GIOCARE CON IL BASIC

SYBEX

Richard Mateosian

GRUPPO EDITORIALE JACKSON

EDIZIONE ITALIANA

GIOCARE CON IL BASIC

di
Richard Mateosian

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

Il Gruppo Editoriale Jackson ringrazia per il prezioso lavoro svolto nella stesura dell'edizione italiana la signora Francesca di Fiore e l'ing. Roberto Pancaldi

Prima edizione - Dicembre 1982

Stampato in Italia da:
S.p.A. Alberto Matarelli Milano - Stabilimento Grafico

Fotocomposizione: CorpoNove Bergamo, tel. (035) 22.33.63-22.33.65

PREFAZIONE

I temi principali di questo libro derivano da tre fonti diverse e sono:

- Apprendimento attraverso il gioco
- Programmazione sistematica con uso del Free BASIC
- Giochi che usano il linguaggio BASIC.

Il primo tema è stato il risultato diretto dell'idea partita da Rodnay Zaks che io scrivа un libro sui giochi per computer in BASIC. "Non solo una raccolta di giochi, ma un libro educativo che aiuti il lettore a progettare di programmi in BASIC". Qualsiasi lettore che usi il mio libro sarà certamente in grado di progettare dei programmi in BASIC sia per giochi che per altri usi.

Il secondo viene dalla mia esperienza personale acquisita nel 1975 con lo sviluppo di diverse applicazioni a media scala su un DIC Microfile — uno dei primi sistemi simile a livello funzionale al sistema a dischi TRS-80. A quel tempo ho sviluppato diverse idee che troverete in questo libro, compreso un antenato del Free BASIC.

Le fonti da cui ho tratto i giochi sono varie. Ho scritto tutti i programmi da solo, partendo da zero. I giochi che ho inventato per questo libro sono I Giochi Aritmetici, L'Orologio, Le Carte, Ten-Key Flicker, Compleanno, Calendario, il Gioco degli Accoppiamenti e gli Incontri Spaziali. Gli altri sono tutti giochi da computer ben noti, ma la loro provenienza è sconosciuta.

Redigere un libro non è cosa facile, è processo lungo e faticoso, ma che certamente riempie di soddisfazione. Uno dei piaceri più grandi è quello di poter ringraziare ufficialmente tutti coloro che hanno contribuito alla sua stesura finale.

Prima di tutto vorrei esprimere la mia gratitudine all'Editore Capo della Sybex — Rudolph Langer — e al Direttore alla Produzione — Roger Gottlieb — per lo standard che hanno saputo mantenere.

Inoltre, vorrei ringraziare gli editori della Sybex — Julie Sickert, Sally Oberlin e Doug Hergert — per aver contribuito a rendere l'edizione più scorrevole e per aver saputo fare un libro di un manoscritto un po' rozzo.

La parte "meccanica" del lavoro è passata per diverse mani: Barbara Ellis ha battuto a macchina il manoscritto originale, Janet Rampa e Natalie Levitt ne hanno poi fatto una versione meravigliosa usando un "Word Processor". Mati Sikk si è occupa-

to della composizione tipografica e Jim Compton ne ha corretto le bozze.

J. Trujillo Smith è stato responsabile delle illustrazioni e della presentazione, e certamente questo libro ha beneficiato della sua abilità ed attenzione ai dettagli. John è stato abilmente assistito da Jeanne Tennant.

Leslie Bouffard e Chris Chambers sono responsabili per il lavoro promozionale e per diversi suggerimenti che hanno contribuito a rendere questo libro più rispondente alle necessità ed interessi del lettore.

Ed infine un ringraziamento particolare va a mia moglie, Virginia Ruth Mason. Non mi sarebbe certamente stato possibile scrivere questo libro senza il suo aiuto, la sua infinita pazienza, i suoi sacrifici e la sua comprensione.

Berkelei, California
Febbraio 1981

SOMMARIO

ILLUSTRAZIONI

INTRODUZIONE

Questo libro è stato scritto per tutti quelli che vogliono capire i giochi con il computer e vogliono imparare come scrivere dei programmi interattivi in BASIC.

Imparerete:

- Come si costruiscono dei programmi interattivi, e come si applicano i principii di sviluppo dei sistemi a piccoli computer.
- Come abbiamo sostenuto le caratteristiche di certi sistemi di computer in BASIC.
- Come il Free BASIC vi può aiutare ad usare un approccio più sistematico nella programmazione in BASIC.

Abbiamo usato dei giochi come esempi di programmazione per diversi motivi:

- I giochi si rivelano uno degli interessi principali delle persone che usano un piccolo computer.
- Per capirli non è necessario avere un'esperienza particolare nel campo.
- Nei giochi troviamo la maggior parte delle situazioni di programmazione che sono importanti in altre applicazioni interattive.

La maggior parte dei lettori di questo libro farà parte delle seguenti categorie:

- Coloro che hanno programmato in passato soltanto su piccoli computers.
- Coloro che hanno lavorato come programmatori su microcomputers o su sistemi ad elaboratore centrale, e conoscono almento un linguaggio avanzato (e forse anche uno di gruppo).

Se non avete mai scritto un programma prima di aver acquistato il vostro piccolo computer, probabilmente avrete consultato il manuale che lo accompagna, e avete provato e riprovato gli esempi che questo contiene. Dopo avrete acquistato dei programmi e dei libri di giochi in BASIC. Forse avrete persino letto un libro sulla programmazione in BASIC.

Questo libro vi darà le basi per poter comprendere i computers, il BASIC e il vostro computer. Inoltre vi:

- Guiderà attraverso dei veri programmi in BASIC.
- Darà diverse descrizioni di strutture dei programmi, codifica delle informazioni e algoritmi che sono usati normalmente durante la programmazione.
- Spiegherà in dettaglio la tecnica usata per la progettazione di un programma in BASIC, in modo da aiutarvi a superare le difficoltà che si presentano quando usate il BASIC.

Se siete un programmatore esperto, saprete già come funziona un computer, e a che cosa servono i diversi linguaggi. Questo vi mostrerà i punti deboli e i vantaggi del vostro nuovo sistema. Vi mostreremo:

- Le caratteristiche del BASIC e del vostro computer.
- Un utilissimo attrezzo di progettazione dei programmi chiamato Free BASIC.
- Degli esempi dei diversi tipi di programmazione interattiva che sono tipici dei piccoli sistemi.

Uno schema consistente viene seguito nella presentazione dei giochi in tutto il libro. Prima di tutto si introduce il gioco descrivendone le regole, compreso un esempio di dialogo tra il giocatore ed il programma. Poi segue una descrizione del programma usato. Questa discussione si accentrerà sui principii e le tecniche usate nella programmazione in BASIC.

CONTENUTO

- **I Giochi Aritmetici** sono giochi educativi molto semplici. L'Addizione viene poi generalizzata, uno dei concetti più importanti in programmazione, e trasformata in Gioco Aritmetico. Nonostante il programma che presentiamo in questo capitolo sia estremamente semplice, la maggior parte dei nostri lettori lo troverà interessante.
- **Giochi d'intuizione** ci presenta una famiglia di giochi d'intuizione di numeri chiamati Uno, Due, Tre, Quattro... Nove e due giochi d'intuizione di parole, Parola e l'Impiccato. Anche qui si usa la generalizzazione per progettare un programma unico per entrambi i gruppi di giochi. Un semplice processo di "cannibalizzazione" trasforma il programma in quello dell'Impiccato. La discussione di questi giochi comprende anche un esempio della strategia usata da un buon giocatore di Quattro. Questo capitolo va letto attentamente.
- **Giochi di Tempo** discute le diverse tecniche che sono state usate per manipolare

l'elemento tempo in diversi programmi. I quattro che qui presentiamo sono chiamati l'Orologio, Le Carte, Ten-Key Flicker, e il Cronometro. La strutturazione del programma e l'uso di "matrici" nello sviluppo "sottosopra" vengono spiegate nel capitolo, come pure la sincronizzazione degli eventi con il meccanismo dell'orologio, l'uso del tempo "Julian", e come generare dei ritardi con durata fissa.

— **Giochi con le Date** contiene i due giochi Compleanno e Calendario. Nel primo dimostriamo come determinare il giorno in cui cade una certa data. In Calendario invece viene mostrata una figura che rappresenta il calendario di un certo mese in un certo anno. Viene sottolineata l'importanza del posizionamento del cursore.

— **L'Esattore delle Tasse** presenta un gioco che vi sfida a scoprirne le regole. (Una volta scoperte, però il gioco rimane sempre interessante). Il suo programma si adatta automaticamente alle diverse idiosincrasie della rappresentazione delle stringhe di numeri nei diversi sistemi in BASIC. Questo capitolo si dimostrerà particolarmente interessante per quelle persone che hanno una mente matematica.

— **Programmazione in Free BASIC** presenta una tecnica che è utile nella progettazione di programmi in BASIC. Il Free BASIC è un BASIC "strutturato" che viene tradotto a mano nei diversi comandi in BASIC da inserire nel computer. Il Free BASIC non è un linguaggio, è un mezzo di descrizione di un programma (come la schematizzazione delle operazioni (flowcharts)). Una descrizione di un programma in Free BASIC non ha numeri di linea, e usa dei nomi simbolici per le subroutines. I comandi in GOTO vengono evitati (non ne appare nessuno nelle descrizioni in Free BASIC in questo libro) e vengono usate delle strutture di controllo come "if...then...else", "repeat...until" e "while" al loro posto.

Tutti i programmi illustrati nei primi cinque capitoli vengono spiegati in termini dei loro comandi in BASIC, e non è necessario conoscere il Free BASIC per capirli. Il programmatore esperto dovrà leggere il capitolo che si occupa della Programmazione in Free BASIC per primo, in modo da poter usare le descrizioni in Free BASIC che compaiono nei primi cinque capitoli.

I programmi compresi negli ultimi tre capitoli vengono discussi interamente in termini delle loro descrizioni in Free BASIC. Tuttavia, ci sono anche i comandi in BASIC. Questi possono essere usati per inserire i programmi a mano nel vostro computer. Nel caso dei programmi illustrati nei Capitoli 7 e 9, il BASIC ha lo stesso formato (senza spazi vuoti) che l'autore ha dovuto creare per usarli sul suo Pet 8K.

— **Il Gioco degli Accoppiamenti** viene discusso nei termini della sua descrizione in Free BASIC, senza fare nessun riferimento ai comandi in BASIC. Questo gioco si basa sull'accoppiamento di diversi membri di due gruppi. I giocatori possono aggiustare il gioco a secondo delle diverse necessità. Il programma che è stato usato per il gioco è vasto, e contiene più di quaranta subroutines e più di 500 linee di descrizione in Free BASIC. Se riuscite ad assimilare e capire tutte le sottigliezze del materiale presentato in questo capitolo, allora siete pronti per progettare ed usare dei sistemi interattivi in BASIC.

— **Craps** è più semplice del Gioco degli Accoppiamenti, ma contiene degli elementi diversi: il controllo dell'immagine sullo schermo, la scelta affidata al caso, lo scambio di contesto e la sicurezza di un gioco scorrevole e veloce. Se studiate questo capitolo e ne applicate i principii, i giochi che sarete in grado di progettare saranno certamente divertenti ed interessanti.

— **Esseri Spaziali** introduce il gioco degli Incontri Spaziali, e comprende il Gioco della Vita. I programmi in questo capitolo vi consentiranno di avere una conoscenza più profonda delle tecniche necessarie per l'uso di displays grafici sul vostro computer.

A QUALI COMPUTER SI RIFERISCE QUESTO LIBRO?

I programmi in questo libro sono progettati per essere usati su qualsiasi sistema di computer. Le sezioni dei programmi che devono cambiare quando il programma viene usato su di un altro sistema sono minime e sono state ben identificate. Ciascun programma è stato sviluppato per poter funzionare su una versione di questi computers:

— Il sistema Radio Shack TRS-80, usando BASIC Level II

— Il Commodore Business Machines Pet

— Il Computer Apple Modello II, usando BASIC Applesoft.

CAPITOLO 1

GIOCHI ARITMETICI

Questo capitolo descrive due semplici giochi aritmetici, l'Addizione e le Quattro Operazioni. Il primo gioco è molto semplice, così da darvi la possibilità di familiarizzarvi con le convenzioni che sono state usate per presentare i giochi in BASIC in questo libro. Il secondo gioco differisce solo leggermente dal primo, ma è certamente molto più interessante e stimolante. I diversi passaggi usati per sviluppare il secondo programma usando il primo come punto di partenza spiegano il principio di generalizzazione.

L'ADDIZIONE

Questo gioco viene giocato come segue. Il programma comincia col chiedere:

```
WHAT IS 1 + 8?
```

Voi inserite una risposta e poi premete il tasto RETURN (o ENTER). Visto che la somma di 1 più 8 è 9, schiacciate il tasto del 9 e poi quello di RETURN. Il programma risponderà

```
THAT'S RIGHT — NOW TRY ANOTHER ONE
WHAT IS 3 + 4?
```

Ogni volta che inserite la risposta giusta, il programma vi porrà un'altra domanda. Ma, poniamo il caso che vi sbagliate, e invece di inserire 7, seguito da RETURN, schiacciate il tasto dell'8. Il programma risponderà

```
I'M SORRY, THAT'S WRONG — TRY AGAIN
WHAT IS 3 + 4?
```

Il senso del gioco è questo: ogni volta che inserite una risposta esatta, il program-

ma vi porrà un'altra domanda, ogni volta che sbagliate la risposta, il programma ripeterà la domanda finchè non otterrà la risposta esatta.

IL PROGRAMMA DELL'ADDIZIONE

Nella figura 1.1 vediamo una schematizzazione delle operazioni di questo gioco. Per leggerla dovremo partire dal circoletto "START" e seguire le frecce. Quando arrivate alla casella a forma di rombo che ha due frecce, seguirete quella del NO se la risposta è errata, e quella del SÌ se la risposta è esatta. Ciascuna di queste due strade vi riporterà ad una parte del diagramma che avete già passato. Un segmento della schematizzazione attraverso il quale siete costretti a passare e ripassare si chiama loop.

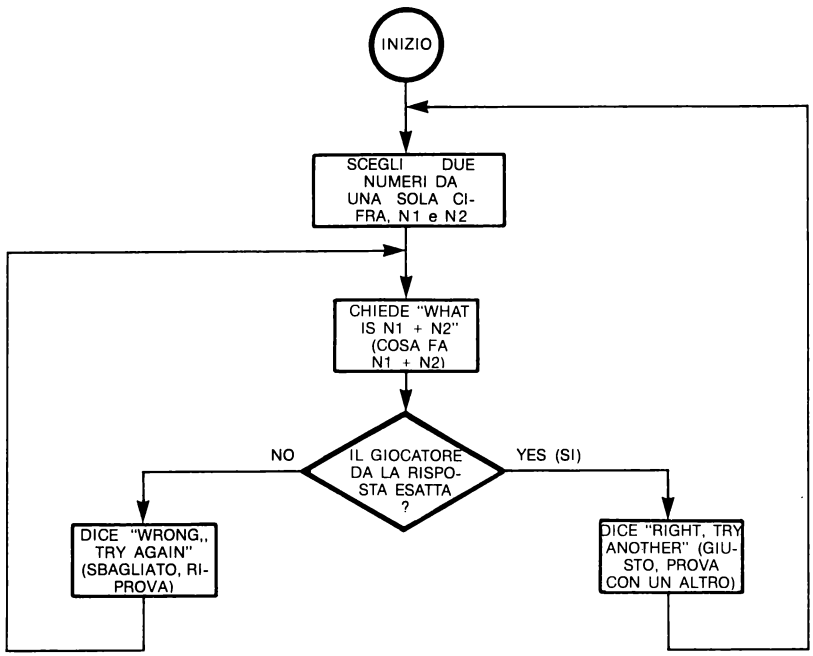


Figura 1.1 — Schematizzazione delle operazioni del gioco dell'Addizione.

Nella Figura 1.2 vediamo un gruppo di comandi in BASIC che fanno sì che il computer agisca proprio come abbiamo appena descritto. Questi comandi corrispondono esattamente alle caselle nella schematizzazione. Le linee numerate 100 e 110 corri-

spondono alla prima casella. Quelle numerate 120 e 130 alla seconda, mentre la linea numerata 140 dà il formato all'output che non è illustrato nella schematizzazione. La linea 150 corrisponde al rombo e al "ramo" del NO, mentre le linee 160 e 170 corrispondono a quello del SÌ.

Esaminiamo ogni passaggio che il programma fa nella Figura 1.2. L'azione inizia alla linea 100, perchè il BASIC comincia sempre con la linea con il numero minore, e prosegue nella lettura delle linee del programma in ordine numerico, a meno che vi sia un comando specifico (come GOTO 120) che cambi questo ordine.

```
100*  N1=INT(RND(1)*10)
110*  N2=INT(RND(1)*10)
120   PRINT 'WHAT IS"; N1;"+"; N2;
130   INPUT A
140   PRINT
150   IF A<>N1+N2 THEN PRINT "I'M SORRY, THAT'S WRONG—TRY A-
      GAIN":GOTO 120
160   PRINT "THAT'S RIGHT — NOW TRY ANOTHER ONE"
170   GOTO 100
```

Questo programma sceglie a caso dei problemi aritmetici con una sola cifra e li presenta al giocatore. Se riceve una risposta esatta, il programma automaticamente passa ad un altro problema. Se invece riceve una risposta sbagliata, il programma ripete la domanda finché il giocatore non inserisce la risposta esatta.

Il programma di cui sopra va bene sia per i sistemi in BASIC del Pet e del TRS-80. Per l'Apple, il numero della linea 120 deve essere cambiato come segue:

```
120 PRINT"WHAT IS";N1;" + ";N2;" ";
```

La differenza nasce dal fatto che il BASIC nell'Apple non fa precedere né seguire i numeri da spazi, mentre nel Pet e nel TRS-80 tutti i numeri positivi vengono preceduti da uno spazio e tutti i numeri vengono seguiti da uno spazio.

---

* Nella versione del TRS-80 il termine RND(0) appare al posto di RND(1). Nel TRS-80, qualsiasi discussione che non sia zero ha un significato speciale. Nel Pet e nell'Apple, zero ha un significato speciale.

Figura 1.2 — Gioco dell'Addizione.

Generazione dei numeri affidata al caso

In questo programma la linea 100 contiene un comando:

```
N1 = INT(RND(1)*10)
```

In genere, il comando sulla linea 100 significa: "Scegliere un numero intero da zero a nove e conservarlo nella casella chiamata N1". Questo numero scelto è il primo di due che il programma vi chiederà di sommare via via che il gioco va avanti.

Nella linea 110 vi è il comando

```
N2 = INT(RND(1)*10)
```

Questo comando è uguale al primo, a parte il fatto che N1 alla sinistra dell'uguale è stato cambiato in N2. La linea 110 dice: "Scegliete un altro numero intero da zero a nove e conservarlo nella casella chiamata N2".

Anche se l'espressione alla destra dell'uguale è la stessa per entrambi i comandi, i numeri che vengono conservati nelle caselle N1 e N2 non sono gli stessi. Questa variazione è dovuta alla presenza del termine RND (1) nel comando. RND vuol dire funzione affidata al caso. Quando si esegue il comando, il termine RND (1) viene sostituito con un numero "imprevedibile" tra zero e nove (ma che non sia mai nè zero nè uno).

Per questo gioco, abbiamo bisogno di numeri interi maggiori o uguali a zero e minori di dieci. La gamma dei numeri dati da RND può essere ampliata dallo zero a uno originale a zero a N, semplicemente moltiplicando per N. Dunque, il termine

```
RND(1)*10
```

in entrambi i comandi verrà sostituito con un numero impredicibile tra zero e dieci, (ma che non sia nè zero nè dieci) quando si esegue il comando.

La funzione INT semplicemente "elimina" la parte decimale di un numero. Per esempio, INT (5,63279) è 5, INT (0,22116) è 0, INT (9,99999) è 9. INT è l'abbreviazione di "integer" cioè numero intero.

Dunque, il termine

```
INT(RND(1)*10)
```

viene sempre sostituito con un numero intero scelto a caso da questo gruppo 0,1,2,3,4,5,6,7,8,9. (Capite perchè il numero intero scelto a caso non può essere dieci? E perchè non può essere zero?).

Il comando Print

Il terzo comando, sulla linea 120 è:

```
PRINT"WHAT IS";N1;"=";N2;
```

Questo comando fa sì che il computer scriva "What IS ____ + ____? (a parte il

punto di domanda che viene scritto dal sistema quando il comando INPUT viene eseguito). Sfortunatamente, quello che in effetti compare sullo schermo differisce a seconda del computer usato. Per esempio su di un Pet, l'output che viene generato da questo comando sarà:

```
WHAT IS 9 + 7
```

Sull'Apple invece

```
WHAT IS9+7
```

Questa diversità nella spaziatura nasce dal fatto che il Pet postpone ad ogni numero uno spazio ed antepone ad ogni numero positivo uno spazio. L'Apple non inserisce nessuno spazio nè prima nè dopo un numero (il TRS-80 funziona come il Pet in questo caso).

Vi sono altre differenze tra i diversi comandi PRINT, e ne parleremo via via che le incontriamo. Prima di tutto vediamo come funziona la forma più semplice del comando PRINT. (Questo comando funziona più o meno lo stesso per tutti i sistemi in BASIC). La forma più semplice di questo comando consiste della parola PRINT, seguita da una lista di variabili e di costanti.

Per definizione una costante è un numero, come 5 o 9,763, o una parte di testo che è racchiusa tra virgolette (come "WHAT IS"). Una variabile è una casella, cioè il nome di un posto nella memoria del computer dove abbiamo in precedenza conservato una costante. In linguaggio più corretto, una variabile è anche chiamata "indirizzo simbolico", poichè indica il nome di uno spazio in memoria. Nel comando

```
PRINT "WHAT IS";N1;"+";N2;
```

la lista che segue la parola PRINT è formata dalla costante "WHAT IS", dalla variabile N1, dalla costante "+" e dalla variabile N2. I punti e virgola sono usati per separare le diverse voci nella lista, e per specificare come l'output apparirà sullo schermo.

Il BASIC esegue un comando PRINT passando attraverso la lista di costanti e di variabili e mostrandone i loro valori. In altre parole, ogni volta che si incontra una costante nella lista, BASIC mostra quella parte di testo; quando invece si incontra una variabile, BASIC mostra il numero o la parte di testo che abbiamo conservato nella casella corrispondente. Dunque, se i nostri due primi comandi sono risultati nell'aver conservato un 5 e N1 e un 7 a N2, allora il comando

```
PRINT "WHAT IS ;N1;"+";N2;
```

darebbe luogo sullo schermo a

```
WHAT IS 5 + 7
```

Abbiamo spiegato come fa il BASIC a determinare quali valori vanno mostrati mentre esegue un comando PRINT. Ora discuteremo come fa il BASIC a decidere in quale posizione sullo schermo vanno mostrati questi valori. Nel nostro esempio, i punti e virgola dicono al BASIC di mostrare i valori "dall'inizio alla fine" senza inserire degli altri spazi o andare a capo. Alternativamente potrete usare delle virgole invece dei punti e virgola per separare le diverse voci. I BASIC, dopo aver visualizzato il valore che corrisponde ad una delle voci nella lista che è seguita da una virgola, sposta automaticamente il cursore alla "posizione del tabulatore" seguente. Il tabulatore funziona come quello di una normale macchina da scrivere, ma nel BASIC le posizioni sono predeterminate, e non possono essere controllate dal programmatore.

Il vostro computer avrà tre o quattro posizioni per ciascuna linea da quaranta caratteri. Dunque se avessimo scritto:

```
PRINT "WHAT IS",N1,"+",N2
```

sarebbe apparso sullo schermo

```
WHAT IS 5   +   7
```

o persino

```
WHAT IS 5   +
7
```

Negli esempi precedenti non abbiamo potuto vedere gli effetti del punto e virgola o della virgola a fine frase. Guardiamo il prossimo esempio dimostrato dal nostro programma. Il comando

```
INPUT A
```

sulla linea 130 viene usato per ottenere l'input da parte del giocatore. L'uso del comando di INPUT fa sempre sì che appaia un "?" sullo schermo. Dunque l'effetto delle linee

```
PRINT "WHAT IS";N1;"+";N2;
INPUT A
```

è la visualizzazione

```
WHAT IS7 + 5?
```

Se avessimo dimenticato l'ultimo punto e virgola dal comando PRINT, avremmo a-

vuto una visualizzazione del tipo

```
WHAT IS7 + 5
?
```

In questi esempi abbiamo così visto che ogni volta che un comando PRINT non finisce con un punto e virgola o con una virgola, il BASIC finisce muovendo il cursore all'inizio della linea successiva.

Abbiamo visto ormai tutti gli aspetti della forma più semplice del comando PRINT. Questa è certamente una delle parti più importanti di quasi tutti i programmi in BASIC, così è assolutamente necessario che la capiate a fondo. Ora cercate di indovinare che cosa risulterà dai comandi:

```
PRINT "WHAT IS",N1,"+",N2
INPUT A
```

e provatelo sul vostro computer.

Il comando Input

Ritornando al programma del gioco, il comando

```
INPUT A
```

fa sì che il BASIC scriva "?" (o "?" senza spaziatura nell'Apple) e poi aspetta che voi inseriate un numero e schiacciate il tasto RETURN (o quello ENTER su di TRS-80). Dopo di che, il BASIC controlla che sia stato veramente inserito un numero, altrimenti (se per esempio aveste inserito una X) vi dirà

```
?REENTER
?
```

o un messaggio simile. Poi aspetterà che riproviate.

La dichiarazione IF

Dopo che è stato inserito un numero, BASIC lo conserverà nella casella A e andrà avanti ad eseguire la linea 150, che contiene

```
IS A< >N1+N2 THEN PRINT"I'M SORRY, THAT'S WRONG
—TRY AGAIN":GOTO 120
```

questo è un nuovo tipo di comando — il comando IF. Infatti poichè la frase inizia con

un "IF" (SE) può avere più di un comando, viene dunque chiamata una dichiarazione IF. La prima parte di una dichiarazione IF (in questo caso A< >N1 + N2) è la condizione. (Il segno "< >" significa "non è uguale a"). Se il numero che è conservato nella casella A non è uguale alla somma dei numeri nelle caselle N1 e N2 (cioè se avete inserito una risposta sbagliata) allora BASIC eseguirà la parte della dichiarazione (chiamata l'"azione") che segue il THEN. Un'azione è composta da uno o più comandi separati da punti e virgola. In questo caso i comandi sono

```
PRINT "I'M SORRY, THAT'S WRONG — TRY AGAIN"
```

e

```
GOTO 120
```

È importante che voi capiate come funziona una dichiarazione IF: se la condizione è valida, tutti i comandi sul resto della linea vengono eseguiti; se la condizione non è valida, allora nessuno dei comandi viene eseguito. In entrambi i casi, a meno che ci sia un GOTO tra i comandi che seguono THEN, BASIC passa alla linea che segue quella contenente la dichiarazione IF. Dunque, nella Figura 1.2, se la condizione è valida (cioè se avete dato una risposta errata) il BASIC eseguirà prima di tutto il comando

```
PRINT "I'M SORRY, THAT'S WRONG — TRY AGAIN"
```

e poi

```
GOTO 120
```

Sappiamo già che cosa farà il comando PRINT, dunque passiamo ad esaminare il comando GOTO, che causa l'alterazione della sequenza di esecuzione dei comandi. Invece di proseguire e passare alla linea 160, BASIC deve tornare indietro a quella numero 120 e ricominciare da lì. Dunque, se inserite la risposta sbagliata, la sequenza di stampare la domanda e poi presentarvela sarà ripetuta, e apparirà:

```
WHAT IS 5 + 7 ? 13

I'M SORRY, THAT'S WRONG — TRY AGAIN
WHAT IS 5 + 7 ?
```

(La vostra risposta — 13 — è stampata in neretto).

