0.000000
000000 XPR 0.000000
000000 YPR 0.000000

```

```

00000000
00000000
04001000
00000000
Z0T0Z0T
M000M00
X00X000
101010
0.1010
0100 TO -10 STEP -1.5
0100 TO 10 STEP .5

```

Se vi siete divertiti, battete allora anche questo noto
motivetto :

[illegible]

Volendo, potete anche dotare il vostro Spectrum di un box sonoro (ottimo quello della Tenkolek) per amplificare il segnale presente sulla presa di EAR. Una buona amplificazione si ottiene anche usando da buffer il registratore stesso. Per far cio', collegare un cavetto tra l'uscita EAR dello Spectrum e l'ingresso MIC del registratore dopodiche' dare il "play".

USO DEGLI OPERATORI LOGICI

Gli operatori logici sono :

AND;OR;NOT;=<>(diverso da);<;>;<=>=

Tali funzioni vengono completate dalle espressioni matematiche +,-,* (moltiplicato) e / (diviso). Gli operatori logici possono sostituire IF...THEN come nell'espressione IF A=D THEN LET C=1 equivalente a "A=B". Se l'espressione e' vera, cioe' se 'A' e' realmente uguale a 'B', l'azione prosegue attribuendo, nel nostro caso, a 'C' il valore 1. Analogamente agisce LET C=(A=B), il cui nucleo e' "(A=B)", le parentesi non sono necessarie all'atto pratico ma chiariscono maggiormente il concetto. Il computer valuta anche qui l'uguaglianza e se questa risulta vera, rende a C il valore 1, viceversa, cioe' nel caso in cui A sia diverso da B, rende 0. Facciamo un esempio, supponete che tanto A quanto B valgano 5 :

C=(A=B) (oppure C=A=B)

C=(5=5)

E' 5 uguale a 5 ? Si, per cui

C= VERO

C= 1

Si nota chiaramente come, l'espressione logica, operi alla stregua degli statement IF...THEN con la sola differenza che, questi mantengono il precedente valore della variabile e non la pongono a 0 come invece succede usando gli operatori logici se questa risulta falsa.

Il segno "=" non funge solamente da comparatore

IF A<>B THEN LET C=1

LET C=(A<>B)

IF A<B THEN LET C=1

LET C=(A<B)

IF A>B THEN LET C=1

LET C=(A>B)

IF A<=B THEN LET C=1

C=(A<=B)

IF A>=B THEN LET C=1

C=(A>=B)

Impiegando espressioni matematiche al posto degli operatori logici, ne deriva che tutte le grandezze diverse da 0, vengono considerate vere e tutte quelle uguali a 0, false. I segni matematici possono essere inseriti in statement di IF...THEN come nella linea : IF A=B THEN LET C=1. In questo caso l'espressione A=B viene considerata esclusivamente numero, che se diverso da 0, rende vera l'espressione permettendo l'esecuzione del comando che segue THEN nella fattispecie LET C=1. Può sorgere il dubbio che l'espressione A=B, così impiegata, equivalga a $A < B$, ma ciò non è assolutamente vero come capirete battendo il programmino che segue in cui le forme interessate sono evidenziate in carattere inverso per una maggior chiarezza :

```
10 LIST
20 LET A=15
30 LET B=5
40 IF A<B THEN LET C=1: GO SUB
B 100
50 IF A=B THEN LET C=1: GO SUB
100
60 LET C=(A<B): GO SUB 100
70 LET C=(A=B): GO SUB 100
80 STOP
100 PRINT "C="; C
110 RETURN
```

Il risultato è la dimostrazione che A=B possiede un valore puramente matematico e non logico.

Per esaminare i corrispondenti operatori logici dei segni matematici, dobbiamo prendere in considerazione le funzioni AND, OR e NOT. Anche queste possono trovare posto in statement IF...THEN:

IF A=5 AND B=6 THEN...

IF A OR B=2 THEN...

IF A<>0 OR B=2 THEN...

IF NOT A=B THEN...

IF A<>B THEN...

qui notiamo che la seconda e la terza linea riguardano la stessa funzione come pure la quarta e la quinta. Il loro impiego permette di inserire espressioni in statement relativi a "operazioni logiche". Nell'esempio, LET C=(A=3 AND B=2), il valore di C e' 1 solamente se A e' uguale a 3 e B e' uguale a 2. La funzione OR raggiunge lo stesso risultato con l'aiuto di IF...THEN. NOT invece, inverte il valore dell'espressione permettendo l'esecuzione sia nel caso di risultato vero (1), sia nel caso di risultato falso (0). In altre parole, trasforma gli "1" in "0" e gli "0" in "1".

Visto cio' e' ora possibile comprendere l'uso dei segni +, * e / come operatori logici. Se consideriamo le espressioni facenti parte di IF (espressione) THEN... , se ne deduce che :

"A-B" equivale a "A<>B"

"A+B" equivale a "A OR B". Notate pero' che se A vale 5 e B vale -5, il risultato della somma e' 0 anche se sono entrambi veri.

"A*B" equivale a "A AND B"

"A/B" equivale ad A stesso, a meno che A sia talmente piccolo rispetto a B che il risultato si approssimi a 0, rivelandosi in tal caso falso.

Gli esempi portati finora per A e B valgono naturalmente anche per le stringhe A\$ e B\$.

Come esempio, supponiamo ora di voler "simulare" l'espressione IF A=B THEN LET C=3 con operatori logici. La forma equivalente risultera' :

```
LET C=(X AND espressione)
```

dove "X" e' il numero (o la stringa) e "C" la grandezza a cui viene attribuito il valore "X" se l'espressione e' vera. Nel caso in cui risulti falsa, "C" varra' 0.

Impiegando l'espressione di cui sopra si otterra'

```
LET C=(3 AND A=B)
```

Anche la funzione OR puo' assumere una tale configurazione, naturalmente gli argomenti saranno diversi :

```
LET C=(X OR espressione)
```

"X" e' sempre solo un numero in quanto con OR non si possono usare stringhe. Il valore di C in questo caso risulta 1 con espressione vera e "X" con espressione falsa.

A dimostrazione di cio', fate girare il programma che segue mettendo in pratica quanto avete imparato dal capitolo e analizzando i risultati.

- Tutto torna alla normalita' dando priorit  all' esecuzione dell' OR per cui   necessario l' intervento di una seconda coppia di parentesi :

Gli operatori logici trovano spesso impiego nei programmi dei giochi in quanto :

- ### Programma 1: usando IF...THEN

[illegible][illegible]

- 37

```

      50 FOR i=1 TO 3: IF a$(i) <> b$
THEN PRINT AT b+i+i, a-(a AND a<0
); a$(m+i, 1-(a AND a<0) TO )

```

Volendo ottenere gli stessi risultati con gli statement di IF...THEN e' neccessario impiegare piu' linee :

```

      50 FOR i=1 TO 3: IF a$(i)=b$ T
HEN GO TO 60
      60 IF a<0 THEN PRINT AT b+i+i,
0; (b+i, 1-a TO ); GO TO 60
      64 PRINT AT b+i+i, a; a$(m+i)
      66 REM ....resto del programma
      ...

```

3) Provocano un minor assorbimento da parte del banco di memoria in quanto occupano meno spazio di IF...THEN come dimostrato dagli esempi di cui sopra.

Nei programmi racchiusi in questo volume si e' fatto un largo impiego di operatori logici cosi' come si puo' notare anche nel prossimo dal titolo "Bomber". Il game consiste nel bombardare una citta' per potervi creare spazio per l'atterraggio. Le istruzioni dettagliate, sono insite nel programma stesso.

Note

| | |
|-----------|---|
| 10 | Inizializza le variabili, scrive le istruzioni. |
| 20-90 | Loop ripetuto per muovere l' aereo. |
| 100-160 | Linee eseguite quando la bomba e' sganciata. |
| 170-180 | Reset delle variabili quando la bomba termina la caduta. |
| 60 | Disegno dell' aereo |
| 140 | Disegno della bomba |
| 500-540 | Presenta il nuovo punteggio e calcola le bombe lanciate. |
| 3000-3050 | Interludio tra due parti. |
| 4000 | Dati per il motivetto. |
| 5000-5060 | Fine del programma. |
| 8010-8080 | Definizione delle variabili. |
| 8110-8170 | Scrive le istruzioni, che volendo possono essere riscritte. |
| 8500-8590 | Predisporre lo scenario disegnando lo schermo. |
| 8560 | Mette otto grattacieli in S (29) |
| 9000-9360 | Definizione della grafica. |
| | A e B - aereo |
| | C e D - bomba |
| | E - grattacielo |
| | F - grattacielo piu' alto |
| | G - Resto del grattacielo bombardato. |

CRASH,
LAND,
SCREEN,
UPDATE,
LOOP

"Tokens" per le subroutines

Le lettere maiuscole alle linee 60,140,160,5000,8540 e 8560
vanno battute in "graphic mode" e servono a generare i
caratteri richiesti.

Gran parte dei games, per non dire tutti, posseggono oggetti in movimento che rappresentano figure spesso bizzarre come i "pacmen" e le navi spaziali. Le difficoltà piu' evidenti nello scrivere programmi nascono dal fatto che :

- a) il BASIC non e' sufficientemente veloce per muovere contemporaneamente diverse immagini,
- b) l' animazione e' a scatti; piu' a lungo una immagine si ferma sullo schermo e peggio e',
- c) gli oggetti non devono travalicare i confini imposti dallo schermo,
- d) le varie parti non possono "sapere" in che direzione spostarsi; ad esempio gli invasori spaziali si muovono solo a destra e a sinistra,
- e) il computer deve esplorare lo schermo per rilevare le eventuali collisioni di due oggetti (ad esempio, la racchetta con la palla).

Tutto ciò si verifica su uno scenario di sottofondo mantenuto fisso. L' oggetto viene stampato (PRINT) o disegnato (PLOT) in relazione ad una serie di coordinate x e y, quindi cancellati e ridisegnati con una o con tutte e due le coordinate cambiate di valore il che dona l' illusione del movimento.

L'effetto dello spostamento a scatti e' appunto dovuto al fenomeno di cancellazione-ristampa e la sua intensita' dipende dal tempo intercorrente tra le due azioni. Va da se' che piu' rapidamente la figura viene rappresentata dopo essere stata cancellata, piu' si ha la sensazione di un movimento costante. Il programma che segue, stampa un quadratino nero al centro e lo sposta fino alla fine dello schermo :

[illegible]

Provatelo e vedrete il quadratino lampeggiare scorrendo verso destra. Come si può migliorare un tale effetto? Un rimedio è quello di inserire il calcolo di avanzamento, di cui alla linea 40, tra il disegno e la cancellazione dell'oggetto il quale, pertanto, si ferma sullo schermo il tempo necessario al calcolo delle sue nuove coordinate.

Il risultato si ha col programma che segue in cui e' stata prevista una temporanea memorizzazione (linea 20) del vecchio valore di "4" per non far stampare erroneamente la linea 30 :

```

1000 LET Y=0
2000 FOR Y=0 TO 31
3000 PRINT AT 10,Y;"■"
4000 Y=Y+1
5000 NEXT Y

```

Nonostante questo accorgimento, il lampeggiare dell' oggetto e' ancora percettibile. Provate allora ad aggiungere una linea che prolunghi la permanenza dell' immagine per mezzo di un loop FOR....NEXT oppure di un bit musicale :

```

45 FOR N=1 TO 5:NEXT N
45·BEEP .01,RND*20

```

Il programma cosi' scritto conduce pero' ad un errore in quanto porta il computer a scrivere dove non gli e' possibile. Fatelo girare e rilevate il messaggio che appare quando si blocca. Per ottenere un effetto analogo potete impiegare il loop FOR....NEXT come segue :

```

1000 FOR Y=0 TO 31
2000 PRINT AT 10,Y;"■"
3000 BEEP .01,RND*20
4000 PRINT AT 10,Y;" "
5000 NEXT Y

```

Se lo spostamento del quadrato avviene di una colonna alla volta in linea retta, si impiega di solito uno statement di PRINT che cancelli la figura disegnata dal primo. Notate che l' ultimo quadretto non viene cancellato :