Se la condizione nella dichiarazione IF sulla linea 150 non è valida (cioè se avete inserito la risposta esatta) allora il resto della linea 150 viene saltato, e BASIC procede alla linea 160. Lì eseguirà il comando

```
PRINT "THAT'S RIGHT — NOW TRY ANOTHER ONE"
```

e va alla linea 170 per eseguire

```
GOTO 100
```

In questo caso, se cioè avete inserito la risposta esatta, apparirà:

```
WHAT IS 7 + 5 ? 12

THAT'S RIGHT — NOW TRY ANOTHER ONE
WHAT IS 9 + 2 ?
```

Il Free Basic

Come abbiamo notato in precedenza, i comandi nella Figura 1.2 seguono la schematizzazione della Figura 1.1. Nella Figura 1.3 vediamo lo stesso programma in Free BASIC. (Il Free BASIC viene descritto nel Capitolo 6). Esaminate la Figura 1.3 per vedere se riuscite a capire questo programma. Notate che il programma in Free BASIC è molto simile a quello illustrato nella Figura 1.2, tuttavia non ha numeri di linea. Inoltre certe parole come "repeat" e "else" compaiono in lettere minuscole nella Figura 1.3; e nella Figura 1.2 non vi sono parole in lettere minuscole.

Il Free BASIC è stato usato dall'autore per progettare i programmi in BASIC che compaiono in questo libro. Nei Capitoli compresi tra il numero 1 e il numero 5, nonostante nella discussione non si faccia riferimento alcuno al Free BASIC, questo è sta-

```
repeat{
  N1=INT(RND(1)*10)
  N2=INT(RND(1)*10)
  repeat{
    PRINT "WHAT IS"; N1;"+";N2
    INPUT A
    IF A = N1 + N2 THEN
      PRINT "THAT'S RIGHT etc"
    else
      PRINT "THAT'S WRONG etc"
    }until (A = N1 + N2)
}
  Questa é una descrizione in Free BASIC del programma del Gioco dell'Addizione.
Il Free BASIC viene spiegato nel Capitolo 6.
```

Figura 1.3 — Il Gioco dell'Addizione in Free BASIC.

to inserito in tutti i programmi e viene illustrato nelle figure. Le descrizioni in Free BASIC sono state inserite per darvi la possibilità di familiarizzarvi con questo tipo di programma, e per capirne meglio la costruzione. Nei Capitoli 7, 8 e 9 ci siamo riferiti soltanto alle descrizioni in Free BASIC e non ai comandi in BASIC.

LE QUATTRO OPERAZIONI

Il programma dell'Addizione è semplicemente un esercizio preparatorio a quello delle Quattro Operazioni. Il programma precedente ci ha dato la possibilità di discutere diversi comandi in BASIC abbastanza importanti, ma come gioco è decisamente troppo semplice e ripetitivo e perde dunque d'interesse. Il gioco delle Quattro Operazioni è simile, ma ha due caratteristiche in più che lo rendono molto più interessante:

- L'uso di numeri a più cifre (il giocatore decide la grandezza del numero).
- Scelta affidata al caso di un'operazione d'addizione, sottrazione, moltiplicazione e divisione.

A parte queste caratteristiche, il gioco delle Quattro Operazioni è molto simile a quello dell'Addizione.

Eccovi un esempio di dialogo tra il programma e il giocatore. Notate che, come in precedenza, le risposte del giocatore sono scritte in neretto.

```
HOW MANY PLACES? 2
WHAT IS 240/16? 15

THAT'S RIGHT — NOW TRY ANOTHER ONE
WHAT IS 8 + 1? 9

THAT'S RIGHT — NOW TRY ANOTHER ONE
WHAT IS 23 x 77? 1781

I'M SORRY, THAT'S WRONG — TRY AGAIN
WHAT IS 23 x 77?
```

IL PROGRAMMA DELLE QUATTRO OPERAZIONI

Nella Figura 1.4 vediamo una schematizzazione del programma per il gioco delle Quattro Operazioni. Nella Figura 1.5 vediamo i comandi in BASIC. Il programma nella Figura 1.5 è molto simile a quello del gioco dell'Addizione illustrato nella Figura

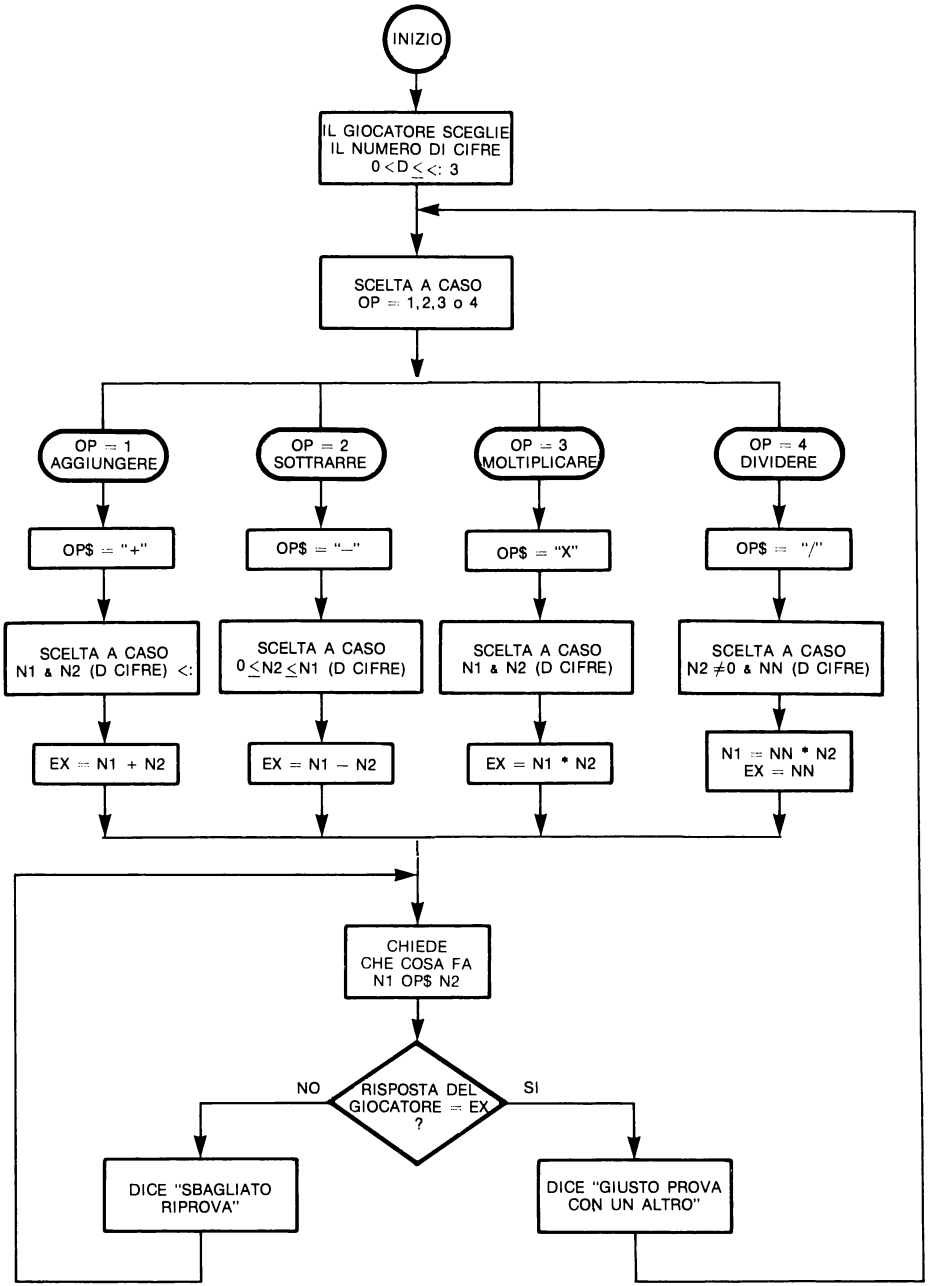


Figura 1.4 — Schematizzazione delle operazioni dei Giochi Aritmetici

1.2. Diverse linee della Figura 1.5 corrispondono quasi esattamente a quelle della Figura 1.2 con piccole differenze. Per esempio:

- La costante "+" sulla linea 130 nella Figura 1.2 è stata sostituita dalla variabile OP$ sulla linea 150 nella Figura 1.5.
- L'espressione N1 + N2 sulla linea 150 nella Figura 1.2 è stata sostituita con la variabile EX sulla linea 170 nella Figura 1.5.
- La formula INT (RND(1)*10) sulle linee 100 e 110 della Figura 1.2 è diventata FNR(D) in tutta la Figura 1.5, dove FNR(D) = INT(RND(1)*10↑D).

Queste modifiche sono chiamate "generalizzazioni". Una generalizzazione è la sostituzione di un elemento con una categoria più vasta, di cui l'elemento originale è un caso speciale. Per esempio, sostituendo a "+" la variabile OP$ è una generalizzazione, perchè "+" è uno dei quattro valori di OP$. Allo stesso modo, sostituendo all'espressione N1 + N2 la variabile EX è una generalizzazione, perchè N1 + N2 è una delle quattro espressioni usate per determinare il valore di EX. (Le altre sono N1 − N2, N1 + N2, e N1/N2). In fine, sostituendo a RND(1)*10 con RND(1)*10↑D è una generalizzazione, perchè 10 (cioè 10↑) è uno dei tre valori possibili presi da 10↑D, poichè D prende i valori 1, 2 e 3.

La generalizzazione è una tecnica estremamente importante per l'allargamento dei programmi per computer. Non appena possibile dovreste progettare il vostro programma in maniera da poterlo poi generalizzare facilmente. Per esempio, avremmo potuto scrivere il programma dell'Addizione (Figura 1.2) leggermente diverso. Avremmo potuto aggiungere due linee all'inizio:

```
80 OP$ = "+"
90 D = 1
```

e poi aver aggiunto una linea dopo la 110

```
115 EX = N1 + N2
```

Così le linee 120 e 150 della Figura 1.2 avrebbero potuto essere identiche alle 150 e 170 della Figura 1.5. Il vantaggio del nostro sistema sta nel poter identificare quegli aspetti del programma che possono essere generalizzati facilmente. Inoltre, l'uso di variabili invece di costanti e la programmazione anticipata di possibili generalizzazioni, sono esempi di buona programmazione. Questo non è stato fatto per la Figura 1.2 perchè altrimenti si sarebbe creata una complessità tale da rendere troppo difficili le spiegazioni del programma più elementare del libro.

Ora diamo un'occhiata alla schematizzazione illustrata nella Figura 1.4. Notate che è praticamente identica a quella nella Figura 1.1, a parte due piccole differenze:

- Il riquadro che contiene "Let player select..." non compare nella Figura 1.1.
- Il riquadro nella Figura 1.1 che contiene le parole "Pick two 1-digit numbers, N1 and N2" è stato sostituito con un solo riquadro nella Figura 1.4 (che contiene le parole "randomly choose OP = 1,2,3,4"), seguito da un gruppo di quattro linee, di cui una soltanto viene fatta passare attraverso il loop.

Ciascuno di questi due punti si occupa della determinazione dei valori delle variabili D, OP$ e EX. Cioè quelle parti della schematizzazione illustrata nella Figura 1.4 che si occupano delle generalizzazioni sono ampliamenti delle parti della schematizzazione illustrata alla Figura 1.1; quelle parti che non si occupano delle generalizzazioni, nella Figura 1.4, sono identiche alle corrispondenti nella Figura 1.1. Ora diamo un'occhiata al programma nella Figura 1.5. Questo illustra delle caratteristiche del BASIC che non abbiamo ancora esaminato. Le prime due linee si occupano della determinazione di un valore per la variabile D. La prima linea è:

```
INPUT "HOW MANY PLACES"; D
```

La costante "HOW MANY PLACES" dà al BASIC la domanda che deve essere posta, mentre la variabile D gli dice dove conservare la risposta. Dunque, questo comando fa sì che il BASIC chieda

```
HOW MANY PLACES?
```

poi aspetterà una risposta numerica, e una volta ottenutala, la conserverà nella casella D. (Questo non è sempre vero, nella Figura 1.5 abbiamo seguito le convenzioni dei computers Pet e TRS-80. Per un Apple, bisognerà scrivere il comando come segue

```
INPUT "HOW MANY PLACES?";D
```

Su entrambi i sistemi del Pet e del TRS-80, il BASIC forma la domanda aggiungendo un "?" alla fine della costante fornitagli nei comandi, mentre nell'Apple non viene aggiunto il "?").

La linea seguente contiene due comandi separati da un segnale di due punti:

```
D = INT (D) : IF D < = 0 OR D > 3 THEN 100
```

All'inizio di questo capitolo abbiamo imparato che la parte di una dichiarazione IF che riguarda l'azione può essere formata da diversi comandi separati da due punti. Ora vediamo che qualsiasi linea di un programma in BASIC può essere composta da diversi comandi separati da due punti. (Ma ricordatevi che qualsiasi comando che segue un "IF" fa parte dell'azione della dichiarazione IF). Conosciamo già INT, che

```
100   INPUT "HOW MANY PLACES";D
110   D=INT(D):IF D<=0 OR D>3 THEN 100
120*  DEF FNR(X)=INT(RND(1)*10↑X)
130*  OP=INT(RND(1)*4)+1
140   ON OP GOSUB 190,220,260,290
150   PRINT "WHAT IS";N1;OP$;N2;
160   INPUT A:PRINT
170   IF A<>EX THEN PRINT "I'M SORRY, THAT'S WRONG — TRY A-
      GAIN": GOTO 150
180   PRINT "THAT'S RIGHT — NOW TRY ANOTHER ONE":GOTO 130
190   OP$="+"
200   N1=FNR(D):N2=FNR(D)
210   EX=N1+N2:RETURN
220   OP$="—"
230   N1=FNR(D):NN=FNR(D):IF NN<=N1 THEN N2=NN:GOTO 250
240   N2=N1:N1=NN
250   EX=N1—N2:RETURN
260   OP$="x"
270   N1=FNR(D):N2=FNR(D)
280   EX=N1*N2:RETURN
290   OP$="/"
300   N2=FNR(D):IF N2=0 THEN 300
310   NN=FNR(D):N1=NN*N2
320   EX=NN:RETURN
```

Questo programma in BASIC rende effettivi i Giochi Aritmetici. Il programma principale consiste di linee numerate da 100 alla 180. Le quattro subroutines occupano le linee 190-210, 220-250, 260-280 e 290-320.

Le linee 100-120 determinano D (il numero delle cifre che compongono i numeri usati nei problemi) e definiscono la funzione FNR che sceglie i numeri usati nei problemi. Le linee 130-140 scelgono una delle quattro operazioni (addizione, sottrazione, moltiplicazione e divisione) da usarsi nel problema e chiamano una delle quattro subroutines per determinare:

- i numeri N1 e N2 da includere nel problema
- il simbolo OP$ che viene inserito tra i due numeri (+, —, x, /)
- la risposta giusta EX.

Le linee 150-180 accettano una risposta, la valutano e/o proseguono con un nuovo problema o ripropongono la domanda.

Per la stessa ragione che abbiamo spiegato nella Figura 1.2, e poichè il BASIC usato nell'Apple non aggiunge un punto di domanda alla stringa costante specificata in una dichiarazione INPUT, il programma di cui sopra viene corretto per il Pet e per il TRS-80. Per un Apple le linee 100 e 150 devono essere cambiate come segue:

```
100   INPUT"HOW MANY PLACES?";D
150   PRINT"WHAT IS";N1;OP$;N2;"";
```

* Nella versione del TRS-80 di questa linea, il termine RND(0) compare invece di RND (Vedere Figura 1.2)

Figura 1.5 — I Giochi Aritmetici

rende D un numero intero eliminando tutti i decimali. Nella dichiarazione IF, il 100 che segue THEN è un'abbreviazione per il comando

```
GOTO 100
```

La linea 120 del programma contiene una definizione di funzione:

```
DEF FNR(X) = INT(RND(1)*10ÎX)
```

La definizione di una funzione è semplicemente un modo per informare il programma che vogliamo accorciare una formula un po' lunga. Il comando di cui sopra stabilisce che FNR(D) è una abbreviazione del termine INT(RND(1)*10ÎD). Questa abbreviazione viene usata sulle linee 200, 230, 270, 300 e 310.

La variabile X che compare ad entrambi i lati del segno di uguale nella definizione di funzione viene chiamata una variabile "falsa". Una variabile falsa non è altro che un'indicazione di dove D andrà inserito nella formula quando è il momento di valutarlo. Per esempio, se scrivessimo FN(2), si inserirebbe un 2 nella formula al posto di X, così che FNR(2) sarebbe interpretata come una abbreviazione del termine INT(RND(1)*10 2).

La prossima linea del nostro programma, la 130, contiene il comando

```
OP = INT(RND(1)*4) + 1
```

Questo comando dà luogo ad una scelta a caso da effettuarsi tra i numeri interi 1, 2, 3, e 4 e il risultato andrà poi conservato nella apposita casella OP. (Se non capite perchè, leggete la spiegazione che è stata fornita in precedenza per spiegare perchè INT(RND(1)*10) assume dei valori scelti a caso tra 0, 1, 2, 3, 4, 5, 6, 7, 8, e 9).

Tra parentesi, tutti gli usi delle funzioni INT e RND in combinata che abbiamo esaminato sono stati fatti solo in modo da raggiungere lo scopo di scegliere un numero a caso fra un gruppo di numeri consecutivi interi (da 0 a 9, da 0 a 99, da 0 a 999, da 1 a 4). Se il BASIC fosse un po' più flessibile, avremmo potuto fare una definizione di funzione che avrebbe contenuto tutti questi casi:

```
DEF FNC(L,H) = INT(RND(1)*(H - L + 1)) + L
```

Allora per esempio,

```
OP + FNC(1,4)
```

sceglierebbe un numero intero da 1 a 4, mentre

```
N1 = FNC(0,9)
```

sceglierebbe un numero intero da 0 a 9. L'espressione FNR(D) sarebbe sostituita

con FNC(0,10ÎD – 1). Sfortunatamente non è possibile dare tale definizione, perchè nel BASIC non sono permesse delle definizioni di funzioni con due variabili false. Quindi, sarà necessario usare INT(RND(1)*(H – L + 1)) + L ogni qualvolta vogliamo rappresentare FNC(L,H).

Ritornando al programma nella Figura 1.5, la linea seguente, la 140, presente un'altra caratteristica che non abbiamo ancora discusso, la dichiarazione ON:

```
ON OP GOSUB 190, 220, 260, 290
```

Ricordate che abbiamo appena determinato che il valore della variabile OP è 1, 2, 3 o 4. Nella dichiarazione ci sono quattro numeri di linee: 190, 220, 260 e 290. Queste sono le linee iniziali di quattro subroutines. Se il valore di OP è 1, viene eseguito il comando GOSUB 190, se il valore di OP è 2, viene eseguito il comando GOSUB 220, se è 3, GOSUB 260 e se è 4 GOSUB 290. Soltanto una delle quattro subroutines viene chiamata (Si avranno delle variazioni a seconda dei diversi sistemi in BASIC se OP viene fissato con un valore che non sia nè 1, nè 2, nè 3 e nè 4).

Lo scopo di OP e delle quattro sue subroutines è quello di far funzionare i quattro «rami» che abbiamo visto nella schematizzazione alla Figura 1.4. Ciascuna subroutine corrisponde ad una delle quattro operazioni: l'addizione, la sottrazione, la moltiplicazione e la divisione. Ogni subroutine fissa anche N1, OP$, N2 e EX; cioè la subroutine decide il primo numero, il simbolo per l'operazione, il secondo numero e il risultato. Discuteremo le subroutines dopo aver finito di analizzare il programma principale. La linea che segue, la 150, contiene il comando

```
PRINT"WHAT IS";N1;OP$;N2;
```

Questa linea è molto simile alla 120 illustrata nella Figura 1.2, ma nel programma nella Figura 1.5, la variabile OP$ sostituisce la costante "+". I due comandi sulla linea 160

```
INPUT A:PRINT
```

sono identici ai comandi sulle linee 130 e 140 nella Figura 1.2. Il comando INPUT accetta la risposta del giocatore. Il comando PRINT è responsabile della spaziatura, così da migliorare la leggibilità del testo.

Le linee 170 e 180 sono analoghe alle 150, 160 e 170 nella Figura 1.2. Le uniche differenze sono EX invece di N1 + N2 e i diversi numeri di linea. Così termina il programma principale.

Le quattro subroutines sono molto simili tra di loro. La prima, che prende le linee 190, 200, e 210 è la più semplice:

```
OP$ = "+"
N1 = FNR(D):N2 = FNR(D)
EX = N1 + N2: RETURN
```

Questa subroutine illustra il modello usato in genere da tutte e quattro le subroutines. Prima di tutto viene deciso il valore di OP$. In questo caso "+" per indicare una addizione. Poi, vengono scelti due numeri e conservati nelle caselle N1 e N2. In fine, la risposta esatta (cioè la somma dei due numeri in questo caso) viene conservata in EX.

La seconda subroutine, che occupa le linee 220, 230, 240 e 250, è un pochino più complicata:

```
      OP$ ="-"
      N1 = FNR(D) : NN = FNR(D):
      IF NN < = N1 THEN N2 = NN: GOTO 250
      N2 = N1:N1 = NN
250   EX = N1 - N2: RETURN
```

Il motivo è che vogliamo che il primo numero sia maggiore o uguale al secondo, così da non poter avere una risposta negativa. Prima di tutto scegliamo N1 e si conserva un secondo numero in NN. Se il secondo numero è minore o uguale al primo numero, viene semplicemente spostato da NN a N2. Ma se il numero a N1 è minore di quello a NN, i due vengono scambiati, e quello a NN va in N1. Quando sia N1 che N2 sono stati scelti correttamente, la risposta esatta è la loro differenza.

La terza subroutine, che occupa le linee 260, 270, e 280, è quasi identica alla prima:

```
OP$ = "x"
N1 = FNR(D): N2 = FNR(D)
EX = N1 × N2: RETURN
```

L'unica differenza è la sostituzione dei simboli dell'addizione con quelli della moltiplica.

La quarta subroutine, che occupa le linee 290, 300, 310, e 320 è la più complicata delle quattro perchè è necessario che:

- il divisore non sia zero
- il risultato sia un numero intero.

Queste necessità si risolvono abbastanza in fretta:

```
      OP$ = ":"
300   N2 = FNR(D): IF N2 = O THEN 300
      NN = FNR(D): N1 = NN*N2
      EX = NN: RETURN
```

La linea 300 ci permette di poter continuare a scegliere un divisore che non sia ze-

ro. Poi, invece di scegliere il primo numero, scegliamo il risultato. Il primo numero viene poi calcolato con il divisore moltiplicato per il risultato.

Così termina la nostra discussione del programma del gioco delle Quattro Operazioni.

```
#Giochi Aritmetici
        DEF fnc(L,H) = INT(RND(1)*(H - L+1) +L     #scelta della funzione
        repeat
          INPUT "HOW MANY PLACES?";D
          until (D = 1 OR D = 2 OR D = 3)
        L =0: U = 10↑D - 1                         #determinazione dei li-
                                                   miti per i numeri
        repeat {                                   #loop principale
          OP = fnc(1,4)                            #scelta di un numero da
                                                   1 a 4
          ON OP GOSUB add, sub, mul, div
          repeat {                                 #la domanda viene po-
                                                   sta finchè la risposta
          }                                        non è esatta
          PRINT "WHAT IS"; N1; OP$; N2;
          INPUT A: PRINT
          IF A < > EX THEN PRINT "THAT'S WRONG etc"
            until (A = EX)
          PRINT "THAT'S RIGHT etc"

add     OP$ = "+"
        N1 = fnc(L,U): N2 = fnc(L,U)               #scegliere due numeri
        EX = N1 + N2: RETURN                       #ottenerne la somma

sub     OP$ = "–"
        N1 = fnc(L,U): N2 = fnc(L,U)               #scegliere due numeri
        IF N1 < N2 THEN swap N1, N2                #mettere per primo il
                                                   maggiore
        EX = N1 – N2: RETURN                       #ottenerne la differenza

mul     OP$ = "x"
        N1 = fnc(L,U): N2 = fnc(L,U)               #scegliere due numeri
        EX = N1*N2: RETURN                         #ottenerne il prodotto

div     OP$ = "/"
        repeat N2 = fnc(L,U) until (N2 < > 0)      #scegliere il divisore
        EX = fnc(L,U)                              #scegliere la risposta
        N1 = EX*N2: RETURN                         #calcolare il dividendo
```

Figura 1.6 — Giochi Aritmetici in Free BASIC

Nella Figura 1.6 c'è una descrizione in Free BASIC del programma. Come abbiamo detto prima, potete ignorare completamente questa descrizione se non siete interessati. Tuttavia, se guardate la Figura 1.6, potrete notare diverse piccole differenze tra i programmi illustrati nella Figura 1.5 e 1.6.

COME MIGLIORARE IL PROGRAMMA E RENDERLO PIÙ COMPLETO

Infine, per quei giocatori che sono interessati a migliorare questo gioco, abbiamo inserito alcuni suggerimenti. Alcuni sono facili da applicare, mentre altri sono più difficili da mettere in pratica.

- Tenere delle statistiche, per esempio: il numero delle risposte esatte dall'ultima sbagliata, o il numero più alto di risposte esatte. Fatele apparire dopo ogni risposta esatta.
- Ricordare quali operazioni il giocatore ha sbagliato, e richiederle di tanto in tanto.
- Aggiungere altre operazioni come le radici quadrate e i numeri al cubo, conversione decimale dei numeri binari ecc.
- Usare il controllo del cursore per ottenere un display fisso: per esempio, posizionate la domanda cosicchè appaia sempre nella stessa posizione sullo schermo, e fate sì che i risultati appaiano sempre alla stessa posizione. Potete anche far scomparire le domande e i risultati dallo schermo, una volta che si presenta la nuova domanda.
- Dare la scelta al giocatore dell'operazione.
- Aggiungere l'elemento tempo (su un sistema con l'orologio, come il Pet) dando un tempo limite per la risposta, o trascrivendo la media del tempo che è stato necessario per la risposta. Se si decide di consentire un tempo limite, potreste anche aggiungere un display dei secondi via via che passano. (Vedi Cap. 3).
- Aggiungere il suono (su di un sistema con l'altoparlante, come l'Apple) per premiare il giocatore quando dà una risposta esatta e punirlo quando sbaglia. Ricordatevi però che sia possibile eliminarlo con un bottone, se il giocatore lo desiderasse.

SOMMARIO

Abbiamo studiato i due giochi dell'Addizione e delle Quattro Operazioni. Nella discussione relativa al programma dell'Addizione, abbiamo imparato la generazione dei numeri affidata al caso usando INT e RND, il comando PRINT, le variabili e le costanti, il comando INPUT e la dichiarazione IF.

Il gioco ed il programma delle Quattro operazioni spiegano il significato del termine generalizzazione e l'uso delle variabili invece delle costanti, uno dei metodi principali per la generalizzazione. Abbiamo anche imparato l'uso delle costanti a stringa in dichiarazioni INPUT, delle definizioni di funzione e delle variabili false, la costruzione ON...GOSUB, e l'uso delle subroutines.

Abbiamo anche discusso brevemente le descrizioni in Free BASIC dei due programmi e abbiamo spiegato l'uso del Free BASIC in questo libro. Vi sono le descrizioni in Free BASIC per tutti i programmi, ma si possono leggere i primi cinque capitoli senza doversi riferire al Free BASIC.

Inoltre abbiamo suggerito dei miglioramenti da apportare al programma, come statistiche, operazioni addizionali e controllo da parte del giocatore, ed infine un uso migliore di certe caratteristiche come l'orologio, o il suono, che sono particolari di certi sistemi.

CAPITOLO 2

GIOCHI D'INTUIZIONE

I giochi che abbiamo presentato nel Capitolo 1 sono stati scelti deliberatamente tra i più semplici. Ci siamo concentrati sul BASIC piuttosto che sui principii della programmazione e della progettazione di questi giochi. Abbiamo tuttavia discusso nel Capitolo 1 un concetto importante: la generalizzazione. Questo concetto giocherà ancora un ruolo importante nei giochi che descriveremo in questo capitolo. Presenteremo diversi giochi basati sull'intuizione di una parola e di un numero segreti. La nostra discussione inizierà con la revisione della forma generale dei giochi d'intuizione. Poi descriveremo un gioco chiamato Quattro ed una serie di intuizioni e di risposte che verranno usate come esempi. Questi illustreranno la strategia che viene usata da un giocatore esperto. Il capire i giochi dal punto di vista del giocatore vi aiuterà a programmare ed inventare giochi più interessanti e stimolanti.

Invece di fornire un programma di gioco che funzioni solo per il Quattro, conviene progettare una generalizzazione. Occorrerà perciò introdurre alcuni giochi d'intuizione di numeri da Uno a Nove (Il Quattro fa parte di questo gruppo). Poi, per generalizzare ancora di più, introdurremo un gioco d'intuizione chiamato Parole, che ha regole simili a quelli con i numeri.

Soltanto allora potremo progettare un programma. Questo, il Programma dei Giochi d'Intuizione, vi insegnerà diverse tecniche utili e numerosi principii. Poichè una di queste tecniche è la "cannibalizzazione" di un programma così da produrne uno simile, introdurremo il Gioco dell'Impiccato all'inizio del capitolo e poi finiremo la spiegazione "cannibalizzando" il programma dei Giochi d'Intuizione così da poter creare il programma del gioco dell'Impiccato.

FORMA GENERALE DEI GIOCHI D'INTUIZIONE

I giochi del tipo che andiamo a descrivere tendono ad essere molto simili l'uno con l'altro. In genere si gioca in due. Il primo giocatore pensa una parola od un numero, il secondo cerca d'indovinarlo facendo diversi tentativi. Le risposte che il primo gioca-

tore dà alla fine di ogni tentativo sono create in modo che il campo delle possibilità si restringa via via, cosicchè prima o poi il secondo giocatore è in grado di indovinare la risposta esatta. Inoltre, il numero dei tentativi che il secondo giocatore ha a disposizione è limitato. Nella versione per computer di questi giochi, il programma prende il posto del primo giocatore, così da consentire al secondo giocatore la certezza dell'esattezza delle risposte che riceve. Infatti è vitale che il secondo giocatore possa fidarsi del primo per usare le informazioni che gli vengono fornite via via che il gioco procede. Se le risposte non fossero più che accurate, il gioco si trasformerebbe da un gioco d'abilità in uno affidato alla fortuna.

QUATTRO

Quattro è un gioco semplice che ha diverse varianti popolari. Il programma inizia con lo scegliere un numero "segreto" di quattro cifre, in cui non vi sia nessuna cifra uguale. Poi tocca a voi. Avete diritto a fare una serie di tentativi per indovinare il numero. Naturalmente dovrete indovinare le cifre nell'ordine corretto. Per esempio, se il numero segreto è 7915, dovrete indovinarlo correttamente e non 1975. Dopo ogni tentativo, il programma vi risponde usando due numeri. Il primo vi dice quante cifre di quelle che voi avete inserito appaiono anche nel numero segreto. Per esempio, se il numero segreto è 7915, e voi avete suggerito 1975. Il primo numero che apparirà sarà 4, perchè tutte le quattro cifre appaiono anche nel numero segreto. Il secondo numero che compare sullo schermo, vi dirà invece quante delle cifre indovinate sono anche nella giusta posizione. Per esempio, se il numero segreto è 7915 e voi avete suggerito 1975, il secondo numero che comparirà sullo schermo sarà 2, perchè sia il 9 che il 5 appaiono nella posizione giusta (la seconda e la quarta).

Dunque il dialogo sullo schermo si svolgerà come segue:

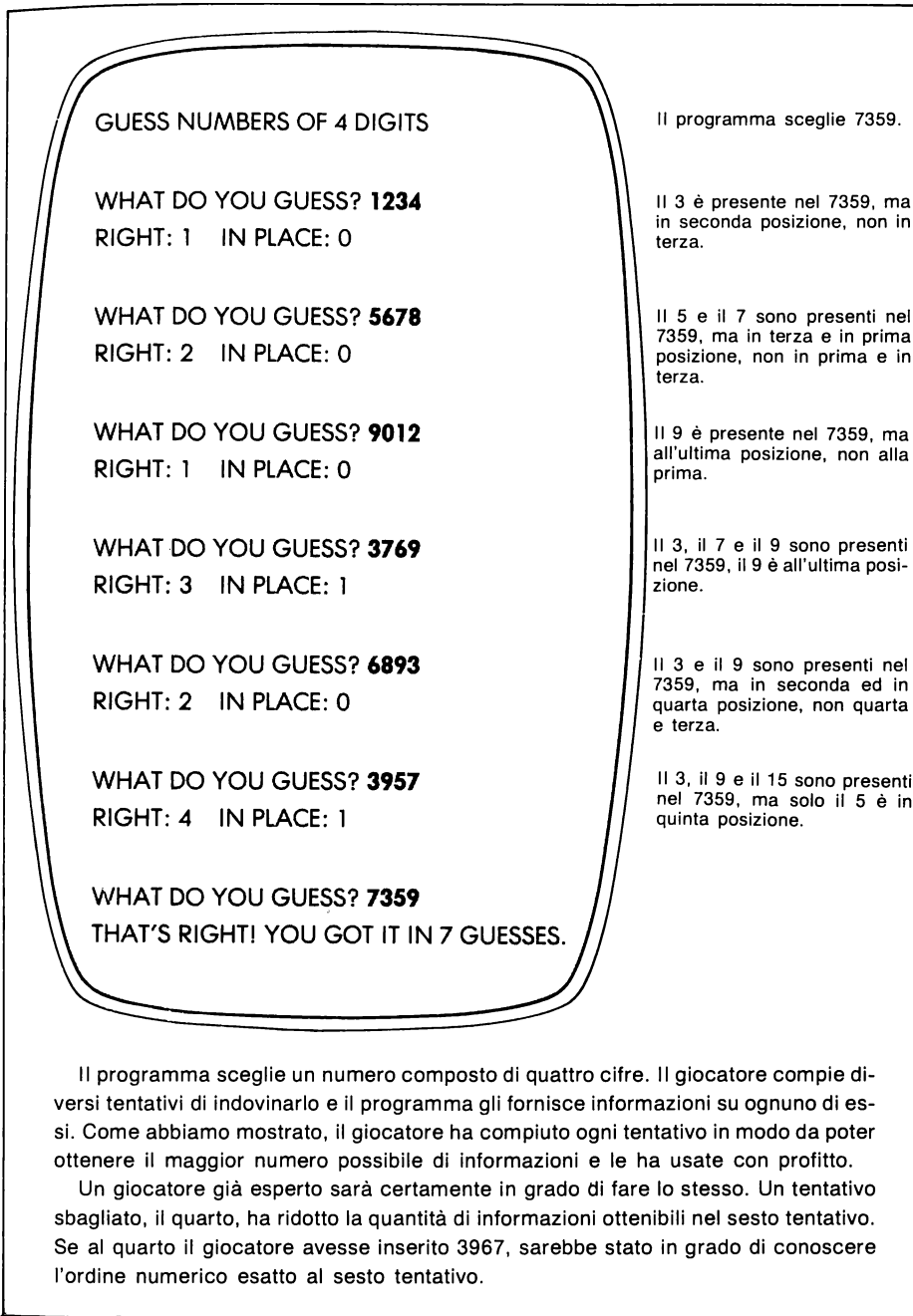
```
WHAT DO YOU GUESS? 1975
RIGHT: 4 IN PLACE: 2
WHAT DO YOU GUESS?
```

Questa sequenza di tentativi e di risposte continua finchè avete indovinato il numero segreto. Se non lo indovinate entro un certo numero di tentativi, il programma finirà il gioco da solo.

UN ESEMPIO

Nella Figura 2.1 vediamo un esempio del gioco. Esaminiamo il dialogo frase per frase così da capire esattamente come funziona il gioco, e come poter indovinare il numero più in fretta.

Il giocatore effettua il primo tentativo senza avere nessuna informazione che ri-



Il programma sceglie un numero composto di quattro cifre. Il giocatore compie diversi tentativi di indovinarlo e il programma gli fornisce informazioni su ognuno di essi. Come abbiamo mostrato, il giocatore ha compiuto ogni tentativo in modo da poter ottenere il maggior numero possibile di informazioni e le ha usate con profitto.

Un giocatore già esperto sarà certamente in grado di fare lo stesso. Un tentativo sbagliato, il quarto, ha ridotto la quantità di informazioni ottenibili nel sesto tentativo. Se al quarto il giocatore avesse inserito 3967, sarebbe stato in grado di conoscere l'ordine numerico esatto al sesto tentativo.

Figura 2.1 — Dialogo per "Quattro"

guardi il numero, sa soltanto che è composto da quattro cifre di cui nessuna è uguale all'altra. La scelta del numero cade sul 1234, una scelta nè migliore nè peggiore di qualsiasi altra. La risposta del programma "RIGHT:1 IN PLACE: 0" dice al giocatore che soltanto una delle cifre che appaiono nel numero segreto appare nel numero 1234, e che nessuna cifra è al posto giusto. Quest'ultima informazione si rivelerà utile più avanti.

Il secondo tentativo, 5678, contiene un nuovo gruppo di quattro cifre. Notate che nei tentativi iniziali il giocatore tende ad identificare al più presto possibile le cifre, usando così la gamma dei dieci numeri. La risposta "RIGHT":2 IN PLACE:0" dice al giocatore che due cifre del numero segreto si trovano anche nel numero che ha appena inserito, e sono o il 5, o il 6, o il 7, o l'8. Dunque gli è possibile dedurre che la quarta cifra è un 9 o uno 0, perchè finora ha identificato solo tre cifre e non ha ancora provato nè il 9 nè lo 0.

Il terzo tentativo del giocatore, 9012, viene effettuato così da poter determinare quale delle cifre 1, 2, 3, e 4 è quella che appare nel numero segreto e per sapere qualcosa a proposito del 9 e dello 0. La risposta: "RIGHT:1 IN PLACE:0" gli dice che sia l'1 che il 2 sono da escludere. Dunque ora sa che una delle cifre è un 3 o un 4 un'altra è un 9 o uno 0 e due devono essere scelti tra il 5, il 6, il 7 e l'8.

Nel quarto tentativo il giocatore inserisce 3769 e perciò prende una cifra dal 3 e dal 4, due cifre dal 5, 6, 7 e 8 e una dal 9 e dallo 0. La risposta: "RIGHT:3 IN PLACE:1" fornisce al giocatore delle informazioni sufficienti per sapere quali sono le cifre che compongono il numero segreto. Ironicamente il sapere che una cifra è al posto giusto non è di nessun aiuto al giocatore, almeno a questo punto.

Il quinto tentativo, 6893, è il risultato di una considerazione attenta dei tentativi precedenti: Infatti, nel quarto tentativo, sia il 3 che il 9 andavano bene, e sia il 6 che il 7. (Poichè c'era una sola cifra errata in tutto il numero, non potevano esserci due cifre errate, una per gruppo). La possibilità d'errore tra le cifre 3 e 9 era del 50%, e la possibilità che entrambe siano esatte è 1 su 4. D'altra parte, poichè ci sono sei modi per scegliere due cifre diverse prese da un gruppo di quattro cifre, la possibilità che sia il 6 che il 7 siano corretti è 1 su 6. Il giocatore decide dunque che sia il 3 che il 9 fanno parte del numero segreto, e che solo una fra le due cifre 6 e 7 ne fa parte. La risposta: "RIGHT:2 IN PLACE:0" conferma al giocatore che certamente il 7 fa parte del numero segreto, perchè l'unico cambiamento effettuato (a parte la posizione) dal quarto al quinto tentativo è stata la sostituzione del 7 con l'8, ed inoltre egli ora sa che l'8 non fa parte del numero segreto.

Per il sesto tentativo, 3957, il giocatore ha deciso di ritenere che sia il 3 che il 9 siano corretti. Sulla base di questa assunzione, il 6 non è esatto, perchè il 3 e il 9 e il 7 sarebbero state le tre cifre esatte del quarto tentativo, 3769. Ma siccome abbiamo escluso il 6 e l'8, ci rimane il 5, perchè due cifre tra il 5, il 6, il 7 e l'8 sono corrette.

L'ordine delle cifre nel sesto tentativo si è basato sull'esame delle informazioni ottenute nei tentativi precedenti. Notate che nei primi cinque tentativi nessuna delle cifre è apparsa in posizione diversa in due tentativi diversi. Poichè quattro di questi cin-

que tentativi hanno avuto come risposta: "IN PLACE:0", il giocatore può escludere alcune posizioni per ciascuna cifra. Rimangono dunque solo le seguenti possibilità:

3 può essere solo in prima o in seconda posizione

5 può essere solo nella seconda, terza o quarta posizione

7 può essere nella prima, nella seconda o nella quarta posizione

9 può essere solo in seconda o in quarta posizione.

Da questo deduciamo che solo il 5 può essere in terza posizione. Ma, non essendone certo, il giocatore decide di provare a piazzare il 9 in seconda posizione, così da avere il 3 in prima posizione, e il 7 in quarta.

La risposta: "RIGHT:4 IN PLACE:1" significa che il giocatore ha indovinato tutte le cifre, ma le ha posizionate tutte salvo una in maniera errata.

Il tentativo finale, 7359, è un colpo sicuro, il giocatore sa già che il 5 è la terza cifra, la supposizione che il 9 era la seconda è errata. Dunque non può essere che la quarta, poi bisognerà scegliere tra 3759 e 7359. Poichè una sola cifra era al posto giusto in 3769, il 3759 va escluso. Il numero dunque non può essere che 7359.

La discussione del Quattro ora verterà sui seguenti punti:

1. Come incorporare diversi giochi simili in un solo programma.
2. Come "cannibalizzare" un programma già funzionante così da crearne uno nuovo rapidamente.
3. Come usare diversi accorgimenti durante la programmazione.

Un gruppo di giochi come il Quattro

Per illustrare il primo punto, abbiamo bisogno di diversi giochi che sono simili al Quattro. Due esempi ovvi sono il Tre e il Cinque. Questi vengono giocati come il Quattro, ma usando dei numeri di tre o di cinque cifre.

Il programma dei Giochi d'intuizione che svilupperemo darà la possibilità di inventare dei giochi contenenti numeri da una a nove cifre.

La generalizzazione che va dal Quattro fino ad un gruppo di giochi come il Tre e il Cinque richiede programmazione, ma è abbastanza semplice. Per far sì che il nostro esempio sia un po' più complicato, il programma darà la possibilità al giocatore di scegliere di giocare con le parole o con i numeri. Infatti, Parola è un gioco che si gioca come il Quattro, a parte alcune differenze:

- Il giocatore cerca di indovinare una parola e non un numero.
- Una lettera può apparire più di una volta nella parola (mentre nei giochi con i numeri una cifra può apparire una volta soltanto in un numero).
- La parola da indovinare viene inserita nel computer da un secondo giocatore (mentre nei giochi con i numeri è il programma che effettua la scelta del numero).

Queste differenze ci costringeranno a preparare il programma per i Giochi d'Intuizione attentamente. Quando abbiamo completato la programmazione, dovrebbe essere possibile aggiungere altri giochi, a scelta del giocatore, usando un programma semplice. Alla fine del capitolo vi diamo dei suggerimenti per poter estendere il programma.

L'impiccato

Per illustrare meglio il secondo punto — come cannibalizzare un programma già funzionante — useremo il Gioco dell'Impiccato. L'Impiccato è un gioco simile a quelli già descritti, ma come vedrete dalla descrizione, differisce in alcune parti.

Un giocatore sceglie una parola e specifica il numero di tentativi che l'altro giocatore può effettuare. Quest'ultimo poi cerca d'indovinarla. Il programma inizia mostrando un gruppo di trattini per indicare il numero delle lettere contenute nella parola. Per esempio se la parola fosse SYZYGY e si possono fare cinque tentativi per indovinarla, sul video apparirà:

```
_ _ _ _ _ _ WRONG GUESSES LEFT: 5
WHAT DO YOU GUESS?
```

Il giocatore dovrà dunque identificare una lettera o l'intera parola. Se indovina una delle lettere che compongono la parola, il trattino corrispondente alla lettera scompare e al suo posto compare la lettera indovinata. Per esempio, se il giocatore inserisce una Y, sullo schermo si leggerà:

```
_ Y _ Y _ Y WRONG GUESSES LEFT: 5
WHAT DO YOU GUESS?
```

Il gioco continua finchè la parola non viene identificata, o il numero di tentativi superato.

Non c'è dubbio che il nostro programma possa essere costruito in modo da poter includere l'Impiccato. Tuttavia, ci sono così tante differenze tra questo e gli altri giochi che converrebbe avere un programma separato, modificando e sostituendo delle parti del programma dei Giochi d'Intuizione.

Accorgimenti da usare durante la programmazione

Il terzo punto si riferisce agli accorgimenti da usare durante la programmazione. I più importanti riguardano la codifica di una parola o di un numero segreti, e il controllo del tentativo del giocatore. La tecnica di codificazione di diverse informazioni in un solo numero è simile a quella di linguaggio che usa diversi gruppi di bits in una parola per scopi diversi. Quando parleremo del sistema che viene usato per codificare la parola od il numero segreto discuteremo di questa tecnica.

Ora esaminiamolo nei particolari, dopo che abbiamo discusso i diversi giochi che hanno lo stesso programma e dopo aver parlato della "cannibalizzazione" del programma stesso.

IL PROGRAMMA DEI GIOCHI D'INTUIZIONE

Questo programma è illustrato nelle Figure che vanno dalla 2.2 alla 2.12. La routine principale viene spiegata nella Figura 2.2. Notate che appare due volte, una volta nella descrizione in Free BASIC e una in quella in BASIC. (Come abbiamo detto prima, il Free BASIC viene illustrato soltanto per aiutarvi ad abituarvici, così da poterlo poi capire più avanti). Se volete potete ignorarlo, ma se paragonate il Free BASIC con la forma in BASIC di questo programma (e degli altri che seguiranno), la prima vi sembrerà più facile da apprendere (anche se non avete letto ancora il Cap. 6).

Il programma dei giochi d'intuizione nella Figura 2.2 consiste di tre loops uno dentro l'altro. Quello più interno, che occupa le linee 140 e 150, corrisponde ai tentativi del giocatore per indovinare la parola od il numero segreto. Il loop viene ripetuto finchè il giocatore ha indovinato tutte le lettere o le cifre e le ha posizionate correttamente (la condizione IP=N), o finchè non si sia raggiunto il numero di tentativi permesso (la condizione IP=−1). Dentro questo loop, le subroutines delle linee 360, 410 e 480 vengono chiamate per accettare, controllare e paragonare il tentativo del giocatore con la parola o il numero segreto, e per dirgli quante cifre o lettere sono corrette (RIGHT:) e quante in posizione corretta (IN PLACE). La variante IP ("in place") che controlla il loop si trova nella subroutine alla linea 410.

Il secondo loop verso l'esterno occupa l'interno programma, ad eccezione delle linee 100 e 110. Questo loop corrisponde ad una serie di giochi di un solo tipo. Alla fine di ogni gioco, il programma aspetta di ricevere un input di una sola lettera che venga dalla tastiera e che contenga i comandi seguenti. La lettera che viene inserita dal giocatore schiacciando un tasto viene accettata dalla subroutine alla linea 710 (chiamata dalla linea 160) e trattenuta dalla variabile NX$. Il comando del loop viene ripetuto finchè NX$, cioè il carattere che il giocatore inserisce, in modo che si giochi sempre lo stesso tipo di gioco, contiene "N" o "E". "N" significa che il giocatore vuole cambiare gioco, "E" che il gioco è finito.

Il loop più esterno include la linea 110, che chiama una subroutine che permette al

```
#Giochi D'Intuizione

        GOSUB init                              #creare gli arrays e le
                                                costanti
        repeat{                                 #loop principale
          GOSUB setup                           #scegliere il gioco e
                                                prepararsi
          repeat{                               #inizio partita
            GOSUB think                         #scegliere il numero o
                                                la parola da indovinare
            PRINT: PRINT IN$: PRINT             #informare il giocatore
                                                del tipo di gioco
            G = 0                               #inizio conteggio tenta-
                                                tivi
            repeat{                             #loop dei tentativi
                GOSUB guess: G = G + 1          #il giocatore fa un tenta-
                                                tivo d'indovinare
                GOSUB check                     #il programma lo con-
                                                trolla
            GOSUB hint                          #il programma dà il suo
                                                responso
              }until (IP = N OR IP = -1)        #si è raggiunto il nume-
                                                ro massimo di tentativi
                                                che è permesso
            GOSUB stats                         #dare il punteggio
            GOSUB next                          #aspettare il segnale di
                                                riinizio partita
          }until (NX$ = "N" OR NX$ = "E")       #"N" cambia il gioco
        }until (NX$ = "E")                      #"E" per smettere di
                                                giocare
        END

100     GOSUB 670
110     GOSUB 590
120     GOSUB 200:PRINT:PRINT IN$:PRINT
130     G=0
140     GOSUB 360:G=G+1:GOSUB 410:GOSUB 480
150     IF IP <> N AND IP <>-1 THEN 140
160     GOSUB 520:GOSUB 710
170     IF NX$="N" THEN 110
180     IF NX$="E" THEN END
190     GOTO 120
```

Questa è la routine principale del programma dei Giochi d'Intuizione. Prima di tutto compare una descrizione in Free BASIC e poi una in BASIC.

Figura 2.2 — Giochi D'Intuizione

giocatore di scegliere il tipo di gioco che vuole giocare. Ogni volta che il programma viene iniziato, o che il giocatore schiaccia "N" alla fine di una partita, il programma si porta alla linea 110 per dare questa possibilità di scelta.

La linea 100, che contiene una chiamata per una subroutine di "inizializzazione" alla linea 670, viene eseguita una volta all'inizio del programma. Siccome non fa parte di nessun loop, non viene più eseguita. Allo stesso modo, il comando END che segue il THEN sulla linea 180 è al di fuori del loop più esterno e viene eseguito solo una volta. Se il giocatore preme "E" alla fine di una partita, il programma lascia sia il loop esterno che quello intermedio, e finisce eseguendo il comando END.

Esaminiamo ora i comandi e le subroutine che fanno parte del programma. La prima linea è una chiamata per una subroutine che occupa dalla linea 670 alla 700. Nonostante la programmazione moderna (secondo Yourdon e altri) non incoraggi questo stile, molti programmatori trovano che aiuti iniziare un programma con una chiamata per una subroutine che dichiari le dimensioni e che decida i valori delle costanti e che sia anche in grado di compiere altre operazioni. Questa subroutine viene scritta dopo aver finito il resto del programma, perchè quando lo iniziate, sapete che sarà un certo numero di operazioni che andranno compiute all'inizio, ma non saprete quali finchè non avrete scritto il resto del programma. Discuteremo poi questa subroutine che è illustrata nella Figura 2.10.

La linea seguente, la 110, contiene il comando

```
GOSUB 590
```

Questa è una chiamata per la subroutine che stabilisce il tipo di gioco e stabilisce i valori di un gruppo di variabili. Queste hanno dei valori costanti durante lo svolgimento di certi giochi, ma il loro valore varia durante altri. Anche questa subroutine verrà esaminata in seguito (è illustrata nella Figura 2.9); in modo da capire il programma principale, abbiamo bisogno soltanto di sapere che questa subroutine determina i valori delle seguenti variabili:

GM (tipo di gioco) viene fissato a 1 per il gioco il cui scopo è d'indovinare un numero o 2, per quello in cui bisogna indovinare una parola.

N (numero delle cifre) viene fissato al numero delle cifre, per il gioco in cui bisogna indovinare un numero. Per il gioco Parola, N viene fissato alla lunghezza della parola segreta della subroutine che permette al secondo giocatore di fare i suoi tentativi e di inserirli.

MG (numero massimo di tentativi consentiti) viene fissato al numero massimo dei tentativi permessi. Nel gioco Parola, MG viene fissato insieme a N.

VG (molto bene) viene fissato ad un numero che rappresenta il numero massimo di tentativi che venga considerato come un buon risultato da parte del giocatore. Nel gioco Parola, VG viene fissato dopo N e MG.

La linea seguente, la 120 contiene quattro comandi:

```
GOSUB 200: PRINT: PRINT IN$: PRINT
```

GOSUB 200 è una chiamata per una subroutine (illustrata nella Figura 2.3) che prepara il problema seguente. La studieremo in seguito più approfonditamente. Per adesso, tutto quello che occorre sapere è che questa subroutine determina la variabile IN$ e codifica il problema in modo che venga capito dalla subroutine che controlla l'esattezza di ogni tentativo del giocatore. I tre comandi seguenti sulla linea 120 sono:

```
PRINT: PRINT IN$: PRINT
```

Questi comandi fanno sì che il testo che è contenuto in IN$ (inserito dalla subroutine alla linea 200) compaia sullo schermo (con linee vuote sia sopra che sotto). Il testo è il messaggio per il giocatore che riguarda cosa va indovinato (numero o parola).