```

100 FOR Y=0 TO 30
200 PRINT AT 10,Y;"■"
300 BEEP .01,RND*20
400 NEXT Y

```

Gli "Asteroidi" e i giochi Arcade rimediano ai limiti di spazio imposti dai bordi facendo girare attorno gli oggetti, per meglio capire, una astronave che esca dal lato destro dello schermo, rientra immediatamente da quello sinistro progredendo nella sua corsa. Sullo Spectrum, questa funzione si implementa facilmente come dimostrato nel programma sotto riportato. La grandezza "y" che e' il numero della colonna nella quale e' disegnato il quadretto, una volta giunto a 32 torna a 0 ripresentando da capo il disegno. Per ottenere un egual risultato in verticale, andra' interessata "x" con valori da 0 a 22.

U
N
I
T
E
D
S
T
A
T
E
S
O
F
A
M
E
R
I
C
A

Con cio' affrontiamo l' argomento circa la "conoscenza" del percorso da parte degli oggetti. Il prossimo programma mostra un invader, nella fattispecie la lettera "V", che si sposta prima verso destra, poi verso sinistra scendendo in continuazione verso il basso :

SECRET

La variabile "direzione" contiene il numero da aggiungere (o da sottrarre se e' negativo) a "y". Alla linea 70 il programma testa se l' invader ha raggiunto uno dei margini dello schermo, se si, la variabile "direzione" viene invertita di segno. La stessa linea incrementa "x" facendo scendere l' invader verso il basso fino a raggiungere l' ultima riga del quadro alla quale il programma si ferma.

Prima di addentrarci nella "conoscenza" del senso di marcia da parte degli oggetti, provate i due programmini che seguono i quali fanno "rimbalzare" l' oggetto stesso sui bordi dello schermo per mezzo di PRINT negativi :

'Rimbazzo' sull'asse u

4001-0000

'Rimbanzo' sull' asse x

0000
1000

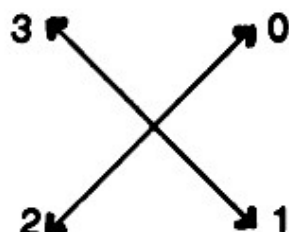
Queste proprietà vengono usate in molti giochi tra cui lo "squash" e' quello che le evidenzia maggiormente. Il nome della variabile usata per la direzione e' di solito "d" oppure "dir" e, nell'esempio dell'invader a "V", assume due valori ben precisi :

- a) 1, nel qual caso l' incremento vale 1 e l' oggetto si sposta verso destra,
- b) -1, in cui si ha un decremento con l' oggetto che si sposta verso sinistra.

Sempre nello stesso esempio, il valore del "marker" di direzione viene inserito direttamente nel calcolo :

```
LET Y=Y+direzione
```

e una tale forma puo' essere usata in qualsiasi momento per indicare dove l' oggetto deve prendere posto ma non per calcolare il movimento. Il valore del "marker" e' compreso tra 0 e 3 ed i numeri singoli si riferiscono alle direzioni :



E' molto difficile manipolare detti numeri per inserirli direttamente nei calcoli per il movimento, pero' e' possibile ricorrere all' aiuto degli operatori logici visti un capitolo addietro. Guardate :

'x' e' la coordinata x (\downarrow)

'y' e' la coordinata y (\rightarrow)

'd' e' il puntatore di direzione con i valori corrispondenti al diagramma.

Cosa succede a 'x' e 'y' per diversi valori.

```
1000 LET x=x+(d=1 OR d=2)-(d=0 OR d=3)
```

```
1010 LET y=y+(d<2)-(d>1)
```

Se un oggetto, nel suo spostamento urta una linea orizzontale (vedremo come piu' avanti analizzando SCREEN\$ e ATTR), puo', sempre usando gli operatori logici, rimbalzare cambiando direzione :

```

1999 REM Urto della linea orizzo
ntale
2000 LET d=d+1-(2 AND (d=3 OR d=
1))

```

Se non siete del tutto convinti che una sola linea possa cambiare correttamente i valori di "d", battete quanto segue e confrontatelo col diagramma :

```

? 10 INPUT "Il valore di 'd' e'
1000 PRINT "Vecchio="; d
2000 INPUT "Nuovo="; d
3000 LET d=d+1-(2 AND (d=3 OR d=1))
4000 GOTO 1000
5000 RETURN

```

Un altro metodo, si presenta con l'adozione delle stringhe di variabili e di VAL. Per vedere l'effetto di dette funzioni, battete questo programma dimostrativo :

```

100 INPUT "Come ti chiami ?"; a$
200 IF LEN(a$)>16 THEN PRINT AT
300 "16", "Troppo lungo, per favore re
400 stringa"; a$; a$=LEFT(a$,16); GOTO 100
500 LET z$="Z"; LET z$=z$+1; PRINT a$; z$;
600 IF LEN(z$)=1 THEN PRINT "Z"; GOTO 100
700 LET z$=VAL(z$); LET z$=z$+1; PRINT a$; z$;
800 IF z$=0 THEN LET z$="Z+1"; GOTO 40
900 GO TO 40

```

La collisione di due oggetti, come ad esempio un missile ed un invasore spaziale, viene rilevata facilmente testando la variabile dei due oggetti stessi : IF X missile=X invader AND Y missile=Y invader THEN...

Ciononostante, la via piu' facile, resta quella di esplorare direttamente lo schermo per mezzo di uno dei tre sistemi sottoelencati (anche se il terzo lo sconsigliamo) :

1) SCREEN#. Questa funzione rende una stringa e va inserita con la forma "SCREEN# (x,y)" dove "x" e "y" rappresentano le coordinate cosi' come per lo statement PRINT AT. Non lavora con gli UDG, ma solo con i caratteri che ritornano normali anche nel caso in cui siano inversi.

2) ATTR. Rende un numero che dipende dagli "Attributi" della cella definita dalle coordinate. Gli attributi sono il colore, il flashing e il brightness. Il relativo statement e' "ATTR (x,y)" della medesima fisionomia del precedente.

ATTR e' la somma di :

128 (10000000 binario) se il carattere lampeggia con FLASH 1,
64 (01000000 binario) se il carattere ha luminosita' piu'
accentuata con BRIGHT 1,

8 il colore di PAPER che occupa il terzo, il quarto e il
quinto posto dell' equivalente binario.

ATTR e' molto usato, in quanto non distingue se il carattere
che sta esaminando e' un UDG o meno.

3) POINT. Rende 1 oppure 0. La forma del suo statement e'
classica "POINT (x,y) in cui "x" e "y" vengono impiegati come
in PLOT x,y. Il risultato della funzione e' 1 se le coordinate
'pescano' un punto, mentre e' 0 se riscontrano un vuoto.

Vediamo ora la velocita'. Deve essere tale da mettere in
condizione il giocatore di poter sparare agli invaders che
scendono spostandosi casualmente a destra o a sinistra. Sia la
base che gli alieni, possono uscire dai margini laterali dello
schermo per riapparire dal lato opposto nella stessa posizione.
Gli invaders, quattro in tutto, vengono presentati dal computer
uno per volta rendendo la gara assai interessante anche perche'
a loro volta possono sparare alla base.

Le coordinate di ogni invader, sono memorizzate in due arrays,
l' array "a" per le coordinate verticali, il "b" per le
coordinate orizzontali. I suoi movimenti sono funzione di
quelli della base e vengono determinati dalla variabile "i"
presente nella linea 20. Fatta questa doverosa presentazione
eccovi il programma con relativa documentazione.

Elenco delle variabili

| | |
|--------|---|
| h | Record |
| a\$(.) | Array di caratteri contenente la forma degli alieni. |
| e\$(.) | Array contenente l' immagine dell' esplosione. |
| s | Punti. |
| men | Numero delle basi rimaste. |
| a(.), | |
| b(.) | Coordinate x e y degli invasori. |
| p | Numero della colonna della base. |
| i | Usato nel loop principale per il movimento degli alieni. |
| il | Usato per calcolare "i". |
| t1 | Numero di pixel della base nello sparo. |
| dr | Sviluppa la traiettoria in salita dello sparo della base. |
| | Usato per calcolare se l' invader e' stato colpito. |
| t2, | |
| t1 | Numero di pixel delle coordinate x e y quando spara l' alieno. |
| dr | Sviluppa la traiettoria in discesa dello sparo degli alieni. |
| x, | |
| i, | |

n Variabili per il controllo dei loop.
 m,
 c\$ Usate per la fine della gara.

Le lettere maiuscole nelle linee 140,1010,1020,1040,1510,1520, 7710,7720 riferiti agli statement di PRINT sono tutti caratteri definiti dall'utente(UDG)

Note

| | |
|-----------|---|
| 10 | Inizializza le variabili e scrive le istruzioni. |
| 20-100 | Parte del loop per muovere gli alieni e per sparare. |
| 105-160 | Parte del loop per muovere la base e per sparare |
| 1000-1060 | Calcolo del fuoco |
| 1400-1420 | Fuoco |
| 1500-1540 | Calcolo del fuoco |
| 1900-1920 | Fuoco |
| 2000 | Incrementa il punteggio |
| 3000-3020 | Esplosione della base |
| 4000-4070 | Fine di una parte |
| 4080-4180 | Fine della gara |
| 7000-7100 | Scrivo le istruzioni |
| 7500-7530 | Inizializza le variabili permanenti come ad esempio il punteggio. |
| 7600-7800 | Inizializza le altre variabili e disegna lo schermo |
| 9000-9030 | Definisce i caratteri UDG |
| 9050-9150 | Dati per gli UDG |
| 9500 | Dati per la fine |
| 9990-9999 | Fine. |

```

10 GO TO 7000: REM istr/iniz
20 LET i1=i: LET i=i+1-(4 AND
i1)+3
30 IF a(i)>=2 THEN PRINT AT a(
i),b(i);" "
40 LET a(i)=a(i)+1: LET b(i)=b
(i)-1+INT (RND*3)
50 IF b(i)=32 THEN LET b(i)=0
60 IF b(i)=-1 THEN LET b(i)=31
70 IF a(i)=17 OR a(i)=18 THEN
PRINT AT a(i),0;" "
80 IF a(i)=20 THEN GO SUB 3000
: REM stirato
90 IF a(i)>=2 THEN PRINT AT a(
i),b(i);a$(i)
100 IF a(i)>=2 AND RND>.9 THEN
GO SUB 1500: REM sparo
105 IF m=0 THEN GO TO 4000: R
EM fine della gara
110 PRINT AT 20,p;" "; LET p=p+
(INKEY$="2")-(INKEY$="1")

```



```

9500 DATA "U I T T O R I A", "GLI
INVASORI HANNO VINTO", "LA TERRA
E' MORTA", "SALVE IMPERATORE", "T
ROPPO FACILE"
9600 PAPER 7: BORDER 7: INK 0: C
9700 PRINT AT 10,12: FLASH 1: "CI
9800"
9900 STOP

```

Il programma che segue e' il classico "Space Invaders" con tanto di laser surriscaldabile e di invasori danzanti. Vengono usate delle stringhe in quanto gli alieni, disposti su tre file vanno colpiti uno alla volta. Le file, memorizzate in arrays, vengono duplicate onde permettere il movimento tipico che caratterizza i giochi riguardanti gli "Space Invaders".

Elenco delle variabili

| | |
|---------|--|
| b | Altezza degli invaders |
| h | Altezza di partenza degli invaders |
| m | Usato per l' alternarsi degli invaders |
| i | Loop di controllo per stampare gli invaders |
| a | Numero della colonna del primo invader |
| s, | |
| si | Usati per variare la velocita' della base in rapporto agli invaders |
| p | Numero di colonna della base |
| in | Usato per leggere la tastiera |
| k | "Marker" per rilevare se il laser e' surriscaldato o no |
| c | Numero degli invaders colpiti |
| l | Livello di temperatura del laser |
| d | Variabile per la direzione degli invaders |
| e | Livello di terra per vedere se gli invaders hanno toccato il fondo |
| men | Numero di basi rimaste |
| wave | Numero delle alternanze |
| v, | |
| x | Usati per i dati del motivetto |
| sc | Punti |
| i | Distanza della traiettoria degli spari |
| r | Numero di pixel della base |
| t | Usato per controllare la stringa e per vedere se ogni invader e' stato colpito |
| z | Usato assieme a "t" |
| a\$(.) | Arrays per la memorizzazione degli invaders |
| b\$ | Stringa vuota per controllare la comparazione ad a\$ di una riga di invader |
| n | Tono della nota suonata |
| clr | Usato per cancellare la tastiera |
| FIRE, | |
| HIT, | |
| SCREEN, | |
| SCORE | Memorizzano per chiarezza i numeri di partenza delle subroutines |

90,5160 e 5170

Note

| | |
|-----------|--|
| 10 | Inizializzazione e istruzioni |
| 30-60 | Muove gli invaders |
| 70-120 | Movimento della base e sparo |
| 130-160 | Laser |
| 170 | Rileva se gli invader sono alla fine della linea |
| 180-250 | Inizializzazione per una nuova ondata e una nuova base se gli invader sono atterrati |
| 800-830 | Suona il motivetto |
| 900-930 | Usato per la fine della partita |
| 2000-2050 | Prepara lo sparo |
| 2060-2080 | Laser |
| 2100-2200 | Fuoco e esplosioni |
| 3000-3060 | Considera quando l' invader risulta colpito |
| 4000 | Presenta il punteggio |
| 5010-5030 | Considera le variabili permanenti come i punti |
| 5150-5280 | Considera le altre variabili |
| 5300 | Chiede se lo schermo e' disegnato |
| 6000-6100 | Disegna lo schermo |
| 8000-8010 | Dati per il motivo |
| 9000-9030 | Definisce gli UDG |
| 9100-9160 | Dati per UDG |

[illegible]

v,
i,
l Variabili di controllo per i loops ecc.

I caratteri grafici da definire battendo le maiuscole in "graphic mode", trovano posto nelle linee :
300,"B",310,"U","D","L","R",330,"Q",510,"A",1200,"C",5000,"Q".

Note

| | |
|-----------|---|
| 10 | Inizializzazione e istruzioni |
| 20-50 | Tastiera |
| 60-70 | Testa per vedere se qualcosa e' stato colpito |
| 90 | Tratta il limite di tempo |
| 100-110 | Tratta la coda |
| 120 | Inserisce i valori nell' array |
| 300-310 | Subroutine per il disegno della testa |
| 500-550 | Disegna una nuova mela, il punteggio e suona il motivetto |
| 1000-1010 | Predisporre i record |
| 1020-1050 | Istruzioni |
| 1060-1080 | Istruzioni circa i limiti di tempo |

Nel caso possediate uno Spectrum a 16K, dovete apportare le modifiche che seguono :

Linee 20,30,110,520,5020,1170,1180. Cambiare ogni numero 500 in 180 e ogni 499 in 179. Se non fate cio' il programma impegna troppa memoria e si blocca con l' errore 4 "Out of memory".

| | |
|-----------|---|
| 1090-1150 | Selezione dei livelli di abilita' |
| 1160 | Colori, che potete anche variare |
| 1170-1190 | Inizializzazione |
| 1200-1220 | Disegna lo schermo |
| 4000-4020 | Definisce gli UDG |
| 4030-4100 | Dati per gli UDG |
| 5000-5050 | Non disegna la coda alla fine della gara |
| 5100-5110 | Punti |
| 5120 | Presenta il record precedente se non e' stato battuto |
| 5130-5240 | Presenta il nuovo record nel caso sia stato ottenuto |
| 5250-5320 | Presenta la tavola dei record e il "bit per la fine" |
| 9000 | Dati per il motivetto |

Serpenti