Per esempio se abbiamo scelto Quattro, il testo contenuto in IN$ sarà

```
"GUESS NUMBERS OF 4 DIGITS"
```

Il comando

```
G = 0
```

sulla linea 130 serve per cominciare il conto del numero dei tentativi che vengono effettuati dal giocatore.

Sulla linea 140 troviamo il loop più interno, questo contiene quattro comandi:

```
GOSUB 360: G = G + 1: GOSUB 410: GOSUB 480
```

Il comando GOSUB 360 all'inizio della linea è una chiamata per la subroutine che accetta il tentativo del giocatore. Questa subroutine (che compare nella Figura 2.5) prende i tentativi del giocatore, ne fa una stringa (G$) e li suddivide poi in un array di caratteri (GC).

Dopo GOSUB 360 sullo linea 140 c'è il comando

```
G = G + 1
```

Ricordatevi che prima di entrare nel loop che inizia sulla linea 140, abbiamo inizializzato G a zero (linea 130). Ogni volta che chiamiamo la subroutine alla linea 360 (cioè quando accettiamo un tentativo del giocatore) aggiungiamo 1 al numero che è contenuto in G. Il comando G = G + 1 significa "aggiungere 1 al numero contenuto in

G''. Dunque questo numero non è altro che il numero di tentativi già effettuati dal giocatore. Il terzo comando sulla linea 140 è una chiamata per la subroutine alla linea 410. Questa subroutine, che compare alla Figura 2.6, controlla il tentativo del giocatore con la risposta esatta, usando l'array codificato. Poichè continuiamo lo studio del programma, tutto quello che è necessario sapere per quanto riguarda la subroutine, è che essa decide i valori delle variabili RG e IP. RG (cioè RIGHT - ESATTO) è il numero dei caratteri (lettere o cifre) che appaiono sia nel tentativo del giocatore che nella risposta esatta. IP (cioè IN PLACE - AL POSTO GIUSTO) è il numero dei caratteri (lettere o cifre) che si trovano nella posizione giusta.

Il comando finale sulla linea 140 è anche quello finale del loop più interno. Si tratta di una chiamata alla subroutine alla linea 480. Questa subroutine, che compare alla Figura 2.7, prende i valori di IP e RG, che sono stati determinati dalla subroutine alla linea 410, e li usa per generare il display:

```
RIGHT: 3 IN PLACE: 2
```

Inoltre è ancora la stessa subroutine che scrive:

```
THAT'S RIGHT!
```

o

```
TOO MANY GUESSES - YOU LOSE
```

La prima frase viene scritta se IP = N (cioè quando tutti i caratteri si trovano al posto giusto); la seconda corrisponde a G = MG (cioè quando si è raggiunto il numero massimo dei tentativi senza però indovinare). In quest'ultimo caso, IP viene fissato a −1 come comando per il programma principale per finire il loop che accetta e controlla il tentativo del giocatore.

La linea 150 è la parte di controllo delle condizioni del loop che inizia alla linea 140. Il comando

```
IF IP< >N AND IP< > −1 THEN 140
```

significa "Se il giocatore non è ancora riuscito ad indovinare la risposta (IP< >N) e se il giocatore non ha ancora raggiunto il numero massimo di tentativi (IP< > −1), allora ritornare alla linea 140 per passare di nuovo attraverso il loop". Il simbolo < > usato in questo comando significa "non è uguale a".

La linea 160 contiene due comandi:

```
GOSUB 520: GOSUB 710
```

Questi sono i due comandi che completano il gioco. Il primo è una chiamata per la

subroutine alla linea 520. Questa routine, illustrata nella Figura 2.8, stampa il punteggio del gioco. Usando il sistema di conteggio dei tentativi G, che è inizializzato sulla linea 130 e aggiornato nel loop sulla linea 140, questa routine scrive:

YOU GOT IT IN 4 TRIES

Se il valore di G è minore od uguale a quello di VG, allora la routine scrive:

VERY GOOD

La routine dice anche:

YOU MUST BE PSYCHIC

Nel caso in cui il giocatore indovini al primo colpo. Anche perchè altrimenti si verificherebbe un errore se la routine scrivesse:

YOU GOT IT IN 1 TRIES

Il secondo comando sulla linea 160 è una chiamata per la subroutine alla linea 710. Questa subroutine, illustrata nella Figura 2.10, accetta un input di un solo carattere e lo conserva in NX$. Questo input è il comando del giocatore per il programma, infatti, dice al programma che cosa va fatto alla fine della partita. Le tre possibilità che vengono interpretate dai comandi sulle linee 170, 180 e 190 sono:

N Permette al giocatore di scegliere un altro gioco.

E Permette di interrompere il gioco e di tornare a BASIC.

qualsiasi altro carattere permette di iniziare un altro gioco dello stesso tipo.

L'uso di un input di un solo carattere fa sì che il gioco proceda senza intoppi. Per esempio, supponiamo di aver appena finito di giocare a Quattro. Il programma ci ha detto:

THAT'S RIGHT! YOU GOT IT IN 8 TRIES

I tentativi precedenti sono ancora sullo schermo, così da dare la possibilità di controllarli e vedere dove sia possibile migliorare. Basterà ora schiacciare la sbarretta dello spazio e lo schermo si pulisce ed inizia così una nuova partita.

Il programma vi dà dunque tutto il tempo che volete per esaminare una partita precedente, oppure potete iniziarne subito un'altra senza dover rispondere alla domanda:

DO YOU WISH TO PLAY ANOTHER GAME (Y OR N)?

```
#Pensare al prossimo problema

think       GOSUB clearscreen                    #Inizio del gioco a
                                                 schermo vuoto
            FOR ZZ = BH TO TH                    #azzerare l'array hit
              HT(ZZ) = 0
              NEXT ZZ
            ON GM GOSUB number, askword
            RETURN

#Pensare ad un numero con N cifre, tutte diverse

number      FOR ZZ = 1 TO N
              repeat
                HH = INT(RND(1)*10) + L          #Codice ASCII per una
                                                 cifra
                until (HT(HH) = 0)               #Nessuna cifra deve es-
                                                 sere uguale ad un'altra
              HT(HH) = HT(HH) + 2↑(ZZ - 1)       #Determinare la poten-
                                                 za per questa posizione
              NEXT ZZ
            IN$ = "GUESS NUMBERS OF" + STR$(N) + "DIGITS."
            GL = N                               #Indovinare la lunghez-
                                                 za
            RETURN
```

```
200   GOSUB 740
210   FOR ZZ=BH TO TH:HT(ZZ)=0:NEXT ZZ
220   ON GM GOSUB 310,240
230   RETURN

310   FOR ZZ=1 TO N
320*  HH=INT(RND(1)*10)+L:IF HT(HH) <> 0 THEN 320
330   HT(HH)=HT(HH)+2↑(ZZ-1):NEXT ZZ
340** IN$="GUESS NUMBERS OF"+STR$(N)+"DIGITS.":GL=N
350   RETURN
```

Questa è la subroutine che costruisce il problema che seguirà. Inoltre vi è anche una delle sue due subroutines — quella che determina un numero di N cifre, ognuna diversa dalle altre. Ogni subroutine viene mostrata prima in Free BASIC e poi in BASIC.

---

* Nella versione per il TRS-80, il termine RND(0) compare invece di RND(1).

** La versione per l'Apple di questa linea è diversa soltanto nella spaziatura nelle costanti a stringa.

Figura 2.3 — Subroutine Think per i Giochi d'Intuizione

Quando schiacciate la sbarretta dello spazio avete il completo controllo del gioco, mentre quando dovete rispondere ad una domanda come quella sopra descritta, è il programma che controlla il gioco. Mantenendo il controllo, il gioco risulta più interessante. Supponiamo ora di voler cambiare invece di Four vogliamo giocare a Five. Invece di schiacciare la sbarretta dello spazio alla fine della partita, premiamo il tasto

```
#Chiedere una parola che diventerà quella da indovinare nel prossimo problema

askword     repeat{                                    #ottenere una stringa
              INPUT "HIDDEN WORD"; H$
            N = LEN(H$)
              } until (0 < N AND N < = MX)
            IN$ = "GUESS WORDS OF" + STR$(N) + "LETTERS."
            GL = N                                     #indovinare la lunghezza
            FOR ZZ = 1 TO N                            #determinare l'array per
                                                       caratteri della stringa
              HH = ASC(MID$(H$, ZZ, 1) )
              HT(HH) = HT(HH) + 2↑(ZZ - 1)
              NEXT ZZ
            VG = N                                     #"very good" (molto bene) se si sono fatti così pochi tentativi d'indovinare
            MG = maxguess                              #"You lose" (Perdi) se > di questo numero di tentativi
            GOSUB clearscreen                          #cancellare la parola dallo schermo
            RETURN

240*  INPUT "HIDDEN WORD";H$:N=LEN(H$)
250   IF N=0 OR N>MX THEN 240
260*  IN$="GUESS WORDS OF"+STR$(N)+"LETTERS.":GL=N
270   FOR ZZ=1 TO N:HH=ASC(MID$(H$,ZZ,1) )
280   HT(HH)=HT(HH)+2↑(ZZ - 1):NEXT ZZ
290   VG=N:MG=20
300   GOSUB 740:RETURN
```

Questa è l'altra subroutine della prima routine illustrata nella Figura 2.3. Questa subroutine chiede al giocatore di inserire una parola. La subroutine viene illustrata prima in Free BASIC e poi in BASIC.

* La versione dell'Apple di queste linee è diversa soltanto nel valore delle costanti di stringa.

Figura 2.4 — Subroutine Askword per i Giochi d'Intuizione

N. Lo schermo si pulisce e il programma vi consente di scegliere un nuovo tipo di gioco. Paragonate questa operazione con il dialogo seguente:

```
DO YOU WISH TO PLAY ANOTHER GAME (Y OR N)? N
DO YOU WISH TO PLAY A DIFFERENT KIND OF GAME (Y OR N)?
```

Naturalmente si tratta solo di una scelta personale. Se preferite il dialogo di cui sopra, sta a voi modificare il programma. Bisognerà soltanto rimuovere il GOSUB 710 dalla linea 160, e sostituire alle linee 170, 180 e 190, dei comandi modificati. Per effettuare queste modifiche non è necessario che voi conosciate a fondo le subroutines. Ora che conosciamo a fondo il programma esaminiamo le sue subroutines in dettaglio.

Inizieremo con la subroutine alla linea 200, illustrata nella Fig. 2.3, e le due sue subroutines alle linee 310, 240, illustrate nelle Figure 2.3 e 2.4.

La subroutine alla linea 200 costituisce un nuovo problema. Per esempio, se giochiamo a Quattro, questa subroutine sceglie il numero a quattro cifre che il giocatore deve indovinare; se invece giochiamo a Parola, essa chiederà al secondo giocatore di scegliere una parola che il primo giocatore dovrà indovinare. Dopo aver scelto un numero o una parola, la routine lo codifica, usando l'array HT, di cui parleremo in seguito.

Lo scopo della codifica è di rendere più semplice il controllo dei diversi tentativi del giocatore.

Un modo ovvio per controllarli sarebbe di controllare ogni carattere che il giocatore inserisce rispetto a quello che si trova nella posizione corrispondente nella parola o nel numero esatti. Se essi fossero uguali, entrerebbero in funzione IP e RG, e il programma potrebbe procedere con il carattere seguente. Se invece il carattere non corrispondesse a quello della risposta esatta, il programma potrebbe cercare il carattere inserito dal giocatore in qualsiasi altra posizione; nel caso in cui lo trovasse, entrerebbe in funzione RG, e il programma potrebbe procedere con il carattere seguente.

Un esercizio interessante potrebbe essere l'uso di una routine che si avvale dell'algoritmo sopraccitato per sostituire la subroutine della Figura 2.6. Probabilmente vi renderete conto che usando questa routine ci vuole molto più tempo per controllare i tentativi. Inoltre, se decideste di usarla, dovreste modificare la subroutine che si trova sulla linea 310 in modo da formare una stringa con le quattro cifre del numero che viene lì costruito. Poi dovreste mettere questa stringa nella variabile H$. La subroutine alla linea 240 usa questa variabile per conservare la parola che viene inserita dal secondo giocatore.

Il metodo che usiamo al momento per codificare il problema permette un controllo rapido; si costruisce un array chiamato HT, che è uguale al numero dei caratteri ASCII possibili: 128 per l'Apple e il TRS-80, 256 per il Pet. (Il Pet usa un gruppo di caratteri ASCII un po' più vasto che ha anche dei codici per i caratteri grafici che possono essere inseriti dalla tastiera).

All'inizio l'array Ht viene azzerato. Poi, per ciascuna cifra componente il numero, o per ciascuna lettera della parola, si eseguisce un inserimento nell'array HT alla posizione corrispondente ai codici ASCII per quel determinato carattere.

Per esempio, diciamo che stiamo giocando a Four e che il numero scelto dal programma è 4726. Per questo numero si eseguono degli inserimenti nell'array HT alle posizioni 52, 55, 50 e 54, perchè il codice ASCII per il 4 è 52, per il 7 è 55, per il 2 è 50 e per il 6 è 54. (Vedere l'appendice A per la lista dei codici ASCII).

Però l'inserimento nell'array HT dei codici ASCII non fornisce nessuna informazione per quel che riguarda l'ordine in cui i caratteri appaiono. Per esempio, sia il 4726 e il 2674 sarebbero inseriti nelle stesse quattro posizioni dell'array HT. Siccome abbiamo bisogno di conoscere la loro posizione, gli inserimenti devono anche specificarla. Per esempio, per 4726 potremmo mettere un 1 in HT (50) un 2 in HT (55), un 3 in HT (50) e un 4 in HT (52), così da far sì che ciascuna posizione nell'array HT contenga un numero che specifichi la posizione del carattere corrispondente nella risposta esatta.

Questo andrebbe bene per i giochi con i numeri come Four, perchè non si ha mai una cifra ripetuta due volte nello stesso numero, ma non per Parola, perchè in questo gioco, una stessa lettera viene ripetuta anche più volte nella parola. Per esempio, se la parola scelta è BILL, allora l'inserimento nell'array HT (76) — cioè il codice per L — avrebbe bisogno di avere sia un 3 che un 4, perchè appare sia in terza che in quarta posizione. Potremmo però far apparire sia il 3 che il 4 nel HT (76) semplicemente regolando l'HT (76) e 24, cioè $2^3 + 2^4$.

Certamente saprete che un numero intero (come 24) può essere diviso in una somma di potenze di due in un solo modo. Dunque, poichè HT (76) ha il valore 24, troviamo che $24 = 2^3 + 2^4$, dunque la 3za e la 4ta sono le due posizioni dove è la lettera L nella parola BILL. Il codice che compie questo calcolo appare nella subroutine che inizia alla linea 410 (Figura 2.6).

La subroutine alla linea 200 azzera prima di tutto le parti dell'array HT che sono necessarie nel gioco prescelto. Per esempio per Quattro e per altri giochi numerici, solo dieci parti dall'HT (48) all'HT (57) (che corrispondono ai caratteri ASCII per le dieci cifre 0, 1 ...9) vengono azzerate. Le variabili BH e TH ("bottom hit" e "Top hit") che specificano questa gamma di indici di array, vengono determinate dalla subroutine presentata nella Figura 2.9. Per Quattro, per esempio, BH = 48 e TH = 57. Allora si azzerano soltanto quelle parti dell'HT che occorrono per il gioco, poichè per azzerare tutte le 256 si perderebbe troppo tempo.

Dopo aver azzerato le parti necessarie, la routine esegue il comando:

```
ON GM GOSUB 310,240
```

Questo fa sì che vengano chiamate o la subroutine alla linea 310 o quella alla 240, a secondo del valore della variabile GM. Questo valore viene determinato nella subroutine alla Figura 2.9. Il valore di GM è 1 per ciascuno dei giochi numerici, mentre è 2 per il gioco Parola. Dunque la subroutine alla linea 310 (di cui alla Figura 2.3) vie-

ne usata per i giochi numerici, mentre quella alla linea 240 (di cui alla Figura 2.4) viene usata per il gioco Parola. Queste due subroutines sono brevi e semplici da capire, ma in alcune parti necessitano di un commento.

Prima di tutto, esaminiamo la linea 320 nella subroutine usata per i giochi numerici:

```
320 HH = INT(RND(1)*10) + L: IF HT (HH)<>0 THEN 320
```

Il primo comando che compare sulla linea 320 sceglie a caso un numero dai dieci numeri iniziando con quello il cui valore è contenuto nella variabile L. La routine d'inizializzazione determina che L sia 48, cioè il codice ASCII per zero, così il comando sceglierà a caso un numero tra i dieci che iniziano con 48. A causa della posizione del gruppo dei caratteri ASCII, questi dieci numeri sono i codici ASCII per 0, 1 ...9.

Il secondo comando sulla linea 320 controlla gli inserimenti dell'array HT che corrispondono al codice ASCII prescelto. Se si tratta di un numero che non sia zero, il programma ritorna all'inizio della linea 320 e ricomincia a scegliere. Cosicchè il programma sia certo di non avere un numero con due cifre uguali.

La linea seguente, la 330, contiene il comando che ha la giusta potenza di due nell'inserimento seguente dell'array HT che corrisponde al codice ASCII scelto:

```
HT(HH) = HT(HH) + 2↑(ZZ - 1)
```

Poichè la linea precedente si è assicurata che HT(HH) abbia valore 0, questo comando equivale a:

```
HT(HH) = 2↑(ZZ - 1)
```

Il valore di ZZ, l'indice per il loop FOR...NEXT, va da 1 al numero delle cifre che si vogliono inserire (per es. 4 cifre per Quattro). Dunque, se 50 (il codice ASCII per 2) viene scelto sulla linea 320 per la prima volta attraverso il loop, allora ZZ = 1 e HH = 50, cosicchè il comando di cui sopra diventa:

```
HT(50) = 2↑(0)
```

Se 54 (cioè il codice per 6) viene scelto per la seconda volta attraverso il loop, allora il comando diventa

```
HT(54) = 2 (1)
```

Se si sceglie di giocare a Quattro, e il numero da indovinare è 2649, allora HT (50) conterrà 1 (poichè 2 0 = 1), HT(54) conterrà 2, HT(52) conterrà 4 e HT (57) conterrà 8. I rimanenti sei inserimenti dell'array HT (dei dieci inserimenti da HT (48) fino HT(57) che abbiamo azzerato nella subroutine della linea 200) rimarranno a zero. Le

operazioni finali che vengono eseguite dalla subroutine alla linea 310 sono quelle di fissare i valori di IN$ (un messaggio "veloce" che appare sul video sia all'inizio della partita che quando il giocatore sbaglia) e di GL (il numero dei caratteri che sono previsti in ciascun tentativo).

La subroutine alla linea 240 (illustrata nella Figura 2.4) è simile a quella alla linea 310. Notate che le linee 270 e 280 contengono i comandi:

```
HH = ASC(MID$(H$,ZZ,1))
HT(HH) = HT(HH) + 2↑(ZZ - 1)
```

Il primo di questi comandi mostra come i codici ASCII corrispondenti ai caratteri della parola da indovinare vengono estratti uno alla volta. L'espressione

```
MID$(H$,ZZ,1)
```

rappresenta una stringa di un solo carattere di lunghezza (a causa della discussione 1), presa da H$ (poichè H$ è la prima discussione) che inizia alla posizione ZZ. Poichè ZZ è la variabile dell'indice per il loop FOR...NEXT, prende dei caratteri presenti in H$ (a causa del comando N = LEN(H$) sulla linea 240). Dunque, MID$(H$,ZZ,1) rappresenta il carattere ZZ di H$ quale stringa ad un solo carattere. La funzione ASC trasforma questa fila in un numero, cioè nel codice ASCII. Questo numero viene raccolto in HH, dove si trasforma in un indice per l'array HT.

Il comando

```
HT(HH) = HT(HH) + 2↑(ZZ - 1)
```

sulla linea 280 è identico al comando usato sulla linea 330 (vedi Figura 2.3). Parlando della linea 330, abbiamo notato che HT(HH) aveva valore zero, così il comando equivaleva a:

```
HT(HH) = 2↑(ZZ - 1)
```

Nel caso della linea 280, questo non è più valido, poichè lo stesso carattere (e dunque lo stesso valore di HH) può essere già capitato prima nelle lettere che formano la parola da indovinare. Dunque, in questo caso, la giusta potenza di due (2 (ZZ − 1)) viene aggiunta al contenuto precedente di HT(HH).

Se il carattere dato non è ancora apparso nella parola segreta, si avrà zero, mentre se il carattere è già apparso, il contenuto sarà una somma di termini del tipo 2 (ZZ-1) con valori di ZZ che corrispondono alle posizioni in cui è apparso il carattere.

Il resto della subroutine della Figura 2.4 non ha bisogno di spiegazione. Notate che finisce con una chiamata per la subroutine alla linea 740 (di cui alla Figura 2.12) per cancellare le immagini sullo schermo. Questo perchè la parola da indovinare rimanga segreta, e che si possa giocare senza perdere nessun dato via via che questi si muovono verso l'alto durante il gioco.

Nella Figura 2.5 potete vedere la subroutine che chiede ed accetta il tentativo del giocatore. Le prime due linee, 360 e 370, contengono i comandi

```
360 INPUT"WHAT DO YOU GUESS";G$)
    IF LEN(G$ GL THEN PRINT IN$:GOTO 360
```

Questi comandi vengono ripetuti finchè il giocatore non inserisca una risposta che contenga esattamente il numero di caratteri specificato in GL (per esempio, quattro per il gioco "Quattro"). La stringa IN$(cioè il messaggio originale) viene ripetuto se il giocatore ha inserito un numero errato di caratteri. Allora il giocatore può riprovare. Notate che i caratteri rifiutati non vengono contati nel numero totale dei tentativi. La variabile G viene solo aumentata dal ritorno di questa subroutine (vedere linea 140 Figura 2.2).

Le due linee seguenti, 380 e 390, contengono i comandi che sezionano la risposta del giocatore e raccolgono i codici ASCII per i caratteri individuali del tentativo nell'array GC:

```
#Accettare il tentativo d'indovinare del giocatore

guess       repeat{
              INPUT "WHAT DO YOU GUESS";G$
              IF LEN(G$) < > GL THEN
                PRINT IN$
              }until (LEN(G$) = GL)
            FOR ZZ = 1 TO GL                    #inserire i caratteri del
                                                tentativo nell'array GC
              GC(ZZ) = ASC(MID$(G$,ZZ,1) )
              NEXT ZZ
            RETURN

360*   INPUT "WHAT DO YOU GUESS";G$
370    IF LEN(G$) < >GL THEN PRINT IN$:GOTO 360
380    FOR ZZ=1 TO GL
390    GC(ZZ)=ASC(MID$(G$,ZZ,1) ):NEXT ZZ
400    RETURN
```

Questa è la subroutine dei Giochi d'Intuizione che chiede ed accetta il tentativo d'indovinare fatto dal giocatore, poi lo divide in un array di caratteri. Questa subroutine viene illustrata prima in Free BASIC e poi in BASIC.

---

* La versione Apple di questa riga differisce in apparenza per un punto di domanda nella stringa costante.

Figura 2.5 — Subroutine Guess per i Giochi d'Intuizione

```
FOR ZZ = 1 TO GL
GC(ZZ) = ASC(MID$(G$,ZZ,1)): NEXT ZZ
```

Questi sono simili ai comandi che sezionano la parola da indovinare (vedi linea 270, Figura 2.4).

Il tentativo del giocatore (ora codificato nell'array GC) viene controllato con la risposta esatta (codificata in HT) dalla subroutine che inizia alla linea 410. Questa su-

```
#Controllare il tentativo d'indovinare del giocatore con la parola o il numero segreti

check     RG = 0                          #iniziare con nessun
                                          corretto, nessuno al po-
                                          sto giusto
          IP = 0
          FOR ZZ = 1 TO GL                #controllare il tentativo
                                          con l'array "hit"
            HH = HT(GC(ZZ))
            IF HH < > 0 THEN              #il carattere è da qual-
                                          che parte nella stringa
              RG = RG + 1
              HH = INT(HH/2↑(ZZ - 1))     #guardare nella potenza
                                          di due corrispondente
              IF HH is odd THEN {         #il carattere è al posto
                  IP = IP + 1             giusto
              }
            NEXT ZZ
          RETURN

410   RG=0:IP=0
420   FOR ZZ=1 TO GL:HH=HT(GC(ZZ))
430   IF HH=0 THEN 460
440   RG=RG+1:HH=INT(HH/2↑(ZZ - 1))
450   IF HH < >2*INT(HH/2) THEN IP=IP+1
460   NEXT ZZ
470   RETURN
```

Questa subroutine controlla l'array GC (che è stato costruito usando il tentativo d'indovinare del giocatore dalla subroutine nella Figura 2.5) con l'array HT. HT consente un solo inserimento per ciascun carattere ASCII. Se il carattere non fa parte della risposta esatta, l'inserimento è una somma di potenze di due – una per ogni posizione nella risposta alla quale compare un certo carattere.

La routine viene illustrata prima in Free BASIC e poi in BASIC.

Figura 2.6 — Subroutine Check per i Giochi d'Intuizione

broutine compare nella Figura 2.6. La routine comincia con l'inizializzazione di "RIGHT" e di "IN PLACE" chiamandoli RG e IP con valore 0 (linea 410). Il resto della routine è un loop, che occupa le linee dalla 420 alla 450. Questo loop è uno dei FOR...NEXT indicizzato da ZZ. Nel loop ogni carattere che compone il tentativo del giocatore di indovinare (conservato negli elementi 1 tramite GL dell'array GC) viene controllato con il carattere corrispondente nell'array HT. Il comando

```
HH = HT(GC(ZZ))
```

sulla linea 420 sceglie l'inserimento dell'array HT che corrisponde al carattere ZZ della risposta fornita dal giocatore. Se HH ha valore zero, allora il carattere ZZ del tentativo del giocatore non apparirà nella risposta, e RG verrà aumentato di 1 (linea 440). Poi il programma controllerà per vedere se 2 (ZZ − 1) è uno dei termini che compongono il valore di HH. Se lo è il programma aumenta IP. Il codice per l'esecuzione di questo comando è nelle linee 440 e 450:

```
HH = INT(HH/2↑(ZZ − 1))
IF HH< >2*INT(HH/2)THEN IP = IP + 1
```

Il primo di questi comandi elimina tutte le potenze esistenti minori di 2 (ZZ-1).

Se preferite pensare a questo comando in termini matematici, il valore originale di HH segue la formula:

$$a_0 + a_1 \times \ldots + a_{GL-1} \times 2^{GL-1}$$

dove a_0, a_1, ... a_{GL-1} prendono i valori di 0 o di 1. Gli indici 0, 1 e GL−1 corrispondono ai valori ZZ da 1 a GL, cosicché l'operazione di cui sopra rimpiazza i valori originali con:

$$a_{zz-1} + (azz \times 2 + \ldots + a_{GL-1} \times 2^{GL-ZZ})$$

quando ZZ GL e da a_{GL-1} quando ZZ = GL. Il valore del termine in parentesi è sempre pari, così il valore di a_{ZZ-1} può essere ottenuto semplicemente determinando se il nuovo valore di HH sia pari (nel cui caso $a_{ZZ-1} = 0$) o dispari (nel cui caso $a_{ZZ-1} = 1$). La condizione

```
HH< >2*INT(HH/2)
```

è equivalente a "il valore di HH è dispari".

Se avete avuto difficoltà a seguire la spiegazione e la discussione dell'operazione di questa subroutine non è il caso di preoccuparvi, non sarà necessario capirne i particolari per capire le spiegazioni seguenti.

La subroutine illustrata nella Figura 2.7 è stata discussa in breve in precedenza, è estremamente semplice, la cosa più importante da ricordare è che essa determina IP al valore di −1 se il giocatore non ha più diritto a nessun altro tentativo.

La subroutine illustrata nella Figura 2.8 è anch'essa abbastanza semplice, ma ha una funzione importante che può essere certamente migliorata ed approfondita. Questa subroutine vi dice come avete giocato e si congratula con voi se avete giocato particolarmente bene. Potrebbe anche, volendo, rendere il gioco più interessante e paragonare la partita attuale con quelle precedenti.

La subroutine illustrata alla Figura 2.9 viene usata per scegliere il tipo di gioco da giocare. Questa routine viene chiamata all'inizio del gioco e tutte le volte che il giocatore schiaccia il tasto "N" alla fine di una partita. Dopo aver pulito lo schermo con una chiamata per la subroutine alla linea 740, la routine chiede "GAME": per sapere a che cosa volete giocare. Poi accetta la vostra risposta che gli sarà data a mezzo di un solo carattere. Se inserite una cifra da 1 a 9, avrete scelto il gioco numerico corri-

```
#Dare un suggerimento (o annunciare una vincita o una perdita)

hint        IF IP = N THEN                          #tutti al posto giusto
              PRINT "THAT'S RIGHT!";
            else IF G = MG THEN {                   #si è raggiunto il nume-
                                                    ro massimo
              PRINT "TOO MANY GUESSES – YOU LOSE!"
              IP = −1                               #segnalare che la parti-
              }                                     ta è finita

            else
              PRINT "RIGHT:"; RG, "IN PLACE:"; IP
            RETURN

480   IF IP=N THEN PRINT "THAT'S RIGHT!";:GOTO 510
490   IF G=MG THEN PRINT "TOO MANY GUESSES – YOU LOSE!":IP= –
      1:GOTO 510
500*  PRINT "RIGHT:"; RG, "IN PLACE:"; IP
510   RETURN
```

Questa subroutine dei Giochi d'Intuizione controlla per vedere se il giocatore ha indovinato la risposta o ha usato il numero massimo di tentativi permesso. Questa è la routine che fissa IP a −1 se il giocatore non ha più tentativi da utilizzare. La routine è illustrata prima in Free BASIC e poi in BASIC.

---

* La versione per l'Apple di questa linea è diversa soltanto nella spaziatura delle costanti di stringa e l'uso di un TAB (11) invece di una virgola dopo "RG".

Figura 2.7 — Subroutine Hint per i Giochi d'Intuizione

spondente, se invece inserite una "W" avete scelto il gioco Parola; qualsiasi altro input selezionerà il gioco Quattro, chiameremo perciò Quattro la soluzione di "default".

L'operazione di questa routine si basa sul fatto che la funzione VAL ridà il valore numerico di ogni stringa che possa essere interpretata come un numero, mentre ridà il valore zero per quella stringa che non rappresenta un numero (per es. "W").

Le due subroutines alle linee 650 e 660 decidono i valori di MG (Maximum Guesses = tentativi massimi permessi) e di VG (Very Good = molto bene) e i limiti BH e TH dell'array HT. Si possono trovare dei modi più sofisticati per scegliere i valori di MG e di VG. Inoltre, si possono aggiungere altre categorie (oltre a VG) e queste pos-

```
#Dare le statistiche per la partita appena conclusasi

stats       IF IP = N THEN {                          #tutti i caratteri al posto
                                                      giusto
              IF G = 1 THEN                           #indovinato al primo
                                                      colpo
                PRINT "YOU MUST BE PSYCHIC."
              else{
                PRINT "YOU GOT IT IN";G;"GUESSES."
                IF G <= VG THEN                       #indovinato in un nume-
                                                      ro basso di tentativi
                  {PRINT: PRINT "VERY GOOD."}
                }
              }
            else IF IP = -1 THEN                      #non è rimasto nessun
                                                      tentativo da utilizzare
              PRINT "YOU COULDN'T GET IT IN";MG;"TRIES."
            RETURN
```

```
520   IF IP<> N THEN 570
530   IF G=1 THEN PRINT "YOU MUST BE PSYCHIC.":GOTO 580
540*  PRINT "YOU GOT IT IN";G;"GUESSES."
550   IF G<=VG THEN PRINT:PRINT "VERY GOOD."
560   GOTO 580
570*  IF IP=-1 THEN PRINT "YOU COULDN'T GET IT IN";MG;"TRIES."
580   RETURN
```

Questa subroutine del programma dei Giochi d'Intuizione viene chiamata quando la partita è finita: il giocatore ha indovinato la risposta o tutti i tentativi sono stati utilizzati. In quest'ultimo caso, il programma ha commesso un errore madornale.

La routine viene illustrata prima in Free BASIC e poi in BASIC.

---

* La versione per l'Apple differisce soltanto nella spaziatura delle costanti di stringa.

Figura 2.8 — Subroutine Stats per i Giochi d'Intuizione

```
#Preparare la partita seguente
setup       GOSUB clearscreen
            PRINT "GAME:";
            GOSUB onech: N = VAL(X$)          #codice per il tipo di gioco
            IF N <> 0 THEN                    #se cifre si tratterà d'indovinare un numero
              GM = 1
            else IF X$ = "W" THEN             #un "W" significherà che bisogna indovinare una parola
              GM = 2
            else
              {N = 4: GM = 1}                 #impossibilità ad indovinare numeri di 4 cifre
              ON GM GOSUB setnum, setask
            RETURN
setnum      VG = N + 2: MG = 2*VG             #fissare il livello di "very good" ed il numero massimo
            BH = L: TH = L + 9                #gamma dell'array da usare
            RETURN
setask      VG = 0: MG = 99                   #VG e MG vengono determinati da askword
            BH = 0: TH = numascii             #gamma di hit array da usare
            RETURN
```

```
590  GOSUB 740:PRINT "GAME:";:GOSUB 720
600  N=VAL(X$):IF N <> 0 THEN GM=1:GOTO 630
610  IF X$="W" THEN GM=2:GOTO 630
620  N=4:GM=1
630  ON GM GOSUB 650,660
640  RETURN
650  VG=N+2:MG=2*VG:BH=L:TH=L+9:RETURN
660* VG=0:MG=99:BH=0:TH=255:RETURN
```

Questa è la subroutine del programma dei Giochi d'Intuizione che permette al giocatore di scegliere una delle versioni possibili del gioco: indovinare un numero o una parola scelta da un altro giocatore. Per la maggior parte la routine suggerisce delle possibilità (per esempio un modo migliore per determinare VG e MG) e fornisce un sistema adatto a facilitare l'aggiunta a posteriori di altri tipi di gioco.

La routine viene illustrata prima in Free BASIC e poi in BASIC.

* La versione per l'Apple e quella per il TRS-80 di questa linea ha sostituito il 255 con 127.

Figura 2.9 — Subroutine Setup per i Giochi d'Intuizione

sono essere usate dal programma nella Figura 2.8 in modo da dare al giocatore un feedback più selettivo.

La variabile GM ed il comando

```
ON GM GOSUB 650,660
```

fanno sì che esista una cornice in cui inquadrare delle routines per altri giochi che possono così essere facilmente. In altre parole, questa parte del programma dei Giochi d'intuizione è stata progettata per facilitare una generalizzazione.

Le subroutines illustrate nella Figura 2.10 rientrano nella categoria che viene di solito per ultima nell'elenco di un programma. La prima subroutine (linea 710) consiste del comando:

```
GOSUB 720: NX$ = X$: RETURN
```

La chiamata per questa routine potrebbe essere sostituita da una chiamata diretta al programma alla linea 720, e NX$ potrebbe essere sostituita da X$ direttamente. Questo porterebbe a dei cambiamenti alle linee 160, 170 e 180 (Vedere Figura 2.2).

La subroutine alla linea 710 fornisce un valido esempio di una situazione che ha luogo spesso in un programma ben costruito. Una routine che è stata originariamente progettata per uno scopo ben definito, può finire per diventare nient'altro che una chiamata singola per un'altra subroutine. Per ottenere chiarezza ed efficienza, queste routines dovrebbero essere eliminate, ma, poichè sono innocue, vengono spesso tenute.

La seconda subroutine illustrata nella Figura 2.10 è la routine d'inizializzazione. Questa comparirebbe normalmente alla fine del programma, ma abbiamo tenuto questa posizione per le subroutines del programma dei Giochi D'Intuizione le cui versioni per l'Apple, per il Pet e per il TRS/80 differiscono in modi non poco importanti.

La routine d'inizializzazione determina due dimensioni d'array e la variabile MX; MX viene usata nella subroutine illustrata nella Figura 2.4 per assicurarsi che la parola da indovinare non abbia più caratteri di quelli allocatigli nell'array GC. (Ricordatevi che l'array GC conserva soltanto i caratteri individuali del tentativo attuale del giocatore). La routine d'inizializzazione inoltre determina il valore L del codice ASCII per zero. In un'organizzazione precedente del programma è stato necessario e logico mettere L nella routine d'inizializzazione, mentre ora è più logico compiere questa operazione nella subroutine alla linea 650 (vedere Figura 2.9).

Nelle Figure 2.11 e 2.12 vediamo delle subroutines che hanno versioni diverse per l'Apple, il Pet e il TRS-80. Tutte le routines precedenti hanno comandi identici per i tre sistemi, a parte le solite piccole differenze – per esempio, lo spazio prima e dopo un numero e il "?" con i comandi d'input e le differenze per quanto riguarda la dimensione dell'array HT.

La prima delle routines che hanno versioni dipendenti dal sistema è la routine d'in-

put ad un solo carattere illustrata nella Figura 2.11 che usa diversi "attrezzi" per ogni sistema diverso. Per esempio, la versione Pet di GET non aspetta di ricevere un input proveniente dalla tastiera. Questo si ottiene includendo il comando GET in un loop che abbia anche una chiamata per la funzione RND. Come abbiamo spiegato nel Cap. 1, questo rende possibile una scelta veramente affidata al caso. La versione per l'Apple del comando GET attende l'input, così non può (o non deve) essere usata in

```
#Ottenere il codice dal giocatore per la prossima partita

next        GOSUB onech                    #ottenere un input ad un
                                           solo carattere
            NX$ = X$                       #inserirlo in NX$
            RETURN
#Iniziamo

init        DIM HT(numascii)               #array indicizzato dal
                                           codice ASCII
            MX = maxword: DIM GC(MX)       #array per i caratteri del
                                           tentativo
            L = ASC("0")                   #limite minimo per la
                                           selezione delle cifre
            RETURN

710   GOSUB 720:NX$=X$:RETURN

670*  DIM HT(255)
680   MX=20:DIM GC(MX)
690   L=ASC("0")
700   RETURN
```

Queste sono due piccole subroutines del programma dei Giochi d'Intuizione. La prima che viene chiamata alla fine di ogni partita, ritorna una stringa ad un solo carattere in NX$ che il programma principale interpreta come una possibilità di un gruppo di tre: – il giocatore desidera giocare ad un altro tipo di gioco (NX$="N"), il giocatore vuole smettere di giocare (NX$="E"), il giocatore vuole continuare con lo stesso tipo di gioco (una qualsiasi altra stringa). Lo scopo principale di questa routine è di fornire la pausa alla fine della partita durante la quale il giocatore può esaminare il risultato della partita stessa.

La seconda subroutine è una subroutine d'inizializzazione. I programmatori più tradizionali iniziano sempre i programmi con una chiamata ad una routine d'inizializzazione. Questa determina le dimensioni di due arrays e i valori di due costanti.

Le routines vengono illustrate prima in Free BASIC e poi in BASIC.

* Le versioni per l'Apple e per il TRS-80 di questa linea usano una dimensione d'array di 127.

Figura 2.10 — Due piccole subroutines per i Giochi d'Intuizione

un loop; perchè il comando GET non riconosce la solita convenzione dell'Apple di controllo – C per interrompere il programma, questa possibilità viene introdotta da un controllo esplicito (CHR$(3) rappresenta una stringa ad un solo carattere che è il controllo – C).

Il TRS-80 usa la funzione INKEY$ esattamente allo stesso modo in cui il Pet usa il comando GET.

```
#Ottenere un input di un solo carattere

#Versione per il Pet

onech       repeat{
              X$ = CHR$(RND(1))
              GET X$
            } until (X$ < >"")
            RETURN

720   X$=CHR$(RND(1)):GET X$:IF X$="" THEN 720
730   RETURN

#Versione per l'Apple
onech       GET X$
            IF X$ = CHR$(3) THEN                    #far funzionare il con-
                                                    trollo C
              STOP
            RETURN

720   GET X$:IF X$=CHR$(3) THEN STOP
730   RETURN

#Versione per il TRS-80
onech       repeat {
              X$=CHR$(RND(0))
              GET X$
            } until(X$ < >"")
            RETURN

720   X$=CHR$(RND(0)):X$=INKEY$:IF X$="" THEN 720
730   RETURN
```

Questa è l'operazione importante di input ad un solo carattere non a ripetizione. I tre diversi modi di seguire l'operazione (per il Pet, per l'Apple e per il TRS-80) sono stati qui illustrati.

Le routines vengono spiegate prima in Free BASIC e poi in BASIC. Notate che tutte e tre le versioni in BASIC hanno gli stessi numeri di linea.

Figura 2.11 – Input ad un solo carattere

La subroutine di pulizia dello schermo illustrata nella Figura 2.12 consiste di un comando singolo per ognuna delle tre versioni. Sul Pet lo schermo si svuota usando PRINT, cioè quando si scrive il carattere che corrisponde al tasto CLR. Per l'Apple e il TRS-80 viene usato un comando BASIC separato: HOME per l'Apple e CLS per il TRS-80.

Così si conclude la discussione sul programma dei Giochi d'Intuizione. Ora vediamo come questo programma può essere trasformato in quello dell'Impiccato.

IL PROGRAMMA DELL'IMPICCATO

All'inizio di questo capitolo abbiamo imparato a giocare all'Impiccato, ora impareremo come si fa a far derivare un programma da un altro già esistente. I cambiamenti principali sono illustrati nelle Figure dalla 2.13 alla 2.17, adesso le discuteremo brevemente.

Prima di tutto, bisogna notare che il gioco dell'Impiccato usa la stessa routine principale dei Giochi d'intuizione. Cioè, il programma illustrato nella Figura 2.2, senza es-

```
#Pulire lo schermo e muovere il cursore nella posizione di partenza

#Versione per il Pet

clearscreen  PRINT "clr";
             RETURN
  740  PRINT CHR$(147);:RETURN
#Versione per l'Apple
clearscreen  HOME
             RETURN
  740  HOME:RETURN
#Versione per il TRS-80
clearscreen  CLS
             RETURN
  740  CLS:RETURN
```

Questa è la subroutine che pulisce lo schermo e muove il cursore all'angolo in alto a sinistra. La pulizia dello schermo viene effettuata per due ragioni: primo perchè sullo schermo appaia solo una partita alla volta, e poi per far si che non sia possibile conoscere la parola o il numero da indovinare.

La routine è illustrata prima in Free BASIC e poi in BASIC, per il Pet, l'Apple e il TRS-80.

Figura 2.12 — Pulizia dello schermo

```
#Inserire la parola da indovinare-per l'Impiccato

think       GOSUB clearscreen
            repeat {
              INPUT "HIDDEN WORD"; H$
              N = LEN(H$)
            } until (0 <N <=MX)
            INPUT "HOW MANY WRONG GUESSES ARE ALLOWED"; WG
            IN$ = "GUESS ONE LETTER OR THE WHOLE WORD"
            GL = 1: IP = 0                          #"in place" (al posto
                                                    giusto) inizia a 0, e non
                                                    diminuisce mai
            MG = 99: VG = 5                         #valori arbitrari
            GOSUB clearscreen                       #celare la parola da in-
                                                    dovinare
            FOR ZZ = 0 TO numascii                  #azzerare l'array hit
              HT(ZZ) = 0
              NEXT ZZ
            FOR ZZ = 1 TO N
              HH = ASC(MID$(H$, ZZ, 1))             #creare l'array hit
              HT(HH) = HT(HH) + 2↑(ZZ - 1)
              GS(ZZ) = ASC("-")                     #"already guessed" (già
                                                    provato) = tutti trattini
              NEXT ZZ
            RETURN

200   GOSUB 740
210*  INPUT "HIDDEN WORD";H$:N=LEN(H$)
220   IF N=0 OR N > MX THEN 210
230*  INPUT "HOW MANY WRONG GUESSES ARE ALLOWED";WG
240   IN$="GUESS ONE LETTER OR THE WHOLE WORD"
250   GL=1:IP=0:MG=99:VG=5:GOSUB 740
280** FOR ZZ=0 TO 255:HT(ZZ)=0:NEXT ZZ
290   FOR ZZ=1 TO N:HH=ASC(MID$(H$,ZZ,1))
300   GS(ZZ)=ASC("-")
310   HT(HH)=HT(HH)+2↑(ZZ - 1):NEXT ZZ
320   RETURN
```

Questa è la versione per il gioco dell'Impiccato delle subroutines illustrate nelle Figure 2.3 e 2.4. La codifica dell'array rimane uguale. Un nuovo array chiamato GS viene inizializzato con tutti i trattini. Le lettere che vengono indovinate sostituiscono via via i trattini corrispondenti.

* La versione per l'Apple di queste linee differisce soltanto nei valori delle costanti di stringa.
** Nelle versioni per l'Apple e per il TRS-80, 127 sostituisce 255.

Figura 2.13 — Subroutine Think per il gioco dell'Impiccato

sere alterato, agisce da routine principale anche per il Gioco dell'Impiccato. Necessita però fare un piccolo compromesso, poichè il comando "N" che il giocatore inserisce e che viene riconosciuto alla linea 170 dalla routine alla linea 590 (chiamata dalla linea 110). Questa routine dovrà essere programmata ad ignorare le richieste di selezione di un nuovo gioco.

Seguendo l'ordine che è illustrato nelle figure che compongono i Giochi d'Intuizione, esamineremo le versioni del codice che si usano nel gioco dell'Impiccato. La Figura 2.13 mostra la versione delle routines usate nel gioco dell'Impiccato a che sono quelle dei giochi d'Intuizione illustrate alla Figura 2.3 e 2.4. La versione del gioco dell'Impiccato elimina la subroutine che riunisce i numeri dalla linea 310 alla 350 nella Figura 2.3, poichè all'Impiccato si gioca usando solo parole che vengono inserite da un giocatore. La versione del gioco dell'Impiccato è simile alla subroutine sulle linee dalla 240 alla 300, perchè ottiene la parola segreta dal primo giocatore e la codifica nell'array HT, ma il primo giocatore deve anche fornire il numero massimo di tentativi permessi al secondo giocatore.

La stringa di suggerimenti IN$ nella Figura 2.13 è anche leggermente diversa da quella nei Giochi d'Intuizione, perchè ogni tentativo nel gioco dell'Impiccato consiste di un solo carattere o di una parola. IP viene qui inizializzato, perchè questa variabile viene usata in modo diverso nei due giochi. Nei Giochi d'Intuizione, IP appartiene soltanto al tentativo precedente, mentre nell'Impiccato una volta indovinato un carattere, questo viene continuamente mostrato dal programma nella posizione corretta. Allo stesso modo, IP inizia da zero e aumenta via via che ogni posizione viene riempita. Il fatto che GL sia fissato ad 1 stabilisce che i tentativi di indovinare nel gioco dell'Impiccato sono limitati ad un solo carattere. (Delle prove speciali sono poi necessarie per far sì che si possa indovinare l'intera parola).

L'array GS al quale ci siamo riferiti alla linea 300 nella Figura 2.13 è nuovo. Questo array viene usato per accumulare i caratteri che il giocatore ha indovinato. In questa routine GS viene inizializzato per contenere dei trattini in tante posizioni quante sono le lettere che compongono la parola da indovinare. Quando via via queste lettere vengono indovinate, esse sostituiranno i trattini sullo schermo.

Nella Figura 2.14 viene illustrata la versione per il gioco dell'Impiccato della subroutine illustrata nella Figura 2.5. Le due subroutines sono molto simili, ma nella versione del gioco dell'Impiccato sullo schermo apparirà una riga di questo tipo:

```
_Y_Y_Y      WRONG GUESSES LEFT: 5
```

L'altra differenza è che nella versione per il gioco dell'Impiccato, se un tentativo d'indovinare contiene lo stesso numero di caratteri della parola segreta, non viene controllato. Altrimenti i caratteri del tentativo vengono conservati in GC usando gli stessi comandi della Figura 2.5. Naturalmente per l'Impiccato, i comandi nel loop FOR...NEXT vengono eseguiti solo una volta, perchè GL ha sempre valore 1. Dunque

GC(1) viene usato per conservare il codice ASCII per il singolo carattere del tentativo d'indovinare, ed il resto dell'array GC non viene usato.

```
*Accettare il tentativo del giocatore

guess       PRINT                                  #a capo
            FOR ZZ = 1 TO N                        #mostrare le parti della
                                                   parola che sono state in-
                                                   dovinate finora
            PRINT CHR$ (GS(ZZ) );
              NEXT ZZ
            PRINT" WRONG GUESSES LEFT:";WG         #e il numero di tentativi
                                                   che si possono ancora
                                                   utilizzare
            repeat {
              INPUT "WHAT DO YOU GUESS:"; G$
              IF LEN(G$) = N THEN
                 RETURN
              else IF LEN(G$) < > GL THEN
                 PRINT IN$
              } until (LEN(G$) = GL)
            FOR ZZ = 1 TO GL                       #inserire i caratteri dei
                                                   tentativi in GC
              GC(ZZ) = ASC(MID$(G$, ZZ, 1) )
              NEXT ZZ
            RETURN

360   PRINT:FOR ZZ=1 TO N:PRINT CHR$(GS(ZZ) );:NEXT ZZ
365*  PRINT" WRONG GUESSES LEFT:";WG
370*  INPUT "WHAT DO YOU GUESS";G$
375   IF LEN(G$)=N THEN RETURN
380   IF LEN(G$) < > GL THEN PRINT IN$:GOTO 370
390   FOR ZZ=1 TO GL
395   GC(ZZ)=ASC(MID$(G$,ZZ,1) ):NEXT ZZ
400   RETURN
```

Questa è la versione per il gioco dell'Impiccato della subroutine illustrata nella Figura 2.5. Le uniche differenze che esistono tra questa versione e quella illustrata nella Figura 2.5 sono: (1) il display iniziale delle parti della parola che sono state indovinate e il numero dei tentativi che il giocatore può ancora utilizzare, e (2) il RETURN immediato se LEN(G$) =N.

---

* La versione per l'Apple di queste linee differisce soltanto nei valori delle costanti di stringa.

Figura 2.14 — Subroutine Guess per il Gioco dell'Impiccato

```
*Controllare il tentativo del giocatore con il numero o la parola da indovinare
check       IF LEN(G$) = N THEN
              IF G$ = H$ THEN IP = N                  #l'intera parola è stata
                                                      indovinata
              else WG = WG - 1
            else {
              IX = 0                                  #contare il numero delle
                                                      volte che il carattere
                                                      compare
              FOR ZZ = 1 TO GL
                HH = HT(GC(ZZ))                       #inserimento per il ca-
                                                      rattere indovinato
                IF HH < > 0 THEN
                  FOR YY = 1 TO N
                    IF HH is odd THEN{
                      IP = IP + 1                     #un altro al posto giusto
                      IX = IX + 1                     #un altro per questo ca-
                                                      rattere
                      GS(YY) = GC(ZZ)                 #sostituire il trattino con
                      }                               il carattere
                  HH = INT(HH/2)
                  NEXT YY
                NEXT ZZ
              IF IX = 0 THEN
                WG = WG - 1
              }
            RETURN
```

```
410  IF LEN(G$) < > N THEN 440
420  IF G$=H$ THEN IP=N:RETURN
430  WG=VG-1:RETURN
440  IX=0:FOR ZZ=1 TO GL:HH=HT(GC(ZZ))
445  IF HH=0 THEN 460
450  FOR YY=1 TO N:IF HH< >2*INT(HH/2) THEN IP=IP+1:IX=I-
     X+1:GS(YY)=GC(ZZ)
455  HH=INT(HH/2):NEXT YY
460  NEXT ZZ
465  IF IX=0 THEN WG=-1
470  RETURN
```

Questa è la versione per il gioco dell'Impiccato della subroutine illustrata nella Figura 2.6. Le routines hanno strutture simili, però con alcune differenze. La variabile RG non compare più ma IP rappresenta sempre il numero dei caratteri al posto giusto. Se l'intera parola viene indovinata al primo tentativo si ha IP=N.

Figura 2.15 — Subroutine Check per il gioco dell'Impiccato

Nella Figura 2.15 vediamo la versione del gioco dell'Impiccato della subroutine illustrata nella Figura 2.14. La routine nella Figura 2.15 controlla questo tentativo direttamente con la parola da indovinare che è contenuta in H$. Se sono perfettamente uguali, allora IP viene portato a N, così da segnalare al giocatore che ha indovinato. Altrimenti il tentativo viene considerato come sbagliato.

Se il tentativo d'indovinare non contiene lo stesso numero di caratteri della parola segreta, allora il carattere singolo contenuto in GC(1) viene controllato con l'elemento corrispondente dell'array HT. Questo ha luogo nel loop FOR...NEXT indicizzato da ZZ. (I comandi del loop vengono eseguiti solo una volta, perchè GL ha valore 1). Il contenuto HH dell'elemento dell'array HT viene controllato (come nella Figura 2.6) per determinare quali potenze di due sono presenti. Il carattere che rappresenta il tentativo d'indovinare viene inserito, in ciascuna posizione per la quale esiste una potenza di due in una parte dell'array GS, al posto dei trattini con i quali l'array era stato indicizzato. Questo viene eseguito dal comando:

```
GS(YY) = GC(ZZ)
```

sulla linea 450.