```

10 GO SUB 4000: GO TO 1000
20 LET h=h-1: IF h=0 THEN LET
h=5000
30 LET j=j-1: IF j=0 THEN LET
j=5000
40 LET a$=INKEY$: IF a$>"4" AN
D a$<"9" THEN LET d$=a$

```


due "variabili di direzione", una per le coordinate "x" (variabile dx) e una per le coordinate "y" (variabile dy), le quali contengono un numero che puo' essere sia positivo che negativo dalla cui grandezza dipende la velocita' con la quale la palla viaggia nelle varie direzioni.

Se per voi il ritmo della gara e' troppo elevato, variate il contenuto delle linee che calcolano "dx" e "dy". La palla va colpita con la racchetta ogni qualvolta venga respinta dal muro. Il movimento della racchetta si ottiene con i tasti 5 e 8 in accordo con le relative frecce direzionali.

Elenco delle variabili

| | |
|------|--|
| x | Coordinate x della palla |
| y | Coordinate y della palla |
| a | Valore temporaneo di "x" per cancellare |
| b | Valore temporaneo di "y" per cancellare |
| dx | Incremento/decremento di "x" |
| dy | Incremento/decremento di "y" |
| h | Altezza aggiornata della palla |
| p | Battuta del giocatore |
| s | Punti |
| true | "Marker" normalmente a 1, assume il valore 0 quando la palla viene mancata |
| ball | Numero aggiornato delle palle rimaste |

Note

| | |
|-----------|--|
| 10 | Inizializza |
| 20-30 | Disegna la palla, memorizza i vecchi valori e calcola i nuovi |
| 40-60 | Test per vedere dov' e' la palla |
| 70-90 | Muove la racchetta |
| 100 | Cancella la palla |
| 110 | Ritorno del loop |
| 120-130 | Palla mancata |
| 140 | Test per vedere se la partita e' finita |
| 200-220 | Preparazione della palla successiva |
| 2000-2200 | Nuovi valori delle variabili, calcolati quando la palla tocca il fondo. Notate la linea 2020 |
| 3000-3200 | Nuovi valori calcolati per i bordi |
| 4000-4200 | Nuovi valori delle variabili inerenti al muro |
| 5000-5100 | Partenza della sola inizializzazione |
| 5300-5700 | Inizializzazione di ogni palla |
| 6000-6400 | Disegna lo schermo, i punti e le palle rimaste |
| 8000-8030 | "Bit" di fine gara |

```

10 GO SUB 5000: REM iniz
110 REM inizio del loop
2000 LET x,y: LET a=x: LET b=y
3000 LET x=x+dx: LET y=y+dy
4000 IF y<0 THEN GO SUB 2000
5000 IF x<0 OR x>248 THEN GO SUB
30000

```


[illegible]

Il computer lista una "videata", quindi si arresta presentando la richiesta 'scroll?' a cui l'operatore puo' rispondere a seconda delle proprie necessita' premendo un tasto qualsiasi per lo scrolling dello schermo verso l'alto, oppure digitando "n", "STOP" o "BREAK" (CAPS SHIFT+SPACE) per arrestare il programma. Questa particolare proprieta' viene sfruttata di buon grado in diversi games quando, ad esempio, si voglia ottenere una astronave in viaggio attraverso ad uno stormo di meteore. Facendo scorrere il paesaggio e mantenendo ferma, o quasi, la navetta, si ha infatti l'impressione che questa si muova in una certa direzione. In condizioni normali, pero', lo Spectrum presenta la scritta 'scroll?' dopo lo scorrimento di ogni 22 righe, il che, nel bel mezzo di un game, non e' molto ben accetto. A tale inconveniente si rimedia semplicemente effettuando una POKE, come avviene nella linea 20 del prossimo programma, e facendola seguire da uno statement di PRINT (linea 50) che porta la posizione di scrittura al limite inferiore dello schermo.

Siete al comando di una astronave che dovete pilotare con i tasti 5 e 8 i quali hanno il potere di cambiare la direzione di marcia per evitare collisioni con le meteore. Attenti perche' l' astronave che lascia dietro a se una coda luminosa, effetto dello scrolling, puo' rimbalzare sui fianchi dello schermo. La lettera "A", presente alla linea 50, va battuta in "graphic mode".

[illegible]

[illegible]

Ricorrendo al codice macchina (m/c), e' possibile ottenere anche lo scrolling a destra e a sinistra. Inserite la routine che segue e vedrete che tutti gli oggetti che escono dallo schermo in un determinato punto, riappaiono tali e quali nel punto diametralmente opposto dando adito ad un singolare effetto. Tenete presente che la subroutine gira sullo Spectrum da 48K, se avete la versione a 16K, cambiate i numeri che seguono :

```

Linea 10      da 64999 a 31999
Linea 8000    da 65000 a 32000
              da 65045 a 32045
Linea 8019    da 65000 a 32000
Linea 8049    da 65023 a 32023

```

Naturalmente i numeri piu' bassi sono dovuti alla minor memoria del 16 rispetto al 48K.

[illegible]


```

8110 DATA 1,255,2,17,255,90,33,2
54,90,237,184,1,0,24,17,255,87,3
3,254,87,237,184,201

```

Uno dei pochi svantaggi dello scrolling, e' che transla tutto quanto riportato sullo schermo ivi compreso il punteggio, che, come si sa, permane costantemente nella parte alta. Pertanto, usando un tale effetto, e' necessario memorizzare il risultato senza presentarlo se non alla fine del gioco. Ed ora eccovi alcuni interessanti programmi basati sullo scrolling. Iniziamo da "Grand Prix". Siete alla guida di un bolide blu attraverso un tracciato di cui dovete evitare i bordi e nello stesso tempo le auto piu' lente che vi si presentano davanti. Lo spostamento coi soliti tasti 5 e 8.

Elenco delle variabili

| | |
|------|---|
| p | Coordinate della colonna dell' auto del giocatore |
| s | Punti |
| x(.) | Usato nella cancellazione del fumo |
| car | Posizione prossima dell' auto da corsa |
| ink | Colore dell' auto da corsa |
| rnd | Numero random per lo snodarsi della pista |
| life | Auto a disposizione (inizialmente 5) |
| l | Coordinata della colonna della pista |
| i, | |
| n | Variabili di controllo dei loop |
| s, | |
| END, | |
| LOOP | Memorizzano i numeri delle subroutines |

Gli UDG presenti in questo programma, corrispondono alle lettere maiuscole di cui alle linee :

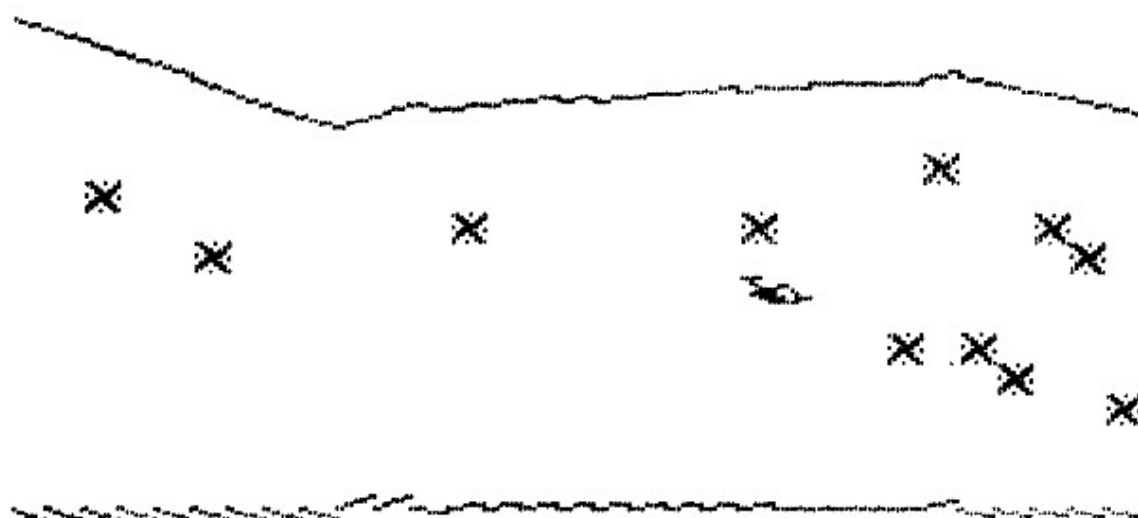
30,60,110,130,2000 e 5120.

```

10 GO SUB 5000: REM iniz
100 REM **ROUTINE PRINCIPALE**
200 POKE 23892,-1: PRINT : REM
SCROLL
30 PRINT INK 0;AT 9,p;"FG": PR
INT AT 10,p;" "
40 LET p=p+(INKEY$="8")-(INKEY
$="5")
50 IF ATTR (11,p)<>113 OR ATTR
(11,p+1)<>113 THEN GO TO END
60 PRINT INK 1;AT 10,p;"AB";AT
11,p;"CD"
70 LET s=s+1
80 PRINT AT 7,x(1);" "
90 LET x(1)=x(2): LET x(2)=x(3
): LET x(3)=p
100 LET car=INT (RND*9+l+2): LE
T ink=INT (RND*4+2)

```


HQDNHQ
DDH-HQ
DODGDA
WIO-
--T R
...V DD
RSC
WH-D
TZ-D
O-TOM
U LP
DS : PDPBR S: INK 7: B
T 10, 13; FLTH 1; " C



Note

| | |
|-----------|---|
| 10 | Inizializzazione |
| 20 | Scroll dello schermo |
| 30-90 | Tratta il movimento dell' auto, delle nuvolette di fumo e rileva se la vettura ha picchiato |
| 100-200 | Sistema l' auto successiva e la pista |
| 2000-2020 | L' auto ha urtato qualcosa |
| 2040-2060 | Fine di una gara |
| 2080-2100 | Nuova partita |
| 5010-5060 | Inizializzazione variabili |
| 5100-5140 | Predisposizione dello schermo |
| 8000-8040 | Definizione UDG |
| 8099-8160 | Dati per gli UDG |
| 9990-9999 | Fine |

Il prossimo game si avvale del codice macchina per effettuare lo scroll orizzontale. E' la versione semplificata del noto "Scramble". Dovete attraversare una lunga caverna le cui pareti si fanno sempre piu' strette senza incocciare nelle mine sospese qua' e la'. La navetta si pilota per mezzo dei tasti W,A,D e Z disposti nelle quattro direzioni cardinali; azionandone due assieme si ottiene lo spostamento diagonale. Tale effetto deriva dall'uso della funzione IN come sopra descritto. Il tasto A (sinistra) produce l'illusione del rallentamento della marcia, mentre il D, quello dell'accelerazione.

"s\$", "r\$" e "q\$" contengono rispettivamente una stringa vuota, la navetta e le mine. Nelle relative stringhe sono racchiusi i caratteri di controllo degli attributi in modo che le varie parti assumano l' appropriato colore per mezzo di semplici PRINT.

PRINTando "q\$" si ottiene ad esempio una mina rossa lampeggiante.

"a\$" usa il carattere di controllo di AT per presentare il disegno nella parte bassa dello schermo onde prevenire l' effetto di "wrap around" (gia' visto come la scomparsa di un oggetto da un lato e la sua riapparizione dal lato opposto) causato dalla subroutine in codice macchina.

POKE 23624,n viene spesso e volentieri impiegato per variare le due linee piu' basse dello schermo; qui' copre un effetto causato dal codice macchina visibile modificando la linea 5330.

Elenco delle variabili

| | |
|-----|---|
| x | Coordinata verticale della navetta |
| y | Coordinata orizzontale della navetta |
| s\$ | Stringa per cancellare la vecchia navetta |
| l | Variabili fittizie per eseguire la subroutine in codice macchina |
| a\$ | Stringa per eliminare il "wrap-around" |
| r\$ | Stringa per disegnare la navetta |
| s | Punti |
| g | Coordinata verticale del suolo della caverna |
| w | Altezza della caverna |
| d | Usato per dare forma ai muri della caverna |
| r1 | Coordinata verticale random delle mine |
| r2 | r1 convertito in numero di pixel per far capire al programma il rapporto tra r1 stesso e la larghezza della caverna |
| q\$ | Stringa per disegnare le mine |
| men | Numero delle navette rimaste |
| b\$ | Stringa fittizia usata in INPUT per dare il via ad una nuova ondata |
| i, | |
| n | Variabili di controllo per i loop FOR...NEXT. |

Gli UDG usati nel programma sono tre. Eccoli : "AB" alla linea 80, "AB" alla linea 5120 e "C" alla linea 5130.