Un errore interessante che si è presentato in una versione precedente della subroutine illustrata alla Figura 2.15. La prova sulla linea 410 era per LEN(G$) = GL invece di LEN(G$) N. Questo sembrerebbe non costituire nessuna differenza, perchè i due soli valori che LEN(G$) può avere sull'inserimento in questa routine sono N e GL, così le due prove dovrebbero aver risultati uguali, tuttavia si potrebbero presentare delle ambiguità se N = GL. In questo caso, tuttavia, non dovrebbe essere importante quale delle due viene scelta, perchè un paragone dell'intera stringa dovrebbe dare lo stesso risultato di un paragone effettuato carattere per carattere. Infatti, quando GL = N il paragone carattere per carattere cade, perchè la subroutine illustrata nella Figura 2.14 non stabilisce l'array GC quando LEN(G$)=N. Dunque la prova all'inizio della subroutine illustrata nella Figura 2.15 deve essere coordinata con la prova illustrata nella Figura 2.14 (oppure la routine illustrata nella Figura 2.14 deve essere alterata per renderla a prova d'errore eliminando così la prova per determinare se LEN(G$) = N quando GL = N).

Due subroutines che hanno bisogno di essere coordinate in questo modo si chiamano "accoppiate". Meno accoppiamenti ci sono nei vostri programmi, maggiori probabilità di successo nella loro modifica.

La Figura N. 2.16 è la versione per il gioco dell'Impiccato della subroutine indicata nella Figura 2.7. Le due subroutines differiscono solo in due punti. Per l'Impiccato, la nuova variabile WG (WRONG GUESSES – tentativi errati) deve essere provata, e non è necessario generare la formula

```
RIGHT: 3 IN PLACE: 1
```

La versione per il gioco dell'Impiccato della subroutine illustrata nella Figura 2.8 non viene presentata. Le due versioni sono identiche, a parte dove la routine illustrata

nella Figura 2.8 usa la parola "IT" riferendosi alla risposta, la versione per il gioco dell'Impiccato usa la stringa H$ che contiene la risposta. Per esempio, la linea 540 della versione per il gioco dell'Impiccato è

```
PRINT "YOU GOT";H$:"IN":G:"GUESSES".
```

Un cambiamento simile viene effettuato sulla linea 570, così da eliminare l'omissione nel programma illustrata nella Figura 2.8: infatti il giocatore non riceve mai la risposta.

Nella Figura 2.17 viene illustrata la versione per il gioco dell'Impiccato della routine di cui alla Figura 2.9. Questa routine è stata ridotta ad una chiamata singola per la subroutine che pulisce lo schermo. Poichè non vi sono altre versioni del gioco che il giocatore può scegliere, questo si presenta più semplice.

C'è solo un altro cambiamento necessario per convertire il programma dei Giochi d'Intuizione in quello dell'Impiccato ed è l'aggiunta di GS(MX) alla dichiarazione di dimensione sulla linea 680 (illustrata nella Figura 2.10). Il resto del programma dei Giochi di Intuizione viene usato senza modificarlo.

Dovreste studiare la creazione del gioco dell'Impiccato partendo dai Giochi d'Intuizione attentamente, perchè si tratta di una tecnica che si dimostrerà utile in futuro, e cioè la "cannibalizzazione" di un programma eseguito velocemente, così da produrne un altro. Mentre la presentazione di questi cambiamenti è stata lunga e detta-

```
#Annunciare una vincita o una perdita

hint        IF IP = N THEN {
              PRINT "THAT'S RIGHT!";
            else IF G = MG OR WG < 0 THEN
              PRINT "TOO MANY GUESSES – YOU LOSE!"
              IP = – 1
              }
            RETURN

      480   IF IP=N THEN PRINT "THAT'S RIGHT! ";:GOTO 500
      490   IF G=MG OR WG<0 THEN PRINT "TOO MANY GUESSES – YOU
            LOSE!":IP=–1
      500   RETURN
```

Questa è la versione per il gioco dell'Impiccato della subroutine illustrata nella Figura 2.7. La differenza consiste nel fatto che questa subroutine controlla che WG < 0 e G=MG quando la partita finisce, e non dà alcun suggerimento al giocatore. Il display del risultato ha luogo nella subroutine illustrata nella Figura 2.14.

Figura 2.16 — Subroutine Hint per il Gioco dell'Impiccato

gliata, la conversione vera e propria dai Giochi d'Intuizione al gioco dell'Impiccato è stata compiuta dall'autore in meno di un'ora. Nel valutare queste modifiche dovrete fare attenzione alla loro semplicità e alla chiarezza del codice convertito. Decidete da soli se è necessario mantenere la variabile GL (che può solo avere valore 1) o la L il cui valore viene stabilito nella routine d'inizializzazione e non viene mai usata.

AGGIUNTE E CAMBIAMENTI POSSIBILI

Questo capitolo vi ha presentato i Giochi d'Intuizione (dieci in uno) e il suo derivato, l'Impiccato. Questi programmi si possono certamente migliorare. Eccovi alcune possibilità, alcune si possono eseguire facilmente, altre hanno bisogno di una programmazione più accurata:

- Statistiche — Per esempio vorreste conoscere il numero minimo di tentativi che avete usato per indovinare la risposta, o la loro media, o il numero di tentativi usati nelle ultime dieci partite.
- Un sommario durante la partita — Per esempio, un riassunto di tutti i tentativi e di tutte le risposte precedenti. Dovrete usare un formato conciso, così da avere abbastanza spazio sullo schermo. Per il gioco dell'Impiccato, è necessario mostrare un alfabeto che elimini le lettere che sono già state cancellate e le sostituisca con degli asterischi.
- Rivedere il display delle domande e delle risposte — usare il cursore così da poter avere sullo schermo la maggior parte delle informazioni disponibili.
- Fornire al programma un minimo di carattere — Per esempio, se si tratta di Quattro, e il giocatore cerca di inserire una cifra che era già stata eliminata da un tentativo precedente, o se il gioco è quello dell'Impiccato, e il giocatore cerca di inserire una lettera che era già stata eliminata, il programma potrebbe gentilmente ed educatamente informare il giocatore (Per esempio potrebbe dire: YOU ALREADY TRIEDJ, ARE YOU A BIT ABSENT-MINDED "HAI GIÀ PROVATO LA J, FORSE

```
#Prepararsi per la partita che seguirà

setup       GOSUB clearscreen
            RETURN
590     GOSUB 740:RETURN
```

Questa è la versione per il gioco dell'Impiccato della subroutine illustrata nella Figura 2.9. In effetti non ha gran che da fare.

Figura 2.17 — Subroutine Setup per il Gioco dell'Impiccato

SEI UN PO' DISTRATTO") Questo tipo di comportamento del programma deve essere progettato attentamente, perchè se il programma risponde con arroganza, il giocatore potrebbe irritarsi.

- Se il vostro sistema ha anche il suono — Potreste usarlo per premiare o punire il giocatore.
- Scegliere un metodo effettivo — Per determinare i livelli di VG (Very Good) e di MG (Maximum number of guesses), così da far si che possano variare a secondo dell'abilità del giocatore.

SOMMARIO

Questo capitolo è iniziato con la discussione dei Giochi d'Intuizione. Abbiamo spiegato le regole del Quattro e di altri giochi suoi derivati, come quello con numeri ad otto cifre, e come Parola.

Abbiamo sviluppato un programma chiamato Giochi d'Intuizione per migliorare sia Quattro che i suoi derivati. La discussione di questo programma si è accentrata sulla sua struttura e sulle tecniche usate per ottenere le generalizzazioni necessarie per avere tutti i dieci giochi in un solo programma.

Dopo aver discusso a fondo il programma dei Giochi d'Intuizione, ne abbiamo derivato quello del gioco dell'Impiccato, tramite la "cannibalizzazione", o il cambiamento e la sostituzione di alcune subroutines chiave. Lo scopo era quello di illustrare le tecniche usate per la conversione di un programma già funzionante in un programma simile, perchè si tratta decisamente del metodo più veloce per la creazione di un nuovo programma.

Infine, sono stati elencati alcuni suggerimenti per migliorare i programmi, così da illustrare le critiche che si possono fare al comportamento del computer, al fine di ottenere degli esercizi stimolanti, interessanti e con ottimi risultati.

CAPITOLO 3

GIOCHI CON IL TEMPO

Il tempo è un elemento importante nei giochi. L'azione veloce e misurata è spesso essenziale per potersi divertire durante una partita. Come vedremo le tecniche usate per controllare l'elemento tempo sono semplici – una volta che le avrete assimilate. I programmi che compaiono in questo capitolo sono stati concepiti in modo da illustrare l'uso dell'elemento tempo.

L'OROLOGIO DEL PET

In questo capitolo i giochi sono stati tutti scritti per il computer Pet, perchè ha un orologio incorporato semplice da usare. Prima di tutto vediamo come funziona. Dentro al macchinario c'è un orologio a cristalli che scatta 60 volte al secondo. Il Pet traduce questi scatti in una stringa a sei cifre che conserva nella variabile TIME$. Le sei cifre rappresentano il numero delle ore, dei minuti e dei secondi che sono passati dalla mezzanotte precedente. Per esempio, la stringa "042715" corrisponde alle ore 4:27 e 15 secondi. La stringa "120000" corrisponde a mezzogiorno e "235959" corrisponde ad un secondo prima della mezzanotte.

Sfortunatamente il Pet non funziona a pile e non ha altra fonte alternativa d'energia per garantire che l'orologio continui a funzionare anche quando viene a mancare la corrente, così ogni volta che accendete il vostro Pet, l'orologio segna 000000, cioè mezzanotte.

Però vi è possibile sapere che ore sono determinando il valore della variabile TIME$. Per esempio, se avete appena acceso il vostro Pet e l'ora esatta è 10.27, potete scrivere

```
TIME$ = "102700"
```

per aggiustare l'orologio. Per essere più precisi potete scrivere

```
TIME$ = "102800"
```

e poi attendere che siano le 10:28 per premere il tasto RETURN. Una volta inserito l'orario nel Pet, vi sarà sempre possibile sapere che ore sono, scrivendo (o inserendo nel programma) il comando

```
PRINT TIME$
```

Il Pet stamperà così una stringa di sei cifre con ore, minuti e secondi nel formato che abbiamo spiegato. (C'è poi un'altra variabile TI, nella quale il Pet conserva l'orario in un altro formato, ma non la useremo in questo capitolo). È possibile avere l'orologio anche sull'Apple e sul TRS-80, e se il vostro computer ha l'orologio, vi sarà possibile usare i programmi che presentiamo in questo capitolo. Le parti del programma che si riferiscono specificatamente al Pet sono state raccolte in sezioni speciali di codice che possono essere facilmente sostituite con codici equivalenti per l'Apple e il TRS-80.

L'OROLOGIO

Iniziamo con un gioco che non è un gioco: faremo diventare il Pet un costosissimo orologio digitale. In questo "gioco" il computer mostra continuamente l'ora esatta nell'angolo in basso a destra dello schermo. Le ore vengono indicate con le dodici ore (a.m. per le ore del mattino e p.m. per quelle dopo mezzogiorno). Per esempio, se l'orologio dice che sono le 152537, sullo schermo apparirà 3:25:37 p.m. mentre il nostro orologio fa lampeggiare i secondi nell'angolo destro dello schermo, aspetterà anche di ricevere dei comandi dalla tastiera. (Sfortunatamente mentre si effettuano altre operazioni su di un Apple, questo non attende di ricevere domande dalla tastiera, perchè la versione Apple di GET aspetta di ricevere una risposta, al contrario della versione Pet di GET o della funzione INKEY$ del TRS-80. Sull'Apple si può usare un segnale speciale per informare il programma che il giocatore vuole inserire un comando usando la tastiera).

Ogni comando consiste di un solo carattere. Essi sono:

F Fa avanzare l'orologio più rapidamente. Il programma chiede un numero N. Aggiungerà così un secondo per ogni numero N di secondi.

S Fa rallentare l'orologio. Il programma chiede un numero N. Rallenterà così di un secondo per ogni numero N di secondi.

T Decide l'orario. Il programma chiederà un astringa di sei cifre che contenga l'ora, i minuti, e i secondi nel formato prima descritto.

A Fissa l'allarme. Il programma chiede una stringa di sei cifre nello stesso formato usato per T. Quando l'orologio segna l'ora specificata, suonerà un allarme (Poichè il Pet non ha segnali acustici, ci sarà un allarme visivo).

Q Spegne l'allarme (se ha già suonato) e lo cancella, così domani non suonerà.

R Spegne l'allarme, ma non lo cancella, così suonerà domani alla stessa ora.

Z Cambia il fuso orario (o aggiusta l'orologio per fissarlo sull'ora legale o solare). Il programma chiede di quante ore si deve spostare. La risposta consisterà in un numero intero, N, positivo se le ore vanno aggiunte e negativo se vanno sottratte. (N o −N)

C Sposta l'orario di pochi secondi. Il programma chiede di quanti secondi si deve spostare. La risposta consisterà in un numero intero, N, positivo se i secondi vanno aggiunti, negativo se vanno sottratti (No −N).

Naturalmente questi comandi sono solo alcuni di quelli che si possono avere in un orologio controllato da un computer.

IL PROGRAMMA DELL'OROLOGIO

La Figura 3.1 mostra la routine principale del programma dell'orologio. Questi programmi saranno illustrati sia in Free BASIC che in BASIC, ma nel testo non si farà

```
#Programma dell'Orologio

          GOSUB init
          GOSUB clearscreen
          repeat {
            GOSUB events          #controllare la sveglia,
                                  ecc.
            GOSUB positime        #posizionare il cursore
                                  per il display dell'ora
            GOSUB showtime        #mostrare l'orario sullo
                                  schermo
            GOSUB command         #eseguire il comando,
                                  se ce n'è
          }

100   GOSUB 1110:GOSUB 920
110   GOSUB 340:GOSUB 240:GOSUB 250:GOSUB 120:GOTO 110
```

Questa è la routine principale per il programma dell'Orologio. L'orario viene controllato automaticamente nel Pet. L'orario compare in una posizione specifica sullo schermo. Il programma controlla, ad ogni scatto dell'orologio, determinati "eventi" periodici: il display di un allarme, e l'aggiustamento necessario per compensare un circuito non molto accurato. Il programma è anche alla continua ricerca di comandi ad un solo carattere provenienti dalla tastiera. Questi possono interrompere il display dell'orario mentre ha luogo il dialogo, ma il programma aggiorna sempre l'orario anche durante il dialogo.

Figura 3.1 — L'Orologio del Pet

nessun riferimento alla versione in Free BASIC. Se avete dato un'occhiata alle versioni in Free BASIC dei programmi precedenti, o se avete letto la descrizione del Free BASIC nel capitolo 6, probabilmente vi sarete resi conto che il Free BASIC è più facile del BASIC.

Il programma dell'orologio è composto da chiamate per le routines d'inizializzazione e di pulitura dello schermo, seguite da un loop infinito. Nel loop l'ora viene mostra-

```
#Esecuzione dei comandi

command    GET CM$
           IF CM$ < > "" THEN {
             GOSUB clearscreen
             on case CM$ GOSUB
               "F"   faster
               "S"   slower
               "T"   settime
               "A"   setalarm
               "Q"   killalarm
               "R"   resetalarm
               "Z"   changehour
               "C"   changesec
             GOSUB clearscreen
             }
           RETURN


120* GET CM$=IF CM$ = "" THEN 230
130  GOSUB 920
140  IF CM$="F" THEN GOSUB 450:GOTO 220
150  IF CM$="S" THEN GOSUB 480:GOTO 220
160  IF CM$="T" THEN GOSUB 520:GOTO 220
170  IF CM$="A" THEN GOSUB 550:GOTO 220
180  IF CM$="Q" THEN GOSUB 590:GOTO 220
190  IF CM$="R" THEN GOSUB 600:GOTO 220
200  IF CM$="Z" THEN GOSUB 690:GOTO 220
210  IF CM$="C" THEN GOSUB 720:GOTO 220
220  GOSUB 920
230  RETURN
```

Questa subroutine, che viene continuamente chiamata nel loop principale nella Figura 3.1, ricerca dei comandi ad un solo carattere provenienti dalla tastiera. Se ne trova uno, chiama la subroutine appropriata.

---

* Nella versione per il TRS-80, GET CM$ viene sostituito da CM$=INKEY$.

Figura 3.2 — L'esecuzione dei comandi per l'Orologio

ta ripetutamente, i comandi provenienti dalla tastiera vengono eseguiti e gli orari fissati per determinati eventi vengono controllati con l'ora attuale. L'aggiustamento dell'ora viene effettuato automaticamente nel Pet, così da non perdere del tempo mentre il programma sta rispondendo ad un comando che provenga dalla tastiera. Nel caso in cui un evento determinato (per esempio il suono dell'allarme) dovrebbe accadere mentre il programma sta eseguendo un comando, accadrà soltanto immediatamente dopo il completamento del comando stesso. Vedremo in seguito come funziona quando esamineremo la Figura 3.4.

La Figura 3.2 mostra la routine di esecuzione di un comando che viene chiamata dal loop principale nella Figura 3.1. Il funzionamento di questa routine è facile da capire. Se non viene schiacciato nessun tasto, la routine esegue il comando RETURN. Se invece si è schiacciato un tasto, la routine pulisce lo schermo, chiama la routine che esegue il comando, pulisce lo schermo e poi ritorna all'inizio.

La Figura 3.3 mostra le due subroutines che sono coinvolte nel display dell'ora. La prima semplicemente posiziona il cursore al suo posto sullo schermo. La seconda ha due funzioni ben distinte: (1) converte la stringa delle ore, dei minuti e dei secondi in ore, minuti e secondi "a.m. e p.m." e (2) li mostra sullo schermo. L'Inserimento di queste due funzioni diverse in una routine non è la preparazione ideale di un programma. La separazione delle funzioni di conversione e di display renderebbe più facile lo sviluppo futuro di altri programmi. Per esempio, se una routine convertisse le ore dal formato TIME$ a quello "a.m. e p.m.", mentre una seconda si occupasse del display vero e proprio, la routine di conversione potrebbe essere chiamata da quella che determina l'orario (Figura 3.6) così da poter mostrare l'ora esatta da inserire. Si possono poi immaginare altre funzioni adatte a questa routine di conversione (Per esercitarvi potreste eseguire questa separazione).

La Figura 3.4 mostra la routine di controllo di un evento particolare. Per capire questa routine dovrete prima comprendere a fondo il concetto di "ora Julian", questo è il numero di secondi trascorsi dall'ultima mezzanotte che si possono assegnare ad ogni secondo. Per esempio, se ora sono le 10 p.m. (cioè le 22) e voi desiderate fissare l'allarme alle 1 a.m., allora l'ora Julian assegnata all'allarme è 90.000, poichè tale è il numero di secondi trascorsi nelle 25 ore passate dall'ultima mezzanotte fino all'ora in cui desiderate far suonare l'allarme. Quando arriva la mezzanotte, due ore dopo aver fissato l'allarme, la base per l'ora Julian cambia. Dunque l'allarme deve essere spostato a 3600, poichè questo è il numero dei secondi in una ora tra l'ultima mezzanotte e l'ora in cui l'allarme deve suonare.

La routine di controllo effettua questo cambiamento sottraendo 86,400 (il numero di secondi in un giorno) da ciascun orario di allarme quando l'orologio passa la mezzanotte.

L'uso di questa ora Julian permette al programma di distinguere tra orari diversi a distanza di 24 ore. Per esempio, se poniamo che l'allarme debba suonare alle 9 a.m. e alle 8.59 a.m. decidiate di usare uno dei comandi. Forse il vostro orologio era un po' lento e allora avete usato il comando F per farlo andare più veloce, ma che cosa succederebbe se ci aveste impiegato un po' e adesso sono già le 9.01 a.m. La routine

che controlla gli eventi viene chiamata e si accorge che le 9.01 a.m. (cioè 32.460) è dopo l'orario d'allarme (32.400 per le 9.00 a.m.) dunque lo spegne. La routine sa per certo che l'ora di allarme sono le 9.00 a.m. di oggi e non di domani, perchè altrimenti queste ultime avrebbero un'ora Julian di 118.800.

Per far sì che questo funzioni, la routine deve essere in grado di sapere che la

```
#Posizionare il cursore per il display dell'orario

positime    LL = TL: CC =TC                    #linea dell'orario e co-
                                               lonna
            GOSUB cursor
            RETURN

#Mostrare l'orario
            gosub hms                          #ottenere l'ora, i minuti
                                               e i secondi
            HR = VAL(HR$)
            IF HR > 11 THEN
              AP$ = "PM"
            else
              AP$ = "AM"
            IF HR = 0 THEN
              HR$ = "12"
            else IF HR > 12 THEN
              HR$ = STR$ (HR-12)
            else
              HR$ = RIGHT$(" " + HR$,2)
            PRINT HR$; ":"; MN$; ":"; SC$; " "; AP$;
            RETURN

240   LL=TL:CC=TC:GOSUB 930:RETURN
250   GOSUB 1050:HR=VAL(HR$)
260   IF HR > 11 THEN AP$="PM":GOTO 280
270   AP$="AM"
280   IF HR=0 THEN HR$="12":GOTO 310
290   IF HR > 12 THEN HR$ = STR$ (HR-12):GOTO 312
300   HR$=STR$(HR)
310   HR$=RIGHT$(" " + HR$,2)
320   PRINT HR$;":";MN$;":";SC$;" ";AP$;
330   RETURN
```

Queste routines mostrano l'orario nel formato "HH:MM:SS AP" in una posizione determinata sullo schermo. L'orario viene aggiustato usando uno spazio per le ore da 1 a 9.

Figura 3.3 — Display dell'orario dell'Orologio

```
#Controllare l'allarme, l'aggiustamento dell'ora ed altri "eventi"

events      GOSUB julian                          #ottenere l'orario "Julian"

            ET = JT
            IF ET < OE THEN {                     #la mezzanotte è appena passata

              IF AJ < > -1 THEN
                AJ = AJ - dayseconds
              IF AL < > -1 THEN
                AL = AL - dayseconds
            }
            IF OE < > ET THEN {                   #controllare gli eventi una volta ogni scatto

            OE = ET
              IF AL < > -1 AND AL < = ET THEN {
                GOSUB setoff                      #far scattare l'allarme
                AL = AL + dayseconds              #fissare allo stesso orario domani
                }

              JF AJ < > -1 AND AJ < = ET THEN {
                DT = AI                           #cambiare l'orario di A1 secondi

                GOSUB bumptime
                AJ = AJ + AF                      #fissare l'orario del prossimo aggiustamento
              }
            }
            RETURN
```

```
340   GOSUB 830:ET=JT
350   IF ET>=OE THEN 380
360   IF AJ<>-1 THEN AJ=AJ-86400
370   IF AL<>-1 THEN AL=AL-86400
380   IF OE=ET THEN 440
390   OE=ET
400   IF AL=-1 OR AL>ET THEN 420
410   GOSUB 610:AL=AL+86400
420   IF AJ=-1 OR AJ>ET THEN 440
430   DT=AI:GOSUB 810:AJ=AJ +AF
440   RETURN
```

Questa routine tiene conto di quando sia necessario effettuare un aggiustamento d'orario, o quando un allarme debba suonare. Questi orari sono conservati in un orario "Julian" che può essere fissato (teoricamente) con diversi giorni d'anticipo.

Figura 3.4 — Controllo degli "eventi" per l'Orologio

mezzanotte è già passata. Questo viene effettuato tramite la variabile 0E, che registra l'ora dell'ultimo controllo eventi. Se la routine di controllo eventi si accorge che l'ora Julian attuale è minore dell'ora registrata all'ultimo controllo, allora sa che è già passata la mezzanotte.

L'altra ragione per ricordare l'ora dell'ultimo controllo, è per essere sicuri che il controllo abbia luogo ad ogni scatto. La routine controlla 3 eventi: (1) se la mezza-

```
#Aggiustare l'orologio di modo che vada più veloce

faster      repeat
              INPUT "ADJUSTMENT INTERVAL (SEC)"; FT
              until (FT > 0 and FT is a whole number)
            IN = 1
            GOSUB setadjust
            RETURN
#Aggiustare l'orologio di modo che vada più lento
slower      repeat
              INPUT "ADJUSTMENT INTERVAL (SEC)"; FT
              until (FT > 0 and FT is a whole number)
                IN = – 1
            GOSUB setadjust
            RETURN
#Determinare l'aggiustamento
setadjust   AI = IN                          #fissare l'aggiustamento
            AF = FT                          #e la frequenza dello stesso
            GOSUB julian
            AJ = JT + AF
            RETURN

450   INPUT "ADJUSTMENT INTERVAL (SEC)";FT
460   IF FT<=0 OR FT <> INT(FT) THEN 450
470   IN=1:GOSUB 510:RETURN
480   INPUT "ADJUSTMENT INTERVAL (SEC)";FT
490   IF FT<=0 OR FT<>INT(FT) THEN 480
500   IN=–1:GOSUB 510:RETURN
510   AI=IN:AF=FT:GOSUB 830:AJ=JT+AF:RETURN
```

Queste routines implementano i comandi dell'aggiustamento della velocità F (per "faster" – più veloce –) e S (per "slower" – più lento –). Il programmatore può richiedere che un secondo venga aggiunto o sottratto dall'orario con frequenza regolare (per esempio ogni 10.000 secondi). Questo può compensare un circuito non accurato.

Figura 3.5 – Comandi di aggiustamento della velocità dell'Orologio

notte è già passata, (2) se l'allarme deve suonare, (3) se è necessario fare un aggiustamento. Se l'allarme deve suonare, suona, e la variabile AL viene aumentata del numero di secondi in un giorno. Cioè l'allarme automaticamente suonerà l'indomani alla stessa ora. Se si renderà necessario aggiustare l'ora, lo fa e registra anche l'ora dell'aggiustamento. Notate se l'orario del prossimo aggiustamento viene fissato aggiungendo la frequenza d'aggiustamento all'orario in cui viene fatto. Cioè se l'aggiustamento viene fatto con un minuto di ritardo, perchè il programma era occupato nell'esecuzione di un comando, allora il prossimo aggiustamento avrà luogo allo stesso orario in cui sarebbe stato fatto se il primo aggiustamento non fosse stato in ritardo. Inoltre, nel caso in cui non si facessero gli stessi aggiustamenti, sia perchè il programma sta eseguendo un comando per un periodo di tempo lungo, o perchè gli aggiustamenti sono molto frequenti, essi saranno eseguiti tutti quando il programma ha finito di essere occupato.

Nella Figura 3.5 vediamo le routine che implementano i comandi F e S che regolano gli aggiustamenti. Esse ne determinano anche la frequenza (AF) e l'aumento (AI) e poi determinano l'ora (AJ) del primo aggiustamento.