```

100 CLEAR 31999: GO SUB 5000
110 REM SUB 1 ISSUE 2, variare
120 REM SUB 1 in 107, 253 in 109
130 REM SUB 254 in 100
140 PRINT AT X,Y: s$: LET X=X-(I
N 645-10=250)+(IN 55=251): LET
6 UN 650-(IN 650=254 AND Y>1)+(IN
300 LET L=USR 32000: PRINT a$: A
T X,Y: r$: LET s=s+1: PLOT 239,9:

```



```

0000100 FOR i=1 TO 3
0000200 FOR n=0 TO 7: READ a: POKE
0000300 CHR$(i+143)+n,a: NEXT n
0000400 NEXT i
0000500 RETURN
0000600 DATA 240,300,15,15,53,15,3
0000700 DATA 1,0,0,4,10,0,7,15,4,0
0000800 DATA 1,0,0,4,10,0,7,10,4,0
0000900 DATA 14

```

Note

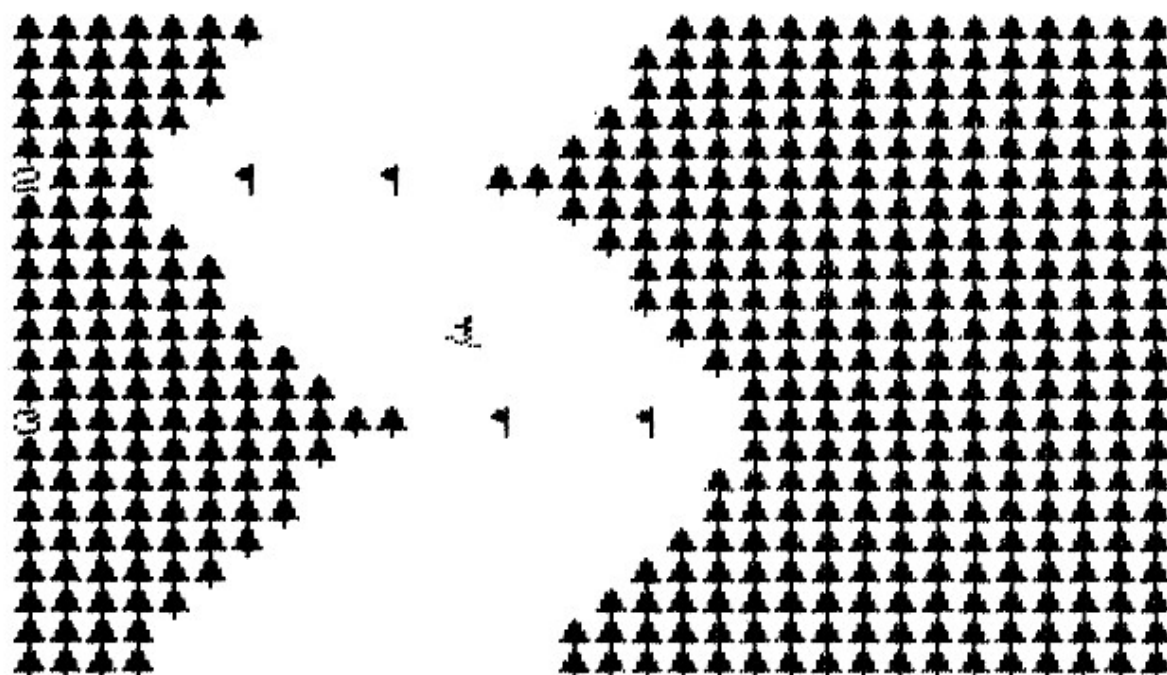
| | |
|-----------|--|
| 10 | Salvataggio della memoria per il codice macchina e inizializzazione |
| 20 | Cancella la navetta, predispone gli INPUT e muove |
| 30 | Effettua lo scroll, aggiorna il punteggio e disegna la caverna |
| 40 | Disegna le mine |
| 50 | Controlla se la caverna esce dallo schermo |
| 60 | Varia l' involuppo della caverna |
| 70 | Riduce l' ampiezza e rileva se la navetta ha urtato qualcosa |
| 80-90 | Urto |
| 100 | Predisporre una nuova ondata |
| 110-140 | Fine della gara |
| 5020-5130 | Variabili da non ridefinire |
| 5200-5210 | Altre variabili |
| 5300-5330 | Predisporre lo schermo |
| 5480 | Introduce il suono |
| 8000-8100 | Predisporre la routine di m/c |
| 9000-9040 | Definisce gli UDG |
| 9100-9120 | Dati per gli UDG |

L' ultimo game di questo capitolo, presenta uno scrolling verticale. Dovete controllare uno sciatore rosso facendolo scendere in slalom attraverso 25 porte. Se urta una bandierina, un albero o salta una porta, la gara ha termine.

I controlli a vostra disposizione sono : F-sinistra, G-giu' e H-destra. Una volta premuto il tasto, lo sciatore prosegue nella stessa direzione fino a che non azionate un altro tasto.

Elenco delle variabili

| | |
|-----|---|
| n | Loop dello sciatore prima che appaia la porta |
| p | Coordinata della colonna dello sciatore |
| a\$ | Stringa contenente l' input da tastiera |
| d | Direzione dello sciatore |
| e | Posizione delle porte |
| r | Numero random relativo alla posizione della porta successiva |
| ri | Aggiunto a "g" per cercare la posizione in cui piazzare gli alberi |



z# Stringa contenente una fila intera di alberi
 presentata con dei PRINT
 a(,) Array relativo al motivetto
 m Si alterna tra 0 e 8 per suonare la prima o la
 seconda meta' della melodia
 e Marker per rilevare se lo sciatore si e'
 schiantato o no
 s Loop che predispone il conteggio delle porte
 g# Stringa contenente la porta. Contiene i caratteri
 di controllo del colore. Lo sciatore deve passarvi
 in mezzo ma non attorno
 n,
 i Usate come variabili di controllo dei loops



Caratteri UDG sono presenti alle linee 310-graphic B; 320
 graphic E; e 5120-graphic D.


```

000000$ 0+"D  "
000100 DIM a(16): RESTORE 8500
000150 FOR i=1 TO 16: READ a(i): N
000200 NEXT i
000250 LET p=15: LET d=2
000300 LET r=0: LET r1=0
000350 LET e=0: LET g=15
000400 BORDER 4: PAPER 7: INK 2: C
000450 GO TO 150
000500 REM Dati per motivo
000550 DATA 0,0,0,0,9,9,11,0,0,0,0
000600 ,5,7,0
000650 RESTORE 9000: FOR i=1 TO 5:
000700 n=0 TO 7: READ a: POKE USR
000750 (i+143)+n,a: NEXT n: NEXT i
000800 RETURN
000850 DATA 48,48,28,48,16,37,28,2
000900
000950 DATA 36,36,128,128,60,36,36
001000
001050 DATA 12,12,56,12,8,196,56,7
001100 DATA 4,28,60,28,4,4,4,4
001150 DATA 8,28,28,52,52,127,8,8

```

Note

| | |
|-----------|--|
| 20-110 | La routine principale e' una subroutine usata in due occasioni |
| 30 | Cancella il vecchio sciatore ed effettua lo scroll |
| 40-50 | Movimento dello sciatore |
| 60 | Disegna il nuovo sciatore |
| 70-80 | Disegna gli alberi |
| 90 | Suono |
| 100 | Rileva l' eventuale urto |
| 150-210 | Loop delle porte che richiama la routine principale |
| 180-200 | Disegna la porta |
| 250-260 | Presenta lo sciatore alla fine, dopo l' ultima porta |
| 300-400 | Fine |
| 500-520 | Routine dell' urto e fine |
| 5000 | Definisce gli UDG |
| 5110-5120 | Stringhe per disegnare alberi e porte |
| 5130-5160 | Inserisce i dati nell' array per il motivetto |
| 5200-5220 | Definisce le variabili |
| 5300 | Colori |
| 9000 | Definisce gli UDG |
| 9100-9140 | Dati per gli UDG |

USO DEGLI ARRAY

Gli array (matrici) sono delle variabili e come tali vanno usati anche se e' possibile affidargli diversi altri compiti. Esistono, come per le variabili, due tipi di array: quelli numerici che contengono unicamente numeri e quelli di caratteri che memorizzano lettere o stringhe. Per comprenderne bene la struttura, immaginateli disposti come una tabellina divisa in settori ciascuno dei quali ospita un numero o un carattere. Prima di poterli usare, vanno dimensionati per mezzo dello statement DIM:

```
10 DIM A (500)
```

In tal modo si prenota lo spazio in memoria per la "tabella" di nome A, la quale inizialmente ha tutti i suoi elementi a 0. Il nome e' seguito da un numero racchiuso tra parentesi chiamato "subscript" della variabile. Analogamente avviene per gli array di caratteri.

```
10 DIM A$ (500)
```

significa che l'array A e' formato da 500 elementi ognuno dei quali contiene un carattere che inizialmente e' uno spazio. Provate ad aggiungere alla linea di cui sopra:

```
20 LET A$ (1) = "Ciao"
```

e quindi date il RUN. Fatto cio' inserite il comando diretto, PRINT A\$ (1) e vedrete che la risposta sullo schermo non sara' "Ciao" bensì solo "C" in quanto ogni singolo elemento non puo' memorizzare piu' di un carattere. Volendo pertanto sistemare "Ciao" in un array dovrete ricorrere a piu' di una dimensione come dimostrato facilmente da:

```
10 DIM A$(100,5)
```

In questo modo avrete dimensionato una matrice da 100*5=500 elementi in totale. Battete ora:

```
20 LET A$(1) = "Ciao"
```

fatelo girare e quindi date PRINT A\$(1). A questo punto la risposta sara' "Ciao" in virtu' del fatto che A\$(1) viene considerato come un array a se' stante lungo 5 caratteri. L'analisi di ogni sua parte e' facilmente ottenibile studiando lo statement:

```
30 PRINT A$(1),A$(1,3)
```

il quale mostra sia A\$(1) per intero, sia il suo terzo carattere. Riscrivendo A\$(1,3) come A\$(1)(3), otterrete una ulteriore tavola di altre due dimensioni. Cio' per dimostrare come un array possa avere quante dimensioni si voglia. Ad esempio, DIM A\$(3,6,7,2) ha quattro dimensioni e possiede $3*6*7*2=252$ elementi. Naturalmente, ogni elemento deve venire caricato con lo stesso numero di "subscript"; non ha senso, ad esempio,

```
10 DIM A(10,2)
20 LET A(1)=7
```

poiche' il computer non sa se caricare il 7 in A(1,1) oppure in A(1,2).

Usando gli array con un loop continuo, potete generare motivetti con dei BEEP random disponendo appropriati valori in ogni elemento :

```

10 GO SUB 5000
20 BEEP : a(m): LET m=m+1-(64
AND m=64)
30 REM Resto del programma
40 GO TO 20
50 DIM a(64)
60 LET m=1
70 RESTORE 6000
80 FOR a=1 TO 64
90 READ a(a)
100 NEXT a
110 RETURN
120 DATA -5,-3,-5,0,-5,0,-5,0,-
130 DATA 4,4,-3,0,4,0,-3,0,-
140 DATA -5,2,-5,2,-5,2,-
150 DATA 5,4,5,0,-3,-5,4,4,4,2,2,2
160 DATA 4,-3,-5,2,-5,4,-5,5,-
170 DATA -5,-5,-3,-1,2,0

```

Constaterete voi stessi questo piacevole effetto nell' ultimo programma del capitolo.

Gli array possono anche contenere le coordinate degli oggetti da muovere sullo schermo. Un esempio di cio', lo potete avere sia nel game "Zap", in cui l' incremento di ogni singola variabile muove a turno un invader, sia nel programma che segue intitolato "Gara delle pulci".

```

10 GO SUB 5000
20 FOR i=1 TO 5
30 PRINT AT i*3+2,x(i); " "
40 LET x(i)=x(i)+RND*3
50 PRINT AT i*3+2,x(i); "A"
60 IF x(i)>27 THEN GO TO 30

```



```

00000000 NEXT      0: NEXT      1: RETURN
00000001 DATA      102,102,24,129,125,0,3
00000002
00000003
00000004 DATA      51,51,204,204,51,51,20

```

Gli UDG sono insiti nel listato alle linee 50-graphic A,
90-graphic B, 5320-graphic A.

Ogni giocatore puo' scommettere su quale pulce vincerà; se entrambi sbagliano, perdono la cifra puntata mentre se vincono incrementano il bottino.

Le matrici possono essere impiegate anche come "marker" dello stato di qualsiasi cosa presente sullo schermo. Lo vedremo tra poco nell' impegnativo programma "ICBM".

Gli array di stringhe trovano particolare uso nella presentazione del display. Battete il programmino che segue il quale, anche senza l'uso di array, dà l'impressione del movimento continuo :

```

10 LET a$=""
20 LET i=1
300 BORDER 0: PAPER 0: CLS
400 BORDER 7: INK 1
500 PRINT AT 10,1;a$(i TO 30);a
# (1 TO i)
600 LET i=i+1-(30 AND i=30)
700 GO TO 50

```

Sullo stesso principio, il prossimo che di linee in movimento ne presenta tre :

```

10 LET A$=""
20 L=1
30 FOR P=1 TO 30
40 FOR R=1 TO 30
50 FOR I=1 TO 30
60 PRINT P;R;I;A$(I TO 30)
70 I=I+1-(I=30)
80 NEXT I
90 NEXT R
100 NEXT P
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300
310
320
330
340
350
360
370
380
390
400
410
420
430
440
450
460
470
480
490
500

```

Piu' difficile si presenta il discorso quando le tre linee devono scorrere in direzioni differenti. Il programmino che segue impiega due variabili denominate "i" per sfilare la linea di centro verso destra e le due laterali verso sinistra.

[illegible]

```

300 BORDER 0: PAPER 0: CLS
400 PAPER 7: INK 1
500 FOR n=1 TO 3
550 LET it=(il AND n<>2)+(ir AND
D 700=2)
600 PRINT AT 8+n*2,1;a$(it TO 3
0) ;a$(1 TO it)
700 NEXT n
800 LET il=il+1-(30 AND il=30)
900 LET ir=ir-1+(30 AND ir=1)
950 GO TO 50

```

Il risultato e' accettabile, ma desiderando variare la velocita' di scorrimento e' necessario apportare complicazioni al programma. E qui subentrano gli array. Per ogni riga, sono a disposizione diversi "quadri" ordinati in modo che la loro sequenza dia l' impressione del movimento. I "quadri" sono memorizzati in un array e le righe in un' altra dimensione dello stesso array.

Per ottenere il movimento di tre righe separate, dovete ricorrere allo statement DIM A\$(3,n1,n2) in cui "n1" e' il numero dei quadri prima che la sequenza si ripeta e "n2" e' la lunghezza di ogni quadro nell' array di stringhe. I due programmi che seguono, mostrano gli array in azione. Nel primo, "Snakey", disponete di un serpente (formato da una fila di "V") che voi dovete guidare in uno stretto cunicolo badando di evitare i blocchi che vi si fanno incontro. La testa del serpente, rappresentata dalla prima "V", la spostate a sinistra con il tasto 5 e a destra con il tasto 8. La partita ha termine dopo aver urtato dieci blocchi ognuno dei quali viene segnalato a fianco con il numero relativo lampeggiante.

Elenco delle variabili.

| | |
|--------|--|
| a(.,.) | Array con i dati del motivo |
| c | Contatore per la parte di array interessata |
| a | Posizione dei blocchi |
| p | Numero della colonna con la testa del serpente |
| n | Numero dei blocchi urtati |
| s | Punti |
| i | Controllo dei vari loop |
| v | Durata della nota del motivetto |
| x | Tono della nota |
| x# | Scrivo i messaggi a fine gara |
| a | Variabile di controllo per predisporre a(.,.) |

```

10 CLS
20 GO SUB 5000
30 LET a=INT (RND*4)+10
40 POKE 23692,-1: PRINT AT 21,
0: PRINT
50 LET p=p+(INKEY$="8" AND p<1
3) -(INKEY$="5" AND p>10)
60 IF ATTR (16,p)=16 THEN LET
n=n+1: BEEP .03,24: PRINT AT 21,

```

[illegible]

Note

```
20      Inizializzazione
40      Scroll
50      Mosse del giocatore
60      Rivela gli urti
```

| | |
|-----------|-------------------------------------|
| 70 | Disegna il serpente e i blocchi |
| 80 | Musica |
| 90 | Punti |
| 200-210 | Fine della partita |
| 220-240 | Bit di fine gara |
| 250-4990 | Fine del programma |
| 5000-5010 | Inserisce i dati nell' array |
| 5020-5030 | Definisce le altre variabili |
| 8000-8030 | Dati per il motivo durante la gara |
| 8100 | Dati per il motivo di fine gara |
| 8200-8240 | Dati per messaggio a fine programma |

Il secondo programma, "ICBM", e' una versione del piu' noto "Missile command".

Scopo del game, e' il salvataggio di sei citta', mostrate sottoforma di blocchi, da uno stormo di missili che si presentano ad ondate successive di venti elementi, ognuna piu' veloce della precedente. Potete distruggere i missili in caduta posizionandovi al disotto e sparando con lo 0.

La vostra posizione varia in senso antiorario col tasto "6" e in senso orario col "7". Se avete qualche perplessita', consultate la freccia nella parte superiore dello schermo. Il game ha termine con la distruzione delle sei citta'. Ad ogni 10.000 punti vi viene regalata una citta', il numero dei punti aumenta ad ogni ondata e varia a seconda dei missili colpiti e delle munizioni risparmiate...il resto lo scoprirete da voi.

Elenco delle variabili

| | |
|-------|--|
| mis | Missile in discesa dal cielo |
| f(.) | Array relativo ai missili |
| m(.) | Punto di origine dei missili situato sulla parte alta dello schermo |
| x(.) | Coordinata orizzontale della testa del missile |
| y(.) | Coordinata verticale della testa del missile |
| d(.) | Numero dei pixel necessario al missile per raggiungere la citta' da distruggere |
| b(.) | Citta' selezionata dal missile. Se e' la 3 il missile centra la base |
| c(.) | Array che riporta quale delle citta' sia stata distrutta |
| a(.) | Citta' selezionata (in b(.)) sottoforma del numero di pixel orizzontale |
| c | Numero delle citta' distrutte |
| x | Coordinata orizzontale della base |
| y | Coordinata verticale della base |
| b\$ | Usata per l' input |
| dir | Direzione della base 0=su, 1=su a destra, 2 a destra ecc... |
| move | Numero dei pixel della base mossi ogni volta (diminuisce man mano che la gara si fa piu' difficoltosa) |
| ammo | Munizioni residue |
| count | Missili per ogni ondata |

| | |
|-------|---|
| ink | Colore di INK |
| z | Usato come controllo dei loop per vedere se ogni missile colpisce (linee 420-460) |
| n1 | Valore per predisporre un nuovo missile |
| sc | Punti |
| dd | Grado di difficoltà |
| paper | Colore di PAPER |
| te | Valore temporaneo di b(.) per calcolare la scrittura della colonna per l'esplosione della città |
| city | Quale città è esplosa |
| mark | Usato per calcolare se tutte le città sono state colpite |
| wave | Conto delle ondate |
| abm | Punti delle munizioni rimaste |
| city | Punti delle città rimaste |
| down | Numero dei pixel relativi ai missili caduti dopo ogni ondata |
| ink 1 | Colore delle munizioni a terra |
| sc 1 | Punteggio in 10.000, usato per calcolare quando dare una città gratis |
| xc | Memorizzazione delle città elargite. Se tutte le città sono ancora in piedi dopo aver raggiunto i 10.000 punti, la città viene conservata per ulteriori bisogni |
| nc | Posizionamento random della nuova città |
| a1, | |
| a2, | |
| a3 | Usati per la casualità dei colori di INK e PAPER |

```

10 GO TO 3000: REM Iniz
30 GO TO 500
99 REM Loop
100 LET mis=mis+1-(4 AND mis=4)
110 IF f(mis) THEN GO TO 200
120 INVERSE 1
130 PLOT m(mis),153: DRAW x(mis)
) -m(mis),y(mis)-153
140 LET y(mis)=y(mis)-down
150 LET x(mis)=x(mis)+(d(mis)*d
own)
160 INVERSE 0
170 IF y(mis)<16 THEN GO SUB 60
0: GO TO 190
180 PLOT m(mis),153: DRAW x(mis)
) -m(mis),y(mis)-153
190 IF c=6 THEN GO TO 4000: REM
FINE
200 INVERSE 1: PLOT x-2,y: DRAW
4,0: PLOT x,y-2: DRAW 0,4
210 LET b$=INKEY$: IF b$="6" TH
EN LET dir=dir-1+(8 AND dir=0)
220 IF b$="7" THEN LET dir=dir+
1-(8 AND dir=7)

```


INK 9; " mun.

```

00000000 DNTD 0,0,00,40,70,0,0,0
00000000 DNTD 0,0,00,10,10,0,0,0
00000000 DNTD 0,0,00,4,0,54,4,0,0
00000000 DNTD 10,0,0,4,0,10,10,0
00000000 DNTD 10,10,10,140,0,4,0,0
00000000 DNTD 1,0,00,70,00,0,0,0
00000000 DNTD 0,10,00,10,00,0,0,0
00000000 DNTD 0,10,4,0,0,0,0,0
00000000 REM Row in the cell
00000000 DNTD 0,0,0,0,4,14,10,0
00000000 DNTD 0,0,0,0,0,0,0,0

```

Note

```

10      Inizializzazione
30      Fuori gioco (succede!)
100     Partenza del loop.
100-180 Movimento del missile
190     Testa se tutte le città' sono state distrutte
200     Cancella l' attraversamento della base
210-240 Valuta l' input e calcola le nuove coordinate
250     Disegna la base
260-300 Testa lo sparo e disegna la freccia
310     Partenza di una nuova ondata
400     Disegna il laser
420     Testa la collisione
430     Cancella il missile
440     Predispone un nuovo missile
450-470 Aggiorna il punteggio, testa il nuovo missile
        scala le munizioni
490     Stampa il nuovo punteggio
500     Stampa i punti
610     Test per vedere se il missile ha colpito la base
630     Esplosione
670-680 Cancella la città dall'array
690     Predispone un nuovo missile
720-740 Predispone i nuovi valori per il missile
800-830 Il missile ha colpito la base, non esplode ma il
        punteggio cala
1010    Nuovo valore dei punti
1020-1070 Punti per le città' rimanenti
1080    Aumento delle difficoltà'
1090    Punti per le munizioni rimanenti
1110-1180 Aggiunge una nuova città' se si sono superati
        10000 punti
1190    Inizializza il riposo, per una nuova ondata
2000    Colori random
2020-2050 Predisposizione dei colori random
2060-2070 Disegna lo schermo
2080    Stampa il punteggio
2090    Munizioni rimaste
2100-2160 Disegno della base e delle città'
3000    Definizione degli UDG

```

| | |
|-----------|--|
| 3100-3140 | Inizializzazione delle variabili (anche 3220-3230) |
| 3200 | Array per i missili |
| 3240 | Predisporre i missili |
| 3300 | Inizializza l'attraversamento del cursore |
| 3400 | Disegna lo schermo |
| 4000 | Fine della gara |
| 9000-9020 | Definisce gli UDG |
| 9100-9130 | Dati per gli UDG |

PEEK e POKE

Sono senza dubbio due tra i comandi piu' usati dello Spectrum. Per capire bene le funzioni che essi svolgono, e' necessario conoscere l' hardware della macchina e il suo funzionamento. Come gia' saprete, ogni computer possiede una memoria la quale si divide in due tipi ben distinti. La ROM (abbreviazione che sta per Read Only Memory = memoria a sola lettura) contiene le informazioni necessarie al funzionamento della CPU (Central Processing Unit = Unita' centrale di processo o piu' semplicemente microprocessore) le quali rimangono memorizzate anche quando l' apparecchio viene spento. Il contenuto della ROM non puo' venire alterato in alcun modo.

Il secondo tipo di memoria si chiama RAM (Random Access Memory = Memoria ad accesso casuale) e memorizza i dati provenienti dalla CPU. Le informazioni e i dati in gioco, sono tutto quanto presente in un programma e visualizzato sullo schermo a partire dalla grafica definita dall' utente (UDG) per arrivare alle variabili del sistema. Al contrario di quanto accade per la ROM, il contenuto della RAM va perso quando si toglie corrente e puo' venir alterato in qualsiasi momento.

Vediamo ora in quale modo viene memorizzata una informazione. Immaginatevi la memoria suddivisa in compartimenti ognuno dei quali in grado di contenere un numero; sia la quantita' dei primi che quella dei secondi dipende dal tipo di computer. Lo Spectrum dispone di 65536 (da 0 a 65535) compartimenti, chiamati bytes capaci ognuno di memorizzare un numero compreso entro la gamma 0-255. Per comodita' i bytes vengono raggruppati a mille per volta e ognuno di tali gruppi e' conosciuto col nome di "Kilobyte", abbreviato con "K". Per essere piu' precisi, un K equivale a 1024 bytes.

Il comando PEEK vi permette di accedere a questi bytes con la forma "PEEK n" dove "n" e' appunto il byte da esaminare.

Il comando POKE, invece, vi abilita addirittura a cambiare il contenuto per mezzo dello statement "POKE n,m" in cui "n" e' il byte, o meglio l' indirizzo, presso il quale effettuare la variazione e "m" il valore del nuovo dato compreso tra 0 e 255. POKE viene quindi usato per memorizzare nuovi dati, in una area di memoria riservata e PEEK per andarli a ripescare.

Vi sono diversi modi per riservare la memoria onde proteggere i dati da alterazioni accidentali. Uno di questi sistemi prevede l' uso dello statement di REM.

Se la prima linea e' :