Nella Figura 3.6 vediamo la routine che si occupa del comando T che determina l'ora. Notate che l'operatore deve schiacciare il tasto RETURN per far funzionare l'orologio. Ne capirete la ragione se avete cercato di far funzionare l'orologio del Pet con un dialogo del tipo:

```
TIME$ = 120430
```

```
#Fissare l'orario

settime     PRINT "SET TIME"
            GOSUB asktime
            PRINT "PRESS RETURN TO START CLOCK"
            GOSUB onech
            GOSUB newtime                    #TM$ diventa il nuovo o-
                                             rario
            RETURN

520   PRINT "SET TIME"
530   GOSUB 640:PRINT "PRESS RETURN TO START CLOCK"
540   GOSUB 1030:GOSUB 1080:RETURN
```

Questa è la routine che implementa il comando T (per "set time" — fissare l'orario).

Il programma richiede un secondo RETURN per far funzionare l'orologio invece di usare il RETURN che si trova alla fine della stringa dell'input dell'orario. Questo sistema evita che il programmatore scopra degli errori di sintassi al momento sbagliato.

Figura 3.6 — Fissare l'orario nell'Orologio

Dopo aver pazientemente atteso che siano le 12 4 minuti e 30 secondi, schiacciate il tasto RETURN e il Pet risponderà:

TYPE MISMATCH ERROR (AVETE COMMESSO UN ERRORE)

perchè vi siete dimenticati di inserire la stringa in virgolette. L'approccio usato nella Figura 3.6 richiede che l'operatore prema il tasto RETURN una volta in più, ma il secondo RETURN farà certamente partire l'orologio, così non ci saranno sorprese.

```
#Fissare l'allarme

setalarm    PRINT "SET ALARM"
            GOSUB asktime
            AL = TM
            GOSUB julian
            IF AL < JT THEN
              AL = AL + dayseconds
            RETURN
#Cancellare l'allarme
killalarm   AL = -1
            GOSUB turnoff
            RETURN
#Rifissare l'allarme
resetalarm  GOSUB turnoff
            RETURN


550  PRINT "SET ALARM"
560  GOSUB 640:AL=TM:GOSUB 830
570  IF AL<JT THEN AL=AL+86400
580  RETURN
590  AL=-1:GOSUB 630:RETURN
600  GOSUB 630:RETURN
```

Queste tre routines implementano i comandi relativi all'allarme A (per "set alarm" – fissare l'allarme), Q (per "turn off the alarm and don't reset it – cancellare l'allarme e non rifissarlo) e R (per "turn off the alarm and reset it" – cancellare l'allarme e rifissarlo). La routine che fissa l'allarme accetta un orario di 24 ore, poi fissa l'allarme non appena questo orario si ripresenta. Il comando Q fa sì che AL sia fissato a – 1, così da segnalare al programma che non c'è nessun allarme. Il comando R non ha nessun effetto su AL. Nella routine illustrata nella Figura 3.4, AL è automaticamente fissato così da suonare ancora dopo 24 ore. Il meccanismo AL potrebbe far suonare l'allarme anche dopo più di 24 ore, ma non esiste nessun comando per questa operazione nel programma attuale.

Figura 3.7 – Comandi per l'allarme nell'Orologio

Nella Figura 3.7 vediamo le routines che incrementano i comandi A, Q, e R che si riferiscono all'allarme. Queste routines sono abbastanza primitive e, come risultato, le capacità di allarme di questo orologio saranno minime. Alla fine del paragrafo vi illustreremo alcuni miglioramenti che si possono apportare.

Nella Figura 3.8 vediamo le routines che si occupano del display degli allarmi. Come potete vedere si possono certamente migliorare.

Nella Figura 3.9 viene illustrata la routine che chiede l'input dell'ora (sia per aggiustare l'orologio, che per inserire l'allarme). Questa routine ha bisogno di un input che abbia esattamente lo stesso formato di quello del Pet per TIME$, con la sola eccezione che questa routine accetta la stringa dell'ora con o senza virgolette. Nelle Figure 3.7, 3.8 e 3.9 vengono illustrati degli esempi di matrici, un concetto che nasce dalla tecnica di sviluppo dei programmi chiamata "sottosopra". Questo tipo di tecnica richiede che la struttura principale del programma sia stesa per prima (come nella Figura 3.1), poi ciascuna delle subroutines che la compongono viene stesa e formata dalle altre subroutines che la compongono. (Per esempio, le Figure 3.2, 3.3 e 3.4 mostrano le subroutines principali che sono state chiamate dalla routine principale illustrata nella Figura 3.1). Se via via durante questo processo una delle subroutine necessarie non deve essere scritta immediatamente, si fornisce una matrice. Questa non è altro che una routine che agirà positivamente quando chiamata, cioè o imple-

```
#Far funzionare l'allarme

setoff      LL = AR: CC = AC          #linea dell'allarme e colonna

            GOSUB cursor
            PRINT AL$;                #stampare la stringa dell'allarme

            RETURN
#Cancellare l'allarme
turnoff     GOSUB clearscreen
            RETURN

610   LL=AR:CC=AC:GOSUB 930
620   PRINT AL$;:RETURN
630   GOSUB 920:RETURN
```

Queste routines forniscono il display dell'allarme. La prima mostra una stringa predefinita AL$ ad una posizione predeterminata dello schermo. La seconda pulisce lo schermo. Un display in movimento (per esempio un lampeggio o, se il sistema è dotato di suono, un allarme sonoro) richiederebbe un approccio completamente diverso, e coinvolgerebbe la routine di controllo degli eventi illustrata nella Figura 3.4.

Figura 3.8 — Display dell'allarme nell'Orologio

menterà l'azione necessaria, o compirà un'azione semplice. Per esempio, la routine del display dell'ora alla linea 250 (nella Figura 3.3) potrà essere data dalla seguente matrice:

```
PRINT"10:25:04AM";
RETURN
```

O la routine che processa i comandi (illustrata alla Figura 3.2) potrebbe essere data in versione matrice che si occupa solo del comando T.

Il termine "matrice" è stato introdotto per poter descrivere queste routines, perchè la struttura di un programma che è stato sviluppato usando delle tecniche di struttura "sottosopra" è come quella di un albero al contrario, usato per descrivere i quadri organizzativi di molte ditte. Una matrice è spesso ad un livello alto nella struttura di un programma, così se ci si ferma nella compilazione del quadro organizzativo, è come se si lasciasse una sola matrice.

L'uso di matrici ha però un punto a suo sfavore. Talvolta una routine che inizia come matrice viene poi trattata come un ramo a parte. Cioè le caratteristiche e le funzioni in più che vanno aggiunte più non tardi non saranno mai inserite.

Nella Figura 3.10 vediamo le routines che implementano i comandi C e Z, che per-

```
#Chiedere l'orario

asktime     repeat {
              INPUT "TIME (6 DIGITS: HHMMSS)"; TM$
              AH = VAL(LEFT$(TM$,2))
              AM = VAL(MID$(TM$,3,2))
              AS = VAL(RIGHT$(TM$,2))
            } until (0<=AH<24 AND 0<=AM<60 AND 0<=AS<60)
            TM=3600*AH + 60*AM + AS
            RETURN

640   INPUT "TIME (HHMMSS)";TM$
650   AH=VAL(LEFT$(TM$,2)):IF AH<0 OR AH>=24 THEN 640
660   AM=VAL(MID$(TM$,3,2)):IF AM < 0 OR AM>=60 THEN 640
670   AS=VAL(RIGHT$(TM$,2)):IF AS<0 OR AS>=60 THEN 640
680   TM=3600*AH+60*AM+AS:RETURN
```

Questa routine permette alla persona che usa il programma di fissare un orario per l'allarme o per l'orologio. Accetta lo stesso formato necessario per fissare TIME$, il che indica che la routine è stata scritta tenendo presente le esigenze del programmatore e non della persona che userà il programma in seguito.

Figura 3.9 — Input "poco elegante" dell'orario per l'Orologio

mettono il cambiamento dell'ora da effettuarsi aggiungendo o sottraendo un certo numero di ore o di secondi senza dover fermare l'orologio. (Una routine per cambiare l'ora usando un certo numero di minuti non è stato inserito perchè ritenuto superfluo).

Nella Figura 3.11 vediamo la routine che viene chiamata dalle due routines nella Figura 3.10. Il suo scopo è quello di aggiustare l'orario e di aggiustare le variabili AL e AJ se l'aggiustamento dell'orario fa superare o passare all'indietro la mezzanotte. Il cambiamento vero e proprio all'orologio del Pet coinvolge due livelli diversi di subroutines.

```
#Cambiare l'ora

changehour  repeat
              INPUT "CHANGE AMOUNT (HOURS)"; CH
              until (CH is a whole number AND -24 < CH < 24)
            CS = CH*3600                    #numero dei secondi
                                            che vanno aggiunti o
                                            sottratti
            GOSUB changeclock
            RETURN

#Cambiare i secondi

changesec   repeat
              INPUT "CHANGE AMOUNT (SECONDS)"; CS
              until (CS is a whole number AND -60 < CS < 60)
            GOSUB changeclock
            RETURN

690   INPUT "CHANGE AMOUNT (HOURS)";CH
700   IF CH<>INT(CH) OR CH<=-24 OR CH>=24 THEN 690
710   CS=CH*3600:GOSUB 750:RETURN
720   INPUT "CHANGE AMOUNT (SECONDS)";CS
730   IF CS<>INT(CS) OR CS<=-60 OR CS>=60 THEN 720
740   GOSUB 750:RETURN
```

Queste routines implementano i comandi C (per "change time by a speficified number of seconds — aggiustare l'orario aggiungendo o sottraendo un determinato numero di secondi —). Ciascuna di queste routinese calcola il numero totale di secondi che devono essere aggiunti o sottratti, e chiama la routine che modifica l'orario illustrata nella Figura 3.11.

Figura 3.10 — Routines di aggiustamento dell'orario per l'Orologio

Il primo, illustrato nella Figura 3.12 si occupa dell'algoritmo per effettuare il cambiamento "in volo", il livello più basso, illustrato nella Figura 3.15, contiene delle routines che si riferiscono in particolare al meccanismo TIME$ del Pet. Questa separazione delle funzioni rende il programma dell'Orologio facilmente adattabile a questi sistemi che hanno un orologio.

```
#Cambiare l'orario

changeclock GOSUB julian                    #ricordare l'orario prima
                                            del cambiamento
            CT = JT
            DT = CS                         #cambiamento da effet-
                                            tuare
            GOSUB bumptime                  #effettuarlo
            GOSUB julian                    #controllo per passag-
                                            gio della mezzanotte
            IF CS < 0 AND JT > CT THEN {    #all'indietro dopo la
                                            mezzanotte
              IF AJ < > -1 THEN
                AJ = AJ + dayseconds
              IF AL < > -1 THEN
                AL = AL + dayseconds
              }
            IF CS > 0 AND JT < CT THEN {    #avanzato dopo la mez-
                                            zanotte
              IF AJ < > -1 THEN
                AJ = AJ - dayseconds
              IF AL < > -1 THEN
                AL = AL - dayseconds
              }
            RETURN

750  GOSUB 830:CT=JT
760  DT=CS;GOSUB 810:GOSUB 830
770  IF CS>=0 OR JT <=CT THEN 785
775  IF AJ< >-1 THEN AJ=AJ+86400
780  IF AL< >-1 THEN AL=AL+86400
785  IF CS< >=0 OR JT>=CT THEN 800
790  IF AJ< >-1 THEN AJ=AJ-86400
795  IF AL< >-1 THEN AL=AL-86400
800  RETURN
```

Questa routine cambia l'orario e fa sì che sia AL che AJ sono corretti se il cambiamento dell'orario attraversa la mezzanotte.

Figura 3.11 — Le operazioni necessarie al cambiamento dell'orario

Il programma illustrato nella Figura 3.12, cambia l'orario usando un incremento fisso (positivo o negativo) specificato nella variabile DT. Il problema è che l'orologio scatta una volta al secondo, dunque se l'orologio scatta nel momento in cui il programma ottiene l'orario da TIME$ e il momento in cui aggiusta l'orario sul computer, si perderà un secondo. (Capite perchè? Questo è un esempio di un problema significativo nella programmazione di processi che procedono paralleli). La tecnica usata nella Figura 3.12 è di ottenere l'orario corrente da TIME$, poi attendere il prossimo scatto prima di aggiornare TIME$. Questo metodo richiede che tutto il processo di computo occupi meno di un secondo, così l'aggiornamento di TIME$ occorrerà prima del secondo scatto che segue il momento in cui si è ottenuto l'orario originale corrente.

Nella Figura 3.13 vediamo le routines che passano avanti e indietro tra il formato numerico dell'ora Julian e il formato a stringa usato con TIME$. Queste routines sono semplici; tuttavia bisognerà fare attenzione per essere certi che gli orari rappresentanti nei formati a stringa abbiano un certo numero di zeri. Questo si ottiene aggiungendo 1.000.000 al numero che ha bisogno di essere di sei cifre, tenendo così solo le sei ultime.

Notate che nonostante il formato a stringa usato con TIME$ viene usato in tutto il programma, TIME$ viene solo usato nelle routines nella Figura 3.15. Non ci sarebbe

```
#Aumentare l'orario di DT secondi

bumptime    GOSUB julian                    #calcolare JT dall'orario
                                            vecchio (TT$)
            JT = JT + DT + 1                #incrementare il co-
                                            mando, più uno per lo
                                            scatto successivo
            GOSUB unjulian                  #calcolare il nuovo ora-
                                            rio TM$ da JT
            GOSUB waitnew                   #attendere lo scatto se-
                                            guente a TT$
            GOSUB newtime                   #fissare il nuovo orario
                                            da TM$
            RETURN

810   GOSUB 830:JT=JT+DT+1:GOSUB 850
820   GOSUB 1090:GOSUB 1080:RETURN
```

Questa routine esegue il comando di cambiamento dell'orario aumentandolo o diminuendolo di un certo numero di secondi, DT. Per evitare un errore, la routine attende sempre lo scatto successivo prima di effettuare il cambiamento.

Figura 3.12 — La meccanica del cambiamento dell'orario

```
#Calcolare l'ora "Julian"

julian      GOSUB hms
            JT = VAL(SC$) + 60*VAL(MN$) + 3600*VAL(HR$)
            RETURN

#Va dall'ora Julian (JT) al formato stringa (TM$)

unjulian    repeat                                    #aggiustare JT così 0 =
                                                      JT secondi in una gior-
                                                      nata
              IF JT < 0 THEN
                JT = JT + dayseconds
              until (HT >= 0)
            repeat
              IF JT >= dayseconds THEN
                JT = JT - dayseconds
              until (JT < dayseconds)
            SC  = JT mod 60                           #estrarre i secondi
            MN  = (JT - SC)/60 mod 60                 #i minuti
            HR  = (JT - SC - 60*MN)/3600              #le ore
            TX = SC + 100*MN + 10000*HR + 1000000     #metterli insieme pre-
                                                      ceduti da 1
            TM$ = RIGHT$(STR$(TX),6)                  #per assicurare uno ze-
                                                      ro come portante
            RETURN
```

```
830  GOSUB 1050
840  JT=VAL(SC$)+60*VAL(MN$)+3600*VAL(HR$):RETURN
850  IF JT<0 THEN JT=JT+86400:GOTO 850
860  JF JT>=86400 THEN JT=JT -86400:GOTO 860
870  SC=JT-60*INT(JT/60)
880  MN=(JT - SC)/60:MN=MN-60*INT(MN/60)
890  HR=(JT - SC - 60*MN)/3600
900  TX=SC+100*MN+10000*HR+1000000
910  TM$=RIGHT$(STR$(TX),6):RETURN
```

Queste routines traducono il formato a stringa usato da TIME$ nel formato "julian" usato nel controllo degli eventi. Val la pena di ricordare il sistema di aggiungere 1.000.000 ad un numero di sei cifre (linea 900) così da poter essere certi della presenza di zeri portanti nella versione a stringa.

Figura 3.13 — Routines per l'ora Julian

```
#Pulire lo schermo

clearscreen   PRINT "clr";: RETURN

#Posizionare il cursore alla linea (0 <= LL < 24) e alla colonna (0 <= CC < 40)
cursor        LL = LL mod 24: CC = CC mod 40
              PRINT "home";
              IF LL < > 0 THEN
                FOR XX = 1 TO LL
                  PRINT "down cursor";
                  NEXT XX
              IF CC < > 0 THEN
                FOR XX = 1 TO CC
                  PRINT "right cursor";
                  NEXT XX
              RETURN

#Inserire un carattere singolo in X$
onech         repeat GET X$ until (X$ < > ""): RETURN
```

```
 920* PRINT CHR$(147);:RETURN
 930* IF LL<0 THEN LL=LL+24:GOTO 930
 940* IF LL>23 THEN LL=LL-24:GOTO 940
 950* IF CC<0 THEN CC=CC+40:GOTO 950
 960* IF CC>39 THEN CC=CC-40:GOTO 960
 970* PRINT CHR$(19);
 980* IF LL=0 THEN 1000
 990* FOR XX=1 TO LL:PRINT CHR$(17);:NEXT XX
1000  IF CC=0 THEN 1020
1010* FOR XX=1 TO CC:PRINT CHR$(29);:NEXT XX
1020  RETURN
1030* GET X$:IF X$="" THEN 1030
1040  RETURN
```

Queste routines si avvalgono delle caratteristiche del Pet per pulire lo schermo, posizionare il cursore e inserire un carattere singolo. Altri sistemi hanno caratteristiche diverse per queste funzioni.

---

* Nella versione per il TRS-80 di queste routines, la pulizia dello schermo viene effettuata mediante il comando CLS (vedi Figura 2.11); i numeri delle linee e delle colonne sono 16 e 64 invece di 24 e 40); i caratteri "home, down cursor e right cursor" (alla posizione di partenza, cursore verso il basso, e cursore verso destra) sono rappresentati da CHR$(28), e CHR$(25) e CHR$(26); il GET X$ è sostituito da X$=INKEY$ (vedi Figure 2.11 e 2.12).

Figura 3.14 — Routines Utility per l'Orologio del Pet

bisogno di cambiare il formato a stringa usato in tutto il programma, anche se dovesse essere adattato per un sistema il cui orologio usasse un formato interamente diverso. La conversione tra questo formato di programma e quello usato da un orologio di un sistema diverso potrebbe essere eseguita da certe versioni delle routines illustrate nella Figura 3.15.

Nella Figura 3.16 vediamo la routine d'inizializzazione per l'Orologio. Se non si desidera avere un sistema di controllo e di aggiustamento della velocità dell'orologio, bisognerà sostituire la prima costante nella dichiarazione dei data della linea 1150 con −1.

```
#Scegliere l'orario

hms         TT$ = TIME$                          #memorizzarlo
            HR$ = LEFT$(TT$,2)
            MN$ = MID$(TT$,3,2)
            SC$ = RIGHT$(TT$,2)
            RETURN

#Calcolare un nuovo orario da TM$
newtime     TIME$ = TM$
            RETURN

#Attendere lo scatto seguente a TT$
waitnew     repeat    until (TT$ < > TIME$)
            RETURN

1050   TT$=TIME$
1060*  HR$=LEFT$(TT$,2):MN$=MID$(TT$,3,2):SC$=RIGHT$(TT$,2)
1070   RETURN
1080** TIME$=TM$:RETURN
1090   IF TT$=TIME$ THEN 1090
1100   RETURN
```

Queste routines sono le uniche nel programma Clock che usano esplicitamente la funzione TIME$ del Pet. Queste routines dovranno essere adattate se si usa un altro sistema dotato di orologio interno. La prima routine legge l'orologio di sistema e dà le ore, minuti e secondi in HR$, MN$ e SC$. La seconda routine calcola l'orario da TM$. La terza routine aspetta finchè TIME$≠ TT$. TT$ è la fascia su cui si legge TIME$ ad ogni richiamo della prima routine.

---

* Su un sistema TRS-80, le bande HR$ e MN$ sono derivate da MID$ (TT$ 10,2) e MID$ (TT$ 13,2).

** Su un sistema TRS-80, non è possibile calcolare TIME$ in questo modo.

Figura 3.15 — Routine TIME$ per Clock.

Si conclude così la nostra discussione sul programma dell'orologio. Naturalmente si potranno apportare delle modifiche e delle migliorie. Per esempio:

- Incorporare la data. Una volta che al programma sia stata detta la data, la potrà per sempre aggiornare.
- Usare il meccanismo di controllo degli eventi per far scattare un allarme che con-

```
#Routine d'inizializzazione

init        OE = – 1                    #nessun controllo degli
                                        eventi precedente
            AL = –1                     #nessun allarme pre-
                                        fissato
            AL$ = "rvs ALARM off"       #"ALARM" in "video re-
                                        verse"
            READ TL, TC
            DATA timeline, timecolumn
            READ AR, AC
            DATA alarmrow, alarmcolumn
            READ FT, IN                 #controllo della velocità
                                        già esistente
            DATA frequency, increment   #FT = frequenza, IN = 1
                                        per F, –1 per S
            GOSUB setadjust
            RETURN

1110   OE=–1:AL=–1
1120*  AL$=CHR$(18)+"ALARM"+CHR$(146)
1130** READ TL,TC:DATA 22,28
1140   READ AR,AC:DATA 11,17
1150   READ FT,IN:DATA 2900, –1:GOSUB 510
1160   RETURN
```

Questa è la routine d'inizializzazione per l'Orologio. La stringa dell'allarme, le posizioni di display per l'orario e quelle per l'allarme sono già fissate. Poi si determina un controllo della velocità. L'idea è che dopo aver sperimentato l'uso dei comandi F e S, si sia in grado di trovare la posizione corretta e sarà in grado di inserire il controllo della velocità nei comandi del programma. Il Pet dell'autore, per esempio, deve andare più lentamente di circa un secondo ogni 2900 secondi (linea 1150).

* Su di un TRS-80, i caratteri "rvs" e "off" non hanno controparti.

** Su di un TRS-80, la posizione corrispondente sullo schermo per il display dell'orario si otterrà usando TL e TC fissati a 14 e 52.

Figura 3.16 — Inizializzazione per l'Orologio

sista di una serie di azioni – sia di durata fissa, che continua, finchè non viene spento. (Per esempio un lampeggio ad un intervallo di un secondo).

- Degli allarmi multipli. Fate sì che una stringa in grado di identificare un testo sia collegata con ogni allarme (per es. "TIME TO WATCH STAR TREK" – "È ORA DI GUARDARE STAR TREK"). Apportare dei cambiamenti necessari alla definizione e all'uso dei comandi Q e R.
- Far sì che sia possibile fissare degli allarmi più di 24 ore in anticipo.
- Permettere che la frequenza di un allarme sia specificata. (Al momento si riattacca automaticamente, così da suonare ancora dopo 24 ore, a meno che non sia cancellata dal comando Q).
- Inserire degli altri formati di input dell'orario al programma illustrato nella Figura 3.9.
- Rendere possibile il display simultaneo di diversi orari, ciascuno identificato da una stringa, come "PARIS", "MOM'S TIME" o "GMT" (Parigi, l'orario di Mamma, Ora Greenwich).
- Rivedere il programma dell'orologio, così da poter usare il sistema TI del Pet, invece del TIME$.

LE CARTE

Ora esamineremo un altro gioco in cui il fattore tempo gioca un ruolo principale. Le Carte. Si tratta di un gioco semplice in cui il programma mostra una serie di carte (per es. CLUB QUEEN – REGINA DI FIORI – DIAMOND 4 – 4 DI QUADRI –). Inizia col chiedere quante carte il giocatore vuole vedere e a che distanza di tempo tra di loro. Per esempio

```
NUMBER OF CARDS? 5
INTERVAL (SEC)? 3
```

farà sì che il programma mostri una sequenza di 5 carte, scelte a caso. Ciascuna rimarrà sullo schermo per circa 3 secondi e poi scomparirà per lasciare il posto alla carta seguente.

Dopo che tutte le cinque carte sono state mostrate, il programma chiederà al giocatore di ripeterle tutte nell'ordine esatto in cui sono apparse. Ogni volta che il programma chiede "CARD?" il giocatore dovrà inserire un carattere singolo che identifichi il seme (C=FIORI, D=QUADRI, H=CUORI, S=PICCHE), poi uno spazio, e poi una lettera che identifichi il valore (A,K,Q,J= ASSO, RE, DONNA, JACK) o il numero (10, 9, 8, 7, 6, 5, 4, 3, 2,) della carta.

Dopo aver completato questa sequenza, il programma dirà al giocatore quante

carte ha identificato correttamente e darà dei dati riguardanti le partite passate. Per esempio dirà:

NON NE HAI INDOVINATA NESSUNA

IN QUESTO GRUPPO: 2 SU 5
TEMPO MIGLIORE: 3 (INTERVALLO 2 SECONDI)

Questo significa che da quando avete chiesto al programma di darvi delle sequenze di cinque carte con 3 secondi di intervallo fra l'una e l'altra, il punteggio migliore è stato quando avete indovinato 2 carte su 5. In una partita precedente avete chiesto al programma di darvi delle sequenze di 3 carte ad un intervallo di 2 secondi tra l'una e l'altra, e le avete saputе identificare tutte almeno una volta.

A questo punto il programma aspetta un input ad un solo carattere. Se inserite "R" il programma farà una lista dell'intera sequenza per darvi la possibilità di controllare, se invece scrivete "N", il programma vi darà la possibilità di inserire dei nuovi valori riguardanti il numero di carte che volete indovinare ed il loro intervallo. Se schiacciate la sbarretta dello spazio, il programma vi darà una nuova sequenza.

IL PROGRAMMA DEL GIOCO DELLE CARTE

Nella Figura 3.17 vediamo la routine principale per il Gioco delle Carte. La struttura di questa routine riflette la struttura dell'intero gioco.

Una delle routines chiamate dalla routine principale viene illustrata nella Figura 3.18, questa routine simula il mescolare delle carte, ma soltanto le prime carte NC vengono inserite nell'ordine mischiato dall'array DK.

Il mischiare viene compiuto in maniera molto semplice, prima si mettono tutte le 52 carte in un ordine "standard" nell'array DD. Poi un numero intero ZZ, compreso tra 1 e 52, viene scelto a caso. La carta DD (ZZ) diventa il primo elemento dell'array DK. Poi tutti gli elementi di DD che hanno indici maggiori di ZZ vengono spostati di un posto verso l'alto, così che i primi 51 elementi di DD contengano le restanti 51 carte.

Il processo viene poi ripetuto, scegliendo ZZ tra 1 e 51 per ottenere una carta DD (ZZ) e trasformarla in DK (2). Gli elementi con un indice maggiore di ZZ vengono spostati verso l'alto, e l'array DD si riduce a 50 carte, e così via.

Le routines che mostrano le sequenze di carte al giocatore vengono illustrate nella Figura 3.19, sono semplici e non hanno bisogno di spiegazione.