```
1 REM AAA
```

La memoria viene prenotata sotto il segno delle A. Provate a battere questa linea facendola seguire dal comando diretto POKE 23760,127 e vedrete che, listando nuovamente la linea, al posto della prima A troverete il simbolo di copyright corrispondente

appunto al carattere 127. Provate con altri valori. L'indirizzo 23760 si riferisce appunto al primo carattere dopo il REM presente alla linea 1. Per pochi dati potete quindi utilizzare lo statement di REM dotandolo di tanti caratteri quanti sono i dati per poi POKare questi ultimi nei vari indirizzi messi a disposizione dallo statement stesso.

Una seconda via per proteggere i dati la mette a disposizione CLEAR il quale li preserva anche dal comando NEW che ha il potere di azzerare tutto quanto.

Di solito i caratteri a disposizione dell'utente, che sono in tutto 21, occupano gli ultimi 168 bytes della memoria in quanto ogni carattere ne impiega 8. Nello Spectrum da 16 K, essi occupano gli indirizzi da 32600 a 32767, in quello da 48 K gli indirizzi impegnati partono da 65368 e arrivano fino a 65535. Lo statement "CLEAR n" predispone una variabile di sistema (che vedremo piu' avanti) all'indirizzo "n".

Se "n" e' piu' alto di 32600 nello Spectrum da 16 K si perde un carattere-user, predisponendolo piu' basso, si prenota una zona di memoria da n+1 a 32599. Analogamente accade nella versione da 48 K con i relativi indirizzi.

Con POKE si predispongono le variabili di sistema le quali vengono considerate dalla CPU per valutare quanto sta succedendo e prendere determinate decisioni. Usando correttamente questa funzione e' possibile creare un gran numero di effetti speciali. Pokando, ad esempio, 23609 col numero 100 potete generare, alla pressione di qualsiasi tasto, un beep piu' consistente del normale che puo' risultare utile nelle operazioni di INPUT dei games. POKando un numero maggiore di 1 all'indirizzo 23692, il computer non vi chiederà piu' lo "scroll?" ogni 22 linee di programma. Per cambiare il colore delle scritte relative ai messaggi sulla parte bassa dello schermo, non dovrete far altro che eseguire un POKE 23624 seguito da un numero a caso. POKE 23624 altera quindi gli ATTRibuti e i colori delle scritte o degli oggetti presenti nella parte alta dello schermo. Dando POKE 23659,1:CLS riempirete lo schermo di strisce la cui ampiezza dipende dal valore POKato in 23693.

Gli indirizzi 23677 e 23678 riguardano la funzione DRAW la quale puo' tracciare linee in tutte le direzioni partendo dall'ultimo punto stabilito da PLOT. Nell'esempio che segue, la posizione di plot orizzontale non e' 20 bensì 120 (100+20), così come quella verticale non e' 60 ma 140 (80+60).

```
10 PLOT 100,80
20 DRAW 20,60
```

Perche' una eventuale linea 30 possa iniziare a tracciare dai valori 0,0 e' necessario l'intervento di alcuni calcoli. Per mezzo delle variabili di sistema 23677 e 23678 potete simulare un DRAW assoluto. La linea 20 puo' essere riscritta nel modo che segue :

```
20 DRAW 120-PEEK 23677,140-PEEK 23678
```

e se prima avreste dovuto battere

```
30 DRAW -120,-140
```

ora dovete inserire

```
30 DRAW 0-PEEK 23677,0-PEEK 23678
```

Lo Spectrum puo' disporre del tempo in modo assai ben definito grazie alle tre variabili di sistema 23674,23673 e 23672. La piu' bassa delle tre viene incrementata ogni 50esimo di secondo fino a che non raggiunge 255 dopodiche' torna a 0 facendo partire il conteggio nell' indirizzo 23673 il quale, una volta raggiunto 255 abilita il terzo che, a sua volta raggiunge 255, resettandosi e facendo ripartire tutto da capo.

Il calcolo del tempo (espresso in secondi) e' assai semplice :

```
LETsecs=(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/150
```

Questa funzione e' di grande aiuto in games tipo auto da corsa e in qualsiasi caso in cui la gara debba essere portata a termine nel piu' breve tempo possibile. Alcune volte puo' succedere che il calcolo dei secondi dia un risultato sbagliato. Nel caso in cui, ad esempio, la locazione 23673 contenga 0 in conseguenza del fatto che lo 23672 abbia raggiunto 255 e poi si sia a sua volta azzerata, il computer puo' valutare contemporaneamente i valori 0 delle due variabili e di conseguenza considerare un tempo doppio di quello reale falsando i risultati. Per ovviare a tale fenomeno, e' necessario POKare a 0 ognuno degli indirizzi.

Oltre ai 22 caratteri, lo Spectrum ne puo' definire altri 96. Il segreto sta' nel bit del POKE il quale, se impiegato negli indirizzi 23606 e 23607, vi permette di cambiare il punto di partenza del set dei caratteri presente da CHR# 32 (spazio) a CHR# 127 (marchio di copyright).

Ecco come dovete fare :

- 1) Disegnate i nuovi caratteri da mettere al posto dei vecchi e calcolatene i dati.
- 2) Decidete i caratteri del set originale da ridefinire e prendete nota di quelli che dovranno ridefinirli. Di solito i caratteri da ridefinire (UDG=User-defined graphic) vengono scelti tra i primi del set originale. Se volete definire lettere e numeri tipo "era spaziale" e' bene segnarsi sempre l' accoppiamento tra i vecchi numeri con i nuovi e tra le vecchie lettere con le nuove. Questo perche', mentre state scrivendo lo statement di PRINT vi ricordate le equivalenze, ma, una volta fatto girare il programma, i caratteri ordinari lasciano il posto a quelli ridefiniti scomparendo dal contesto delle linee.

- 3) Prenotate la memoria usando CLEAR n (come descritto poco fa') o come comando diretto o inserendolo nel programma stesso. Il numero di bytes da riservare deve corrispondere a quello degli UDG compreso il primo e l'ultimo del set. Il numero va naturalmente moltiplicato per 8 essendo ogni carattere formato da altrettanti numeri. La cifra "n" viene ricavata sottraendo il numero così trovato da 32599 per lo Spectrum a 16 K e da 65367 per quello da 48 K.
- 4) Predisponete gli indirizzi 23600 e 23607 in modo che il primo UDG risulti a "n"+1. Per far ciò procedete come segue :
 - a) Trovate "n"+1
 - b) Contate quanti caratteri ci sono prima di quello da ridefinire; esempio, se il primo carattere da ridefinire è CHR# 38, avrete 38-36=6 caratteri.
 - c) Moltiplicate questa cifra per 8.
 - d) Supponiamo di chiamare quanto ottenuto, "char", per cui i valori da POKare sono dati da
 POKE 23606, char-256*INT (char/256)
 POKE 23607, INT (char/256)
- 5) Qualsiasi UDG, può cambiare anche caratteri del set alternativi, come potete vedere dal programma che segue il quale ridefinisce "#" con un carattere grafico.

```

1 REM Per lo Spectrum da 16K
2 CLEAR 32599: REM Valore di
"n"
3 LET num chars=1
4 LET bytes reserve=num chars
*8
5 LET n=32599-bytes reserve
6 LET char before first=CODE
"#"-32
7 LET byte before first=char
before first*8
8 LET char=n+1-byte before fi
rst-256
9 LET chr in 23606=char-256*I
NT (char/256)
10 LET chr in 23607=INT (char/
256)
20 FOR i=n+1 TO n+bytes reserv
e: READ a: POKE i,a: NEXT i
30 DATA 255,36,36,24,24,36,36,
255,0
40 POKE 23606,chr in 23606
50 POKE 23607,chr in 23607
60 PRINT 11,15; "#"
70 POKE 23608,0
80 POKE 23609,0

```

Differenze per lo Spectrum a 48 K

```
2 CLEAR 65380: REM valore di "n"
```

```
5 LET n=65367-riserva bytes
```

Esistono diverse variabili di sistema che non vengono usate normalmente come le altre ma che restano libere a disposizione. I loro indirizzi sono 23681, 23728 e 23729.

Molti computer posizionano gli oggetti sullo schermo per mezzo di PEEK e POKE, con lo Spectrum cio' non e' possibile. Se volete una dimostrazione fate girare il programma che segue e osservate gli strani risultati :

```
10 REM Per dimostrare l'impos  
sibilita' di pokare lo schermo  
200 LET screen=15384  
300 LET length=24*32*8  
40 FOR i=screen TO screen+length  
th  
500 POKE i,35  
600 NEXT i  
700 BEEP 1,10  
100 REM Per dimostrare l'impos  
sibilita' di pokare lo schermo  
200 LET screen=15384  
300 LET length=24*32*8  
40 FOR i=screen TO screen+length  
th  
500 POKE i,35  
600 NEXT i  
700 BEEP 1,10
```

Lo Spectrum risolve la cosa usando PRINT AT al posto di POKE e ATTR, POINT, SCREEN\$ al posto di PEEK.

Il nostro prossimo programma, di nome "Circuit", fa un esteso uso di PEEK e POKE e vi permette di guidare la vostra auto attraverso un intricato circuito sterzando nelle quattro direzioni cardinali e nelle quattro diagonali.

Al posto dell'auto si era pensato di disegnare un carro armato, ma i veloci cambi di direzione non si addicevano a tale mezzo. I tasti sono codificati per mezzo della funzione IN onde poterne azionare due contemporaneamente. L'auto dispone di quattro marce ed, alla partenza, e' innestata naturalmente la prima come potete constatare sul lato sinistro dello schermo. Se volete ingranare marce superiori (e quindi accelerare) premete "Q", per scalare "A". La guida e' molto realistica, col tasto "P" ruoterete in senso orario, se ad esempio siete diretti verso nord volterete verso nord-est oppure se siete orientati verso ovest girerete verso nord-ovest ecc...; col

tasto "O" la ruotazione avverrà in senso antiorario. Il game prevede il conteggio del tempo che viene però visualizzato solamente dopo aver concluso il giro oppure dopo esservi schiantati da qualche parte, per non rallentare eccessivamente lo svolgimento della gara. A vostra disposizione avete tre auto con le quali dovrete cercare di completare il tracciato; ogni volta che picchierete, dovrete ripartire daccapo. Il numero delle auto a vostra disposizione lo trovate nell'angolo in basso a sinistra.

La caratteristica più importante del gioco, e' sicuramente il set di caratteri definiti, pensate che, oltre agli ottanta UDG (di cui dieci sono numeri) sono state ridefinite, come visto sopra, anche diverse lettere comprese tra "a" e "z". Poiche' il tutto occupa diverso spazio in memoria, abbiamo suddiviso il programma in due tronconi per permettere il caricamento del secondo blocco (il gioco vero e proprio) anche sugli Spectrum da 16 K.

La prima parte predispone il set dei nuovi caratteri, definisce gli UDG e inoltre carica il secondo blocco. Se avete uno Spectrum da 48 K, potete congiungere assieme le due parti.

Molto interessante e' l'effetto di PRINT #0;"....." presente alle linee 8030 e 8060, infatti tale funzione abilita alla scrittura anche nelle due linee più basse dello schermo riservate normalmente ai commenti e ai messaggi di errore. Quanto scritto da PRINT 0, viene cancellato dagli INPUT alle linee 8020 e 8040 perche' non vizi il contenuto del display al sopraggiungere dello scrolling. Ora che conoscete alcuni segreti del programma, battetene la prima parte e registratela sul nastro con :SAVE "Circuit" LINE 10. Dopo aver eseguito il VERIFY, battete anche la seconda parte e registratela con :SAVE "C" LINE 10. Verificate quindi anche quest'ultima.

```

80093 REM Definizione caratteri
80094
80095 CLEAR 32399: RESTORE 9200
80096 PRINT AT 9,2: "FERMA IL REGI
80097 STRIP
80098 PRINT AT 11,10: FLASH 1: "AT
80099 END
80100 FOR i=32400 TO 32599
80101 READ a: POKE i,a: NEXT i
80102 FOR i=1 TO 18: FOR n=0 TO 7
80103 READ a: POKE USR CHR$(i+14
80104 )+n,a: NEXT n: NEXT i
80105 CLS: PRINT AT 9,7: "PREMI P
80106 REY: "; AT 11,5: "SUL TUO REGISTRATO
80107
80108 PRINT AT 13,0: "(Batti LOAD"
80109 )
80110 LOAD "C"
80111 REM Data for letters and
80112 symbols:
80113 DATA 0,24,36,36,126,102,102
80114
80115 DATA 0,124,66,66,126,70,70,

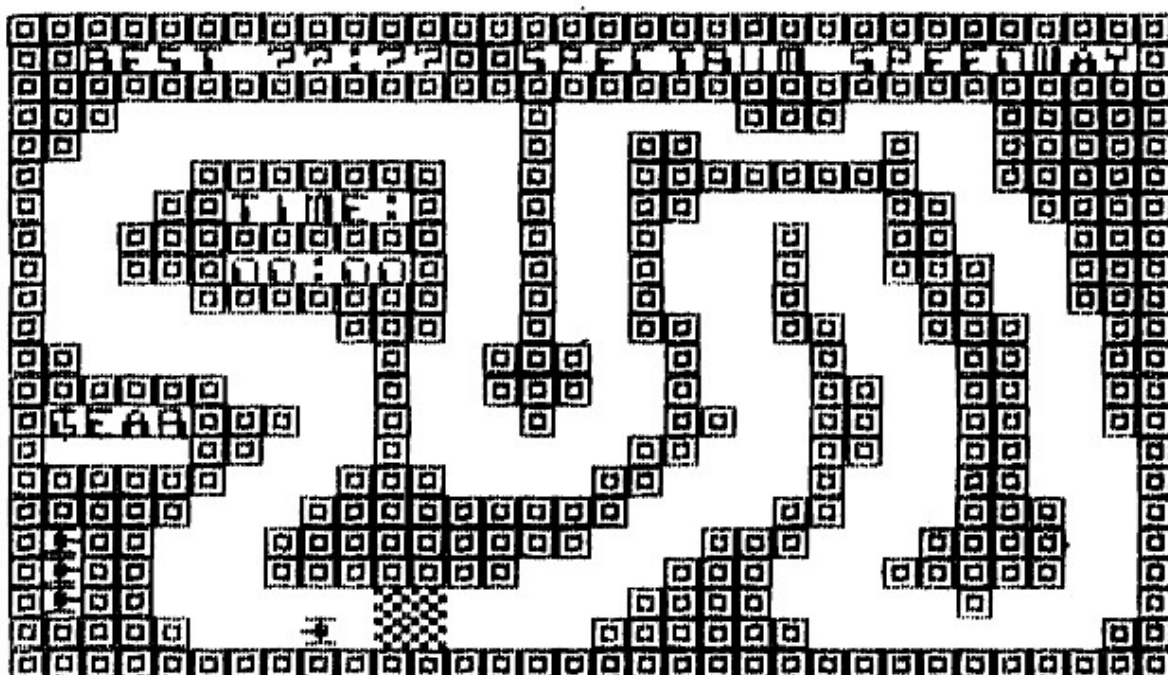
```



```

000000 DATA 0,64,96,100,100,126,4,
4
000000 DATA 0,62,62,64,60,6,6,126
000000 DATA 0,62,64,64,124,70,70,1
000000 DATA 0,100,60,6,10,10,10,10
000000 DATA 0,60,60,60,60,70,70,60
000000 DATA 0,60,60,60,60,6,6,10,60
: RETURN 000000,0: POKER 000007,60

```



Ammirate la copia dello schermo, a colori poi e' ancora meglio.

Ed ora vediamo la seconda parte del programma che crea il game vero e proprio.

Elenco delle variabili

```

h          Secondi impiegati e record
b*(2,5)b*(1) Contiene i caratteri del circuito in corso, b*(2)
              contiene gli spazi del circuito
n          Si alterna tra i valori 0 e 1 durante il disegno
              del circuito da parte dell' array b*(.)
c          Numero delle auto rimaste
m          Minuti
s          Secondi
x          Coordinata verticale dell' auto
y          Coordinata orizzontale dell' auto

```

dr Direzione dell' auto da 0 a 7. (0-su,2-destra ecc...)

g Marcia. Da 1 (piu' lenta) a 4 (piu' veloce)

al Legge la tastiera con IN per vedere se e' stato premuto il tasto 0 per l' accelerazione

dl Come il precedente ma stavolta riferito al tasto A per la decelerazione

mv Verifica se sia stato battuto uno dei tasti per spostare l' auto (0 oppure P)

a,
b Coordinate che la subroutine usa per disegnare qualsiasi cosa. Tale subroutine e' richiamata da due diversi posti, "a" e "b" vengono assegnati prima di chiamare la subroutine in modo che vengano PRINTate le coordinate esatte

t Tempo (in secondi) impiegato per completare il circuito. Viene comparato con "h"

z# Usata come stringa d' ingresso

LOOP,
ALTER,
NORM,
TIME,
START,
NEXT CAR,
END,
CRASH,
FINISH,
BYE Sono tutti "tokens" per la partenza delle subroutines: ALTER e' la subroutine usata per inserire il set alternativo di caratteri; NORM (normale) e' per cambiare di nuovo; TIME presenta tempo; START inizializza la partenza di una nuova gara; NEXT CAR inizializza una nuova auto; END e' il punto in cui finisce il programma quando l' auto urta contro qualcosa; CRASH e' per quando l' auto urta contro il circuito e FINISH e' per quando l' auto raggiunge la griglia di partenza.

Nel listing sono presenti degli user-graphic alle linee 5300 (quattro I) e 5310 (lettera C).

Circuito (Parte 2)

```

10 GO TO 5000: REM init/instr
30 REM Con lo Spectrum issue 2
31 REM Variare 255 con 191,
32 REM 254 con 190 e 253 con 1
33
100 LET al=IN 64510-255: LET dl
=IN 65022-255: LET mv=IN 57342

```


Note

| | |
|-----------|--|
| 10 | Inizializzazione |
| 100 | Legge la tastiera |
| 120 | Cambia la direzione |
| 140 | Cambia le coordinate |
| 160 | Disegna l' auto |
| 180 | Cambia marcia |
| 200 | Suono |
| 220 | Cancella l' auto |
| 240 | Verifica se l' auto ha urtato qualcosa |
| 1000 | Set di caratteri alternativo |
| 1500 | Set di caratteri normale |
| 2010 | Il tempo viene inserito in una stringa |
| 2030-2040 | Scrive il tempo |
| 5000 | Miglior tempo inizializzato |
| 5010-5070 | "Tokens" per le subroutines inizializzate |
| 5100 | Disegna le strisce |
| 5130 | Colori |
| 5150-5170 | Disegna lo schermo |
| 5300 | Scrive il tempo |
| 5310 | Auto rimaste |
| 5320 | Scrive sulla 23esima riga |
| 5430 | Azzeri i contatori del tempo |
| 6500-6590 | Dati usati alla linea 5150 per scrivere sullo schermo |
| 7000 | Cancella il marker delle marce |
| 7010-7020 | Calcola il tempo e lo presenta |
| 7030 | Salta a "CRASH" o a "FINISH" |
| 7500 | Ruota l' auto |
| 7510 | Riposiziona il circuito |
| 7520-7540 | Diminuisce il numero delle auto e salta di conseguenza |
| 8000 | Disegna l' auto |
| 8030 | Congratulazioni |
| 8050 | Inserisce il nuovo tempo nella variabile "best time" |
| 8080 | Riposiziona la griglia |
| 9000 | Verifica per un' altra gara |
| 9020 | Tutto pronto |
| 9980-9999 | Allegorie |

Seguono due ottimi programmi sviluppati originalmente per tutt' altro. L' idea originale era di usare una racchetta o una superficie scorrevole per catturare i blocchi che piovono dal cielo. Ogni blocco puo' valere da uno o due punti. I blocchi non catturati piombano al suolo accatastandosi fino a formare un muro che, se raggiunge l' altezza della racchetta, fa finire la gara.

Il primo dei due games basati su tale principio e' stato chiamato "Chomper".

Dovete mangiare il cibo che cade dal cielo composto da pere, salsicce, birra, polli arrosto e hamburger. Cadono anche dei

fucili che dovreste recuperare per poter sparare ai mostri che crescono mangiando quanto non siete riusciti ad acchiappare. Le istruzioni sono contenute nel programma stesso. Le coordinate dei tre oggetti che cadono contemporaneamente (siano questi cibo o armi) sono memorizzate in array per poterli muovere tramite loop.

```
Graphic A      : linee 1000 e 3000 (doppio)
Graphic B      : linea 4000
Graphic J e K  : linee 2010, 4040 e 5410 (primo set della linea)
Graphic L e M  : linea 2000 (primo set)
Graphic N e O  : Linea 2000 (secondo set) e 5410 (secondo set)
Graphic P      : linea 7010 (primo)
Graphic Q      : linea 7010 (secondo)
Graphic R      : linee 7020 e 7050
Graphic S      : linea 7030
```

Elenco delle variabili

| | |
|-------|---|
| i | Variabile di controllo dei loop per far cadere gli oggetti dal cielo |
| a(.) | Array per le coordinate verticali degli oggetti |
| b(.) | Array per le coordinate orizzontali degli oggetti |
| t2 | Altezza dell' ultimo oggetto mangiato dal mostro. Usato per vedere se il mostro ha raggiunto l' altezza del "Chomper" |
| c(.) | Memorizza il colore di PAPER di ogni oggetto in caduta |
| d(.) | Memorizza il colore di INK di ogni oggetto in caduta |
| c#(.) | Memorizza la forma dell' oggetto (ad esempio una pera) |
| m | Usato per alternare il movimento della bocca de "Chomper" |
| as | Per leggere la tastiera |
| p | Coordinata orizzontale del giocatore |
| gun | Numero dei fucili recuperati |
| rnd | Usata per selezionare la specie di cibo da far cadere |
| n | Controllo per il loop di lettura dei colori del cibo |
| t1 | Usata per ingrassare il mostro quando questo ha mangiato |
| z | Variabile di controllo per la cancellazione dei mostri quando siano stati colpiti |
| s | Punti |
| i, | |
| n, | |
| z | Variabili di controllo per gli altri loop |

108

e nell'oceano. Anche questo programma usa gli array per muovere indipendentemente le gocce, ma pur avendo diverse tecniche in comune con il precedente, fornisce un risultato del tutto diverso e sicuramente migliore. Il miglior risultato che io ricordi e' 48, valore che con un po' di pratica riuscirete sicuramente a battere.

Gli UDG del programma si trovano alle linee : 70-lettera A, 1000-lettera H e 8000-due lettere I.

Elenco delle variabili

| | |
|------|---|
| i | Controllo del loop per muovere le gocce |
| a(.) | Coordinata verticale della goccia |
| b(.) | Coordinata orizzontale della goccia |
| temp | Memorizzazione temporanea di b(.) per cancellare la vecchia goccia dopo che sia stata inizializzata la sua sostituzione. Da' l'impressione della continuita' |
| s | Punti |
| a\$ | Usato per leggere la tastiera |
| l | Marker per spostare i giocatori a destra o a sinistra |
| m | Si alterna tra 0 e 1 facendo camminare gli uomini |
| p | Coordinata orizzontale del primo uomo |
| t\$ | Stringa composta da due elementi per memorizzare le due possibilita' di orientare verso destra o verso sinistra la meta' superiore dell'omino, "l" lo portera' poi nella giusta direzione |
| b\$ | Simile a t\$, riguarda la parte inferiore dell'omino. La direzione la stabilisce "m" |
| wl | Livello dell'acqua |
| h | Record |
| i,n | Variabili di controllo dei vari loop |

```

10 GO TO 5000
20 FOR i=1 TO 3
30 IF a(i)>0 THEN PRINT AT a(i)
  ,b(i);";
40 LET a(i)=a(i)+1
50 IF a(i)=15 AND (b(i)=p OR b
(i)=p+10) THEN LET temp=b(i); GO
SUB 1000; PRINT AT 15,temp;";
60 IF a(i)=10 THEN LET temp=b(
i); LET s=s-1; GO SUB 1000; GO 3
UB 2000; PRINT AT 17,temp;";
70 IF a(i)>0 THEN PRINT INK 5;
AT a(i),b(i);"A"
80 NEXT i
90 LET a$=INKEY$; IF a$="5" OR
a$="8" THEN PRINT AT 15,p;";T
10 p=p+10;";TAB p;";TAB p+10;";
11 LET l=(a$="8"); LET m=m-m+(N
OT m)

```



```

10: GO TO 8050
20: "Nuovo record"
30: "Vuoi giocare"
40: (1930 AND INKEY#)
50: THEN GO TO 80
60: "Vuoi le istru"
70: (3085 AND INKEY#)
80: INKEY#="D")
90: Use graphics
100: FOR i=1 TO 9:
110: DO a: POKE USR:
120: i<4 THEN LET t$(
130: i<8 THEN LET b$(
140: NEXT i
150: 0,10,20,14,5,15,15,12,15,
160: 0,74,137,3,3,43,72,50
170: 0,0,12,12,12,34,35,1
180: 0,0,12,12,12,11,9,24
190: 34,0,40,40,40,42,36,4
200: DATA 34,60,48,48,48,208,144
210: DATA 0,0,0,195,35,153,102,2
220: DATA 0,0,0,24,155,153,90,60
230: Seti motivo
240: 0,0,0,0,12,0,-1,5,0
250: 0,0,0,0,12,0,0,0,0
260: 0,0,0,0,12,0,0,0,0
270: 0,0,0,0,12,0,0,0,0
280: 0,0,0,0,12,0,0,0,0
290: 0,0,0,0,12,0,0,0,0
300: 0,0,0,0,12,0,0,0,0
310: 0,0,0,0,12,0,0,0,0
320: 0,0,0,0,12,0,0,0,0
330: 0,0,0,0,12,0,0,0,0
340: 0,0,0,0,12,0,0,0,0
350: 0,0,0,0,12,0,0,0,0
360: 0,0,0,0,12,0,0,0,0
370: 0,0,0,0,12,0,0,0,0
380: 0,0,0,0,12,0,0,0,0
390: 0,0,0,0,12,0,0,0,0
400: 0,0,0,0,12,0,0,0,0
410: 0,0,0,0,12,0,0,0,0
420: 0,0,0,0,12,0,0,0,0
430: 0,0,0,0,12,0,0,0,0
440: 0,0,0,0,12,0,0,0,0
450: 0,0,0,0,12,0,0,0,0
460: 0,0,0,0,12,0,0,0,0
470: 0,0,0,0,12,0,0,0,0
480: 0,0,0,0,12,0,0,0,0
490: 0,0,0,0,12,0,0,0,0
500: 0,0,0,0,12,0,0,0,0
510: 0,0,0,0,12,0,0,0,0
520: 0,0,0,0,12,0,0,0,0
530: 0,0,0,0,12,0,0,0,0
540: 0,0,0,0,12,0,0,0,0
550: 0,0,0,0,12,0,0,0,0
560: 0,0,0,0,12,0,0,0,0
570: 0,0,0,0,12,0,0,0,0
580: 0,0,0,0,12,0,0,0,0
590: 0,0,0,0,12,0,0,0,0
600: 0,0,0,0,12,0,0,0,0
610: 0,0,0,0,12,0,0,0,0
620: 0,0,0,0,12,0,0,0,0
630: 0,0,0,0,12,0,0,0,0
640: 0,0,0,0,12,0,0,0,0
650: 0,0,0,0,12,0,0,0,0
660: 0,0,0,0,12,0,0,0,0
670: 0,0,0,0,12,0,0,0,0
680: 0,0,0,0,12,0,0,0,0
690: 0,0,0,0,12,0,0,0,0
700: 0,0,0,0,12,0,0,0,0
710: 0,0,0,0,12,0,0,0,0
720: 0,0,0,0,12,0,0,0,0
730: 0,0,0,0,12,0,0,0,0
740: 0,0,0,0,12,0,0,0,0
750: 0,0,0,0,12,0,0,0,0
760: 0,0,0,0,12,0,0,0,0
770: 0,0,0,0,12,0,0,0,0
780: 0,0,0,0,12,0,0,0,0
790: 0,0,0,0,12,0,0,0,0
800: 0,0,0,0,12,0,0,0,0
810: 0,0,0,0,12,0,0,0,0
820: 0,0,0,0,12,0,0,0,0
830: 0,0,0,0,12,0,0,0,0
840: 0,0,0,0,12,0,0,0,0
850: 0,0,0,0,12,0,0,0,0
860: 0,0,0,0,12,0,0,0,0
870: 0,0,0,0,12,0,0,0,0
880: 0,0,0,0,12,0,0,0,0
890: 0,0,0,0,12,0,0,0,0
900: 0,0,0,0,12,0,0,0,0
910: 0,0,0,0,12,0,0,0,0
920: 0,0,0,0,12,0,0,0,0
930: 0,0,0,0,12,0,0,0,0
940: 0,0,0,0,12,0,0,0,0
950: 0,0,0,0,12,0,0,0,0
960: 0,0,0,0,12,0,0,0,0
970: 0,0,0,0,12,0,0,0,0
980: 0,0,0,0,12,0,0,0,0
990: 0,0,0,0,12,0,0,0,0
1000: 0,0,0,0,12,0,0,0,0

```

Note

| | |
|-------|--|
| 10 | Inizializza |
| 20-80 | Loop per muovere le gocce e per verificare |
| 30 | Cancella la goccia |
| 50 | Rileva se ha colpito l' omino |
| 60 | Rileva se ha colpito il suolo |
| 70 | Disegna la goccia |
| 90 | Fornisce l' input e cancella la vecchia sagoma dell' omino |

| | |
|-----------|---|
| 100 | Cambia la posizione |
| 110 | Ristampa l' uomo |
| 1000 | Stampa lo splash, aggiorna il punteggio e inizializza una nuova goccia |
| 2000 | Disegna il nuovo livello |
| 2010 | Incrementa il livello dell' acqua e verifica se e' ad altezza d' uomo |
| 5005-5030 | Istruzioni |
| 5040-5050 | Tutto pronto |
| 5060 | UDG |
| 5070 | Inizializza le variabili |
| 5080 | Inizializza le gocce negli array |
| 5090-5110 | Disegna lo schermo |
| 8000 | Disegna l' omino carico e suona il motivetto |
| 8010-8090 | Bit finale (punti, nuova gara ecc...) |
| 9000 | Inizializza le variabili stringa per l' omino |
| 9010-9040 | Definisce gli user-graphic e li pone nelle variabili stringa |
| 9050-9130 | Dati per gli UDG |
| 9990-9999 | Fine del game |

COME RENDERE PIU' INTERESSANTI I GAMES

L' attrattiva di un game molto dipende dal modo in cui viene presentato. Non cadete nella prolissita', eliminate le frange inutili senza fare piu' di quanto il listato non richieda. Badate anche a non introdurre troppe presentazioni o infioriture che, se da un lato migliorano l' aspetto del display, dall' altro rendono piu' lento il programma.

Eccovi alcuni preziosi suggerimenti :

- Stendete le istruzioni con una nota di humor rendendole piu' divertenti e fuori dalla norma apportando qua' e la' piccoli ritocchi tanto da far capire che i games servono a divertire e non vanno presi troppo seriamente.

- In casi particolari, il computer deve dare l' impressione di commentare l' evento. Ad esempio, in un game tipo "Mastermind", potete inserire prima e dopo ogni mossa, frasi del genere : "Siamo duri eh?..." seguito da una pausa e poi "finalmente ci sei arrivato!". Anche qui pero', occhio a non esagerare perche' cadreste nel noioso.

- Per dare forma agli oggetti usate gli UDG e non simboli come l' asterisco, il punto esclamativo o simili che non rendono assolutamente alcuna idea. La gente non ricorda se la bomba e' grossa la meta' dell' aereo che la sgancia, bensì nota il realismo delle scene e del succedersi delle azioni.

- Sperimentate diversi colori usando le piu' svariate combinazioni, e magari randomizzando i colori di PAPER e di INK in funzione di certi eventi così come avviene nel programma ICBM.

- Generate spesso le esplosioni, sono cariche di effetto. State pero' attenti che all' atto dell' esplosione, il resto del programma rimane inattivo per cui un abuso in questo senso porterebbe a interruzioni troppo frequenti che ostacolerebbero l' effetto prefissato.

- Dovendo inserire effetti musicali, non affidatevi a frequenze di note random, ma ricorrete a filastrocche piacevoli come spiegato nel contesto del volume.

Fate attenzione alla velocita' con la quale il programma gira. E' vero che piu' una azione e' veloce e meglio e', ma anche qui non bisogna esagerare sconfinando nell' impossibile. Se ritenete troppo rapido lo svolgimento di determinate fasi, rallentatelo frapponendo piacevoli effetti musicali. Ovviamente la velocita' di un game dipende da piu' fattori quali ad esempio : le tecniche usate, ivi compresa la compattezza del programma in codice, l' organizzazione dei componenti e quella delle variabili. Un programma con "x" linee comprendenti ognuna uno statement gira piu' lentamente di "x" statement raggruppati nella stessa linea.

Il componente piu' importante di un game e' il loop principale, seguito dalle subroutines, prima quelle piu' usate poi quelle meno (come le istruzioni). Quando il programma incontra GO TO o

GO SUB, esplora in successione i numeri delle linee fino a che non raggiunge quella richiesta. Poiche' il computer parte dall'inizio del listato, piu' vicina si trova la linea alla quale va eseguito il salto e piu' rapidamente questo avviene. Stesso discorso vale quando il computer consulta una variabile. Quelle insite nel primo blocco considerato vengono inizializzate per prime e quindi rintracciate piu' velocemente.

Il BASIC viene spesso criticato per essere un linguaggio "non strutturato" a causa appunto del GO TO che, se usato impropriamente, dirotta fuori destinazione il flusso del programma. E' possibile a tale scopo, strutturare i programmi in modo da far loro assumere le caratteristiche del PASCAL il quale impiega dei loop formati da GO SUB. Questo procedimento si basa su di un "sottoprogramma", richiamato dal loop principale, in grado di fare le operazioni piu' svariate come causare un' esplosione, far sparare un laser, oppure muovere gli invaders.

Provate a dare un' occhiata a quanto segue, e tenetene presente come intelaiatura di games :

```

10 GO SUB 5000: REM inizializz
20 REM LOOP PRINCIPALE
30 GO SUB GET INPUT
40 GO SUB MOVE BASE
50 GO SUB MOVE INVADERS
60 GO SUB CHECK END: REM IF en
70 THEN END
80 GO SUB FIRE
90 GO SUB CHECK HIT
100 IF NOT HIT THEN GO TO 20
110 REM PRINCIPAL LOOP
120 GO SUB EXPLODE
130 STOP
140 REM *** INPUT ***
150 RETURN
160 REM *** MOVIMENTO BASE ***
170 RETURN
180 REM *** MOVIMENTO INVADERS ***
190 RETURN
200 REM *** FINE ***
210 REM IF [hit] THEN end=1
220 RETURN
230 REM *** FUOCO ***
240 RETURN
250 REM *** RILEVAMENTO URTI ***
260 RETURN
270 REM *** ESPLOSIONE ***
280 RETURN
290 REM *** INIZIALIZZAZIONE ***

```

```


000000 REM istruzioni
000010:
000020 REM inizializzazione
000030 LET end=0
000040:
000050 LET GET INPUT=2000
000060 LET MOVE BASE=5000
000070 LET MOVE INCRAD=1000
000080 LET CHECK END=1500
000090 LET CHECK=1000
000100 LET CHECK HIT=2500
000110 LET PLOCK=3000
000120 RETURN

```

Come potete constatare, questo metodo e' chiarissimo ma, porta a programmi di una certa lunghezza. Suo grosso vantaggio, e' quello della tecnica modulare in base alla quale ogni blocco puo' essere registrato separatamente e aggiunto ad altri tramite MERGE.

La "standardizzazione" delle subroutines e delle variabili e' talvolta indispensabile per rendere piu' comoda la stesura del listing. Io, di solito, uso "i" e "n" per il controllo dei loop, "s" oppure "x" per determinare i punti e cosi' via. La ricorrenza di questi simboli mi permette di scegliere appropriatamente il nome delle altre variabili.

Ed eccoci giunti al termine. Con l'augurio che questo libro vi abbia fornito le informazioni indispensabili a realizzare games sempre piu' divertenti, mi congedo augurandovi "buona programmazione"!



Il volume descrive dettagliatamente tutte le tecniche di stesura di giochi ARCADE, partendo dalla lettura della tastiera e toccando la definizione grafica, l'impiego del suono e l'uso degli operatori logici, per migliorare la qualità dei programmi. Altri argomenti esaminati in dettaglio sono l'animazione degli oggetti, lo scrolling dello schermo e l'impiego dei comandi PEEK e POKE per il loro uso più corretto.

Il tutto è accompagnato da 18 programmi la maggior parte dei quali o sono inediti, oppure riguardano versioni migliorate di games di grande successo come "INVADERS" e "BOMBER". I più interessanti sono stati registrati sulla **cassetta allegata al volume** al fine di farvi risparmiare ore di digitazione.

Cod. 9003

L. 25.000
prezzo comprensivo di cassetta