La traduzione vera e propria tra la forma numerica delle carte che codifica un numero tra l'1 e il 52 e la forma che verrà poi mostrata sullo schermo (una stringa del tipo "SPADE 6" — "6 DI PICCHE") ha luogo nelle routines nella Figura 3.20; la codifica da un numero dall'1 al 13 alle carte che faranno parte dei fiori, dal 14 al 26 a quelle che faranno parte di quadri, dal 27 al 39 a quelle che faranno parte dei cuori e dal

```
#Prova di memoria per Le Carte
            GOSUB init                          #inizializzare
            repeat {                            #loop per un gioco diverso
              GOSUB clearscreen
              GOSUB gamevalues                  #intervallo nuovo e nuova lunghezza di sequenza
              repeat {                          #inizializzazione per questo gioco
                GOSUB clearscreen
                GOSUB startgame                 #mischiare le carte
                GOSUB shuffle                   #mostrare la sequenza
                GOSUB showcards                 #chiedere al giocatore di ripeterla
                GOSUB akplayer                  #dire al giocatore quante carte ha indovinato
                GOSUB stats
                repeat {
                  GOSUB next                    #finire il gioco e procedere
                  IF NX$ = "R" THEN
                GOSUB showall
                  } until (NX$ <> "R")
              } until (NX$ = "N" or "E")        #N = fissare dei nuovi parametri
            } until (NX$ = "E")                 #E = ritornare al basic
            END
```

```
100   GOSUB 960
110   GOSUB 860:GOSUB 920
120   GOSUB 860:GOSUB 910:GOSUB 180:GOSUB 240
130   GOSUB 390:GOSUB 660
140   GOSUB 830:IF NX$="R" THEN GOSUB 270:GOTO 140
150   IF NX$="N" THEN 110
160   IF NX$="E" THEN END
170   GOTO 120
```

Il giocatore ha diritto a scegliere una lunghezza di sequenza NC compresa tra 1 e 52 e un intervallo tra ogni carta di DC. L'intervallo è un numero intero di secondi maggiore od uguale a zero. Poi al giocatore vengono mostrate delle sequenze di carte. Ogni carta viene cancellata dallo schermo dopo un intervallo di DC. Infine il programma chiede al giocatore di ripetere la sequenza. Dopo ciascuna carta che il giocatore inserisce, il programma mostra quella corretta.

Figura 3.17 — Prova di memoria per il gioco delle Carte

40 al 52 alle picche. Entro ogni blocco di 13 numeri, i valori delle carte vengono assegnati in quest'ordine: asso, due, tre, quattro, cinque, sei, sette, otto, nove, dieci, jack, donna e re.

Nella Figura 3.21 vediamo la routine che chiede al giocatore di ripetere la sequenza già mostrata. Il tentativo d'indovinare del giocatore viene accettato e controllato. Se il giocatore inserisse una "Q" allora verranno mostrate tutte le carte. Altrimenti il tentativo d'indovinare viene paragonato alla risposta esatta, e il programma prende nota del fatto che sia esatto od errato. Dopo di che, il programma mostra la risposta esatta, pulisce lo schermo e domanda la carta seguente. La routine nella Figura 3.21 chiama le routines nella Figura 3.22 e 3.23 per l'input della carta e per il suo controllo. Questi programmi si spiegano da soli.

```
#Mischiare le carte

shuffle     FOR XX = 1 TO 52                    #prima di tutto mettere
                                                le carte in ordine
              DD(XX) = XX
              NEXT XX
            FOR XX = 1 TO NC                    #poi sceglierne a caso
              YY = 53 - XX                      #numero di carte rima-
                                                ste
              ZZ = INT(RND(1)*YY) + 1           #1 <= ZZ <= YY
              DK(XX) = DD(ZZ)                   #carta seguente
              IF XX < > 52 AND ZZ < YY THEN
                FOR WW = ZZ TO YY - 1           #riempire lo spazio la-
                                                sciato libero da
                  DD(WW) = DD(WW + 1)           #carta scelta
                  NEXT WW
              NEXT XX
            RETURN

180   FOR XX=1 TO 52:DD(XX)=XX:NEXT XX
190   FOR XX=1 TO NC:YY=53 - XX
200*  ZZ=INT(RND(1)*YY)+1:DK(XX)=DD(ZZ)
210   IF XX=52 OR YY<=ZZ THEN 230
220   FOR WW=ZZ TO YY - 1:DD(WW)=DD(WW+1):NEXT WW
230   NEXT XX:RETURN
```

Questa subroutine determina i primi elementi NC dell'array DK per discernere i numeri interi scelti a caso dalla gamma da 1 a 52.

* In the TRS-80 version of this line, the term RND(0) appears instead of RND(1).

Figura 3.18 — Subroutine Shuffle per Le Carte

Nella Figura 3.24 vediamo la routine che annuncia il risultato (cioè se i vostri tentativi sono stati esatti). Come con altri programmi simili in questo libro, questa routine è stata progettata così da fornire varietà al gioco e da renderlo interessante. Nella Figura 3.25 vediamo la subroutine che viene chiamata dal programma nella Figura 3.24 così da mostrare sullo schermo i risultati migliori di partite precedenti.

```
#Mostrare le prime carte NC ad un intervallo di DC

showcards   FOR I = 1 TO NC
              CD = DK(I): GOSUB displaycard
              DL = DC: GOSUB delay
              GOSUB clearscreen
              NEXT I
            RETURN

#Mostrare tutte le carte

showall     FOR I = 1 TO NC
              CD = DK(I): GOSUB displaycard
              NEXT I
            RETURN

#Mostrare una carta CD

displaycard GOSUB decode                    #convertire CD in una
                                            stringa
            PRINT CD$
            RETURN

240   FOR I=1 TO NC
250   CD=DK(I):GOSUB 290:DL=DC:GOSUB 870:GOSUB 860:NEXT I
260   RETURN
270   GOSUB 860:FOR I=1 TO NC:CD=DK(I):GOSUB 290
280   NEXT I:RETURN
290   GOSUB 310:PRINT CD$
300   RETURN
```

Le prime due routines che vediamo qui sono usate per mostrare un intero set di carte per ogni partita. La prima le mostra una alla volta, pulendo lo schermo dopo un determinato intervallo di tempo. La seconda le lascia tutte sullo schermo così il giocatore le può studiare. La terza routine, chiamata da entrambe, traduce un numero compreso tra 1 e 52 in una carta (per esempio 6 di PICCHE) e la mostra sullo schermo.

Figura 3.19 — Routines del display delle carte

Nella Figura 3.26 vediamo diverse routines che abbiamo usato in altri programmi qui descritti. Le prime due implementano la routine che viene chiamata dopo il punteggio del gioco. La terza serve alla pulizia dello schermo.

Nella Figura 3.27 vediamo una routine che attende finchè il tempo (TIME$) che era stato fissato è cambiato di DL secondi. Un'alternativa a questa routine appare

```
#Routine di codifica/decifrazione

decode      FV = mod(CD,13)                    #valore della carta
            SU = (CD - FV)/13 + 1              #seme
            IF FV = 0 THEN
               {FV = 13:SU = SU - 1}
            FV$ = FV$(FV):SU$ = SU$(SU)        #formare stringhe
            CD$ = SU$ + " " + FV$              #far diventare carte
            RETURN

encode      IF NOT (1 <= SU <= 4
            AND 1 <= FV <= 13) THEN
              CD = -1                          rendere nullo il seme o il
                                               valore
            else
              CD = 13*(SU - 1) + FV
            RETURN
```

```
310  FV=CD-INT(CD/13)*13
320  SU=(CD-FV)/13+1
330  IF FV=0 THEN FV=13:SU=SU-1
340  FV$=FV$(FV):SU$=SU$(SU)
350  CD$=SU$+" "+FV$:RETURN
360  IF SU<1 OR SU>4 OR FV<1 OR FV>13 THEN CD=-1:RETURN
370  CD=13*(SU-1)+FV
380  RETURN
```

La prima delle due routines qui illustrate traduce un numero intero compreso tra 1 e 52 in un seme (fiori, quadri, cuori e picche) e in un valore di una carta (Asso, Re, Regina, Fante, 10, 9, 8, 7, 6, 5, 4, 3, 2). I semi sono codificati nella variabile SU come segue: 1=fiori, 2=quadri, 3=cuori, 4=picche. I valori delle carte sono codificati in FV come segue: 1+Asso, e dal 2 al 10 si codificano da soli, 11=Fante, 12=Regina e 13=Re.

La seconda delle due routines va da SU e FV ad un numero CD compreso tra 1 e 52.

Figura 3.20 — Routines Decode/Encode per il gioco delle Carte

```
#Chiedere al giocatore di ripetere la sequenza appena mostrata

askplayer    PRINT "REPEAT CARDS IN ORDER"
             FOR I = 1 TO NC
               repeat {                          #ottenere il tentativo del
                                                  giocatore
                 GOSUB askcard                    #CD$ è la risposta del
                                                  giocatore
                 GOSUB cardnumber                 #convertire in CD
               } until (CD < >-1)                 #-1 = input errato
               IF CD = -2 THEN                    #il giocatore si è ritirato
                                                  (Q)
                 {GOSUB showall: RETURN}          #mostrare tutte le carte
               else IF CD = DK(I) THEN
                 GOSUB right                      #il giocatore ha indovi-
                                                  nato
               else {
                 GOSUB wrong                      #il giocatore ha sbaglia-
                                                  to
                 CD = DK(I)                       #mostrare la carta cor-
                                                  retta
                 }
               GOSUB displaycard                  #mostrare la carta
               DL = SC: GOSUB delay               #(brevemente)
               GOSUB clearscreen
               NEXT I
             RETURN

right        RG = RG + 1: RETURN

wrong        RETURN
```

```
390  PRINT "REPEAT CARDS IN ORDER"
400  FOR I=1 TO NC
410  GOSUB 490:GOSUB 500:IF CD=-1 THEN 410
420  IF CD=-2 THEN GOSUB 270:RETURN
430  IF CD: DK(I) THEN GOSUB 470:GOTO 450
440  GOSUB 480:CD=DK(I)
450  GOSUB 290:DL=SC:GOSUB 870:GOSUB 860:NEXT I
460  RETURN
470  RG=RG+1:RETURN
480  RETURN
```

Questa routine accetta i tentativi d'indovinare del giocatore. Dopo ciascun tentativo (sia esso giusto o sbagliato) appare la carta corretta. Se il giocatore schiaccia "Q" appaiono tutte le carte, e finisce la partita.

Figura 3.21 — Input del giocatore nel gioco delle Carte

nella Figura 3.34. Incidentalmente, la routine illustrata nella Figura 3.27 è la sola in cui viene usata questa caratteristica temporale (TIME$) in tutto il gioco. In un sistema senza orologio, questa funzione potrebbe essere compiuta da un loop che usi la forma:

```
FOR XX = 1 TO N*DL : NEXT XX
```

```
#Domandare una carta al giocatore

#Formato da ottenere: seme (1 carattere), spazio, valore (1 o 2 caratteri).
#                     Il seme è F, Q, C o P
#                     Il valore della carta è A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K.

askard      INPUT "CARD"; CD$
            RETURN

490*  INPUT "CARD";CD$:RETURN
```

Questa routine chiede ed accetta il tentativo d'indovinare del giocatore. Il controllo viene effettuato nella Figura 3.23.

* La versione per l'Apple di questa linea differisce soltanto nel valore della costante di stringa.

Figura 3.22 — Routine Ask-for-Card per il gioco delle Carte

```
500   IF LEN(CD$) >=3 THEN 530
510   IF CD$="Q" THEN CD=-2:RETURN
520   CD=-1:RETURN
530   L$=LEFT$(CD$,1):R$=MID$(CD$,3):VR=VAL(R$)
540   IF L$="C" THEN SU=1:GOTO 590
550   IF L$="D" THEN SU=2:GOTO 590
560   IF L$="H" THEN SU=3:GOTO 590
570   IF L$="S" THEN SU=4:GOTO 590
580   SU=-1
590   IF VR>0 THEN FV=VR:GOTO 650
600   IF R$="A" THEN FV=1:GOTO 650
610   IF R$="K" THEN FV=13:GOTO 650
620   IF R$="Q" THEN FV=12:GOTO 650
630   IF R$="J" THEN FV=11:GOTO 650
640   FV=-1
650   GOSUB 360:RETURN
```

Figura 3.23a — Forma BASIC per l'analisi della risposta del giocatore

Il valore di N può essere determinato provando diverse volte.

Infine, nella Figura 3.28 vediamo tre routines d'inizializzazione per il gioco delle Carte. La prima (linea 960) viene chiamata una volta sola quando il programma inizia. La seconda (linea 920) viene chiamata ogni qualvolta si decidano dei nuovi parametri per il gioco, per esempio, all'inizio e ogni qualvolta un giocatore schiacci "N".

La terza, (linea 910) viene chiamata all'inizio di ogni nuovo display di una sequen-

```
#Tradurre il tentativo d'indovinare del giocatore CD$ in un numero CD

cardnumber  IF LEN(CD$) < 3 THEN                    #i tentativi d'indovinare
                                                    sono di 3 o 4 caratteri
              IF CD$ = "Q" THEN
                CD = -2                             #il giocatore si ritira
              else
                CD = -1                             #tentativo d'indovinare
                                                    non valido
            else {
              L$ = LEFT$(CD$,1)                     #stringa del seme
              R$ = MID$(CD$,3)                      #stringa del valore della
                                                    carta
              VR = VAL(R$)
              IF case
                L$ = "C" THEN SU = 1
                L$ = "D" THEN SU = 2
                L$ = "H" THEN SU = 3
                L$ = "S" THEN SU = 4
              else
                SU = -1                             #seme non valido
              IF VR > 0 THEN                        #R$ è numerico
                FV = VR
              else IF case
                R$ = "A" THEN FV = 1
                R$ = "K" THEN FV = 13
                R$ = "Q" THEN FV = 12
                R$ = "J" THEN FV =11
              else
                FV = -1                             #valore non valido
              GOSUB encode                          #SU, FV vengono com-
              }                                     binati in CD
            RETURN
```

La routine "analizza" CD$ nei campi SU e FV, poi chiama una delle subroutine mostrate in Figura 3.20 per calcolare CD.

Figura 3.23b — Forma in Free BASIC per l'analisi della risposta del giocatore

za. Come per altri giochi che vi abbiamo presentato, vi sono diversi modi in cui si può migliorare il gioco delle Carte. Eccone alcuni:

- Mostrare le carte in figura e non chiamiamole per nome. Sul Pet, questo cambiamento si può effettuare nella Figura 3.19 sostituendo

  GOSUB 310: PRINT CD$

```
#Mostrare il risultato della partita e paragonarlo con il "migliore"

stats     PRINT                                    #saltare una linea
          IF RG = NC THEN                          #il giocatore ha indovi-
                                                    nato tutte le carte
            IF NC = 1 THEN
              PRINT "YOU GOT IT!"
            else IF NC = 2 THEN
              PRINT "YOU GOT BOTH OF THEM!"
            else
            PRINT "YOU GOT ALL"; NC; "OF THEM!"
          else IF RG = 0 THEN                      #il giocatore non ha in-
                                                    dovinato nessuna carta
            IF NC = 1 THEN
              PRINT "YOU MISSED IT!"
            else
            PRINT "YOU DIDN'T GET ANY!"
          else
            PRINT 'YOU GOT"; RG; "OUT OF"; NC
          PRINT                                    #saltare una linea
          IF RG = NC AND NC > BC THEN{
            PRINT "A NEW RECORD LENGHT!"
          BC = NC: BD = DC                         #nuova lunghezza "mi-
                                                    gliore" e intervallo
          }

          IF RG > BR THEN                          #nuovo risultato miglio-
                                                    re per questo gruppo
          BR = RG
          PRINT
          GOSUB saybest                            #elencare i records at-
                                                    tuali
          RETURN
```

Questa routine dice al giocatore il risultato per la sequenza appena terminata. Poi il risultato migliore (cioè la stringa corretta più lunga nel presente gruppo) viene mostrato sullo schermo.

Figura 3.24 — Dare il risultato per il gioco delle Carte

sulla linea 290 con

```
PRINT CP$(CD)
```

dove CP$ è un array di stringhe di figure di carta (usando il cursore del Pet e i suoi

```
660   PRINT:IF RG < > NC THEN 700
670   IF NC=1 THEN PRINT "YOU GOT IT!":GOTO 740
680   IF NC=2 THEN PRINT "YOU GOT BOTH OF THEM!":GOTO 740
690*  PRINT "YOU GOT ALL";NC;"OF THEM!":GOTO 740
700   IF RG < > 0 THEN 730
710   IF NC=1 THEN PRINT "YOU MISSED IT!":GOTO 740
720   PRINT "YOU DIDN'T GET ANY!":GOTO 740
730*  PRINT "YOU GOT";RG;"OUT OF";NC
740   PRINT
750   IF RG < > NC OR NC<=BC THEN 780
760   PRINT "A NEW RECORD LENGHT!"
770   BC=NC:BD=DC
780   IF RG > BR THEN BR=RG
790   PRINT:GOSUB 800:RETURN
```

* La versione per l'Apple di queste linee differisce soltanto nella spaziatura nelle costanti di stringa.

Figura 3.24a — Forma in BASIC per il Risultato del Gioco delle Carte

```
#Mostrare i risultati migliori

saybest     PRINT "BEST FOR THIS SET:"; BR; "OUT OF"; NC
            IF BC > 0 THEN
              PRINT "BEST LENGHT:"; BC;" (DELAY:"; BD; "SEC)"
            RETURN

800*  PRINT "BEST FOR THIS SET:";BR;"OUT OF";NC
810*  IF BC>0 THEN PRINT:PRINT "BEST LENGTH:";BC;" (DE-
      LAY:";BD;"SEC)"
820   RETURN
```

Questa routine mostra i risultati migliori ottenuti finora.

* La versione per l'Apple di queste linee differisce soltanto nella spaziatura nelle costanti di stringa.

Figura 3.25 — Richiamo dei risultati record per il Gioco delle Carte

caratteri grafici). CP$ sarebbe inizializzato nella routine d'inizializzazione alla linea 960 (illustrata nella Figura 3.28). Naturalmente dovreste ancora far sì che entrambi gli inputs Q e R risultino nella lista dei nomi delle carte, non nelle figure. Su di un Apple, potrete usare le capacità grafiche a colori. Occorrerebbe però una riorganizzazione sostanziale e dei nuovi codici.

- Migliorare le routines d'input. Per esempio, potreste inserire un numero indicando che carta volete indovinare (per es. prima, seconda). Potreste anche dare un codice che voglia dire "Mi sono dimenticato questa carta, passiamo oltre".
- Migliorare la capacità del programma di tenere delle statistiche: i cinque punteggi precedenti, le medie, ecc. ecc.
- Sincronizzare il display della prima carta. Al momento, il comando

  DL = DC: GOSUB 870

```
#Alcune routines ben conosciute

next        GOSUB onech                          #fine del gioco, specificare l'azione seguente
            NX$ = X$
            RETURN

onech       repeat {                             #input ad un solo carattere
              X$ = CHR$(RND(1))                  #scegliere a caso, mentre si attende
              GET X$
            } until (X$ < > "")
            RETURN

clearscreen PRINT "clr";                         #pulire lo schermo
            RETURN

830   GOSUB 840:NX$=X$:RETURN
840   X$=CHR$(RND(1)):GET X$:IF X$="" THEN 840
850   RETURN
860   PRINT CHR$(147):RETURN
```

Queste sono le subroutines per accettare un comando ad un solo carattere alla fine della partita, e per svuotare lo schermo. Le versioni qui illustrate sono quelle per il Pet. Quelle per l'Apple e per il TRS-80 sono uguali a quelle illustrate nelle Figure 2.11 e 2.12.

Figura 3.26 — Alcune routines preferite per il Gioco delle Carte

sulla linea 250 (nella Figura 3.19) può causare un ritardo di circa un secondo. Per evitare questo problema, il comando

```
FOR 1 = 1 TO NC
```

sulla linea 240 può essere preceduto da una chiamata alla routine che attende che l'orologio cambi.

```
#Attendere finchè l'orario cambia di DL secondi — Versione per il Pet

delay       DX$ = TIME$
            GOSUB convertsec: DZ = DX
            repeat {
              DX$ = TIME$
              GOSUB convertsec
            } until (DX - DZ >= DL)
            RETURN


convertsec  DX = 3600*VAL(LEFT$(DX$, 2)) +
                 60*VAL(MID$(DX$, 3, 2)) +
                    VAL(RIGHT$(DX$, 2))
            RETURN

870  DX$=TIME$:GOSUB 890:DZ=DX
880  DX$=TIME$:GOSUB 890:IF DX-DZ<DL THEN 880
885  RETURN
890  DX=3600*VAL(LEFT$(DX$,2))+60*VAL(MID$(DX$,3,2))
     +VAL(RIGHT$(DX$,2))
900  RETURN
```

Questa routine controlla l'orario al momento dell'inserimento, poi attende finchè ha cambiato di DL secondi prima di ritornare.

Le versioni per l'Apple e per il TRS-80 devono usare un loop di ritardo ("delay-loop").

I comandi in BASIC per questi sistemi sono:

```
870  FOR DX = 1 TO DL * SS: NEXT DX: RETURN
```

Il valore della variabile SS viene determinato empiricamente e fissato nella routine d'inizializzazione (Figura 3.28).

Figura 3.27 — Routine di ritardo per il Gioco delle Carte

TEN KEY FLICKER

Questo è uno di quei giochi estremamente frustranti in cui il programma vi da un problema ed un certo tempo determinato per rispondere. Se non trovate una soluzione in questo tempo determinato, il programma commenterà e vi eliminerà. Ecco come funziona.

Nella Figura 3.29 vediamo una parte della tastiera del Pet. I numeri che vanno dall'1 al 9 sono posti in un quadrato, cosicchè sia facile per il giocatore usarli.

Il programma inizia col chiedere

DELAY INTERVAL?

Voi risponderete con un numero intero, che indicherà il numero di secondi (da ze-

```
#Routines d'inizializzazione

startgame     RG = 0
              RETURN

gamevalues    repeat
                INPUT "NUMBER OF CARDS"; NC
                until (1 < = NC < = 52)
              INPUT "INTERVAL (SEC)"; DC          #intervallo di tempo fra
                                                  una carta e l'altra
              SC = 1                              #intervallo di tempo per
                                                  la risposta
              BR = 0                              #risultato migliore con
                                                  questo intervallo
              RETURN

init          DIM SU$(4), FV$(13), DK(52), DD(52)
              FOR I = 1 TO 4
                READ SU$(I)
                DATA "CLUB", "DIAMOND", "HEART", "SPADE"
                NEXT I
              FOR I = 1 TO 13
                READ FV$(I)
                DATA "ACE", "2", "3", "4", "5", "6", "7",
                  "8", "9", "10", "JACK", "QUEEN", "KING"
              NEXT I
              BC = 0: BD = 0                      #inizializzare il miglior
                                                  risultato
              RETURN
```

Figura 3.28 — Inizializzazione per il gioco delle Carte

```
 910   RG=0:RETURN
 920*  INPUT "NUMBER OF CARDS";NC:IF NC< 1 OR NC >52 THEN 920
 930*  INPUT "INTERVAL (SEC)";DC
 940   SC=1
 950   BR=0:RETURN
 960   DIM SU$(4),FV$(13),DK(52),DD(52)
 970   FOR I=1 TO 4:READ SU$(I)
 980   DATA "CLUB","DIAMOND","HEART","SPADE"
 990   NEXT I
1000   FOR I=1 TO 13:READ FV$(I)
1010   DATA "ACE", "2", "3", "4", "5", "6", "7", "8", "9", "10", "JACK",
       "QUEEN", "KING"
1020   NEXT I
1030   BC=0:BD=0:RETURN
```

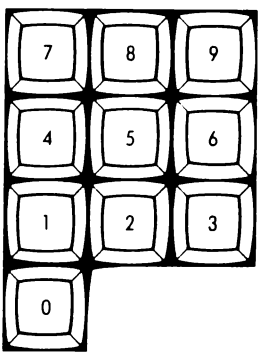
* La versione per l'Apple differisce soltanto nei valori delle costanti di stringa. Inoltre, le versioni per l'Apple e per il TRS-80 hanno un extra linea per fissare il valore di SS (Figura 3.27). La versione per l'Apple è:

```
965  READ SS: DATA 700
```

Quella per il TRS-80 è:

```
965  READ SS: DATA 200
```

Figura 3.28a — Forma in BASIC per l'inizializzazione per il gioco delle Carte

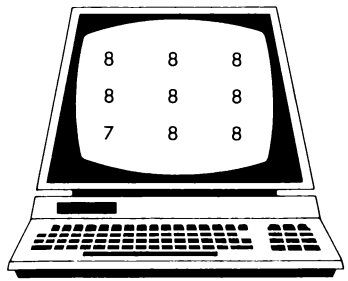


Questa è una porzione della tastiera del Pet. Il gioco Ten-Key Flicker usa il posizionamento a forma di quadrato dei numeri dall'1 al 9 in questa parte della tastiera per permettere al giocatore di indicare la cifra che è diversa dalle altre in un quadrato contenente nove cifre che viene mostrato brevemente sullo schermo.

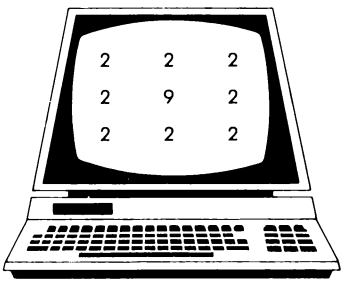
Figura 3.29 — Posizioni dei tasti dei numeri per il Pet

ro in su) che volete che vi siano dati per rispondere. Potrete giocare con un intervallo di zero (nel cui caso sarete certamente sovraumano) o più lungo. Dopo che avrete inserito l'intervallo desiderato, il programma vi mostrerà un disegno di nove numeri in un quadrato sullo schermo. Se tutti i numeri sono uguali, dovrete schiacciare il tasto dello "0" prima che il tempo dell'intervallo sia passato, o se uno dei numeri sarà diverso dagli altri otto, dovrete schiacciare il tasto la cui posizione corrisponde a quella del numero diverso. (Vedi Figura 3.30).

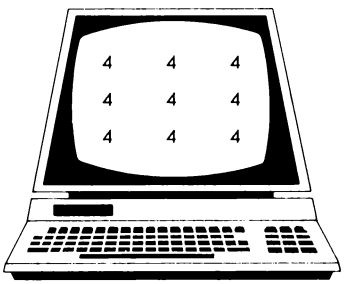
Se riuscite a dare una risposta prima che sia scaduto il tempo massimo, il programma vi dirà "THAT'S RIGHT" — VA BENE — o "THAT'S WRONG" — SBAGLIATO



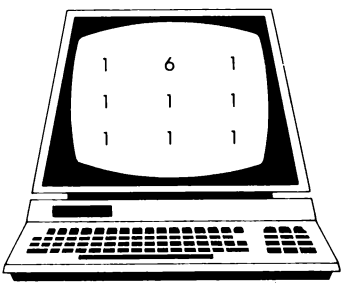
La risposta esatta è 1



La risposta esatta è 5



La risposta esatta è 0



La risposta esatta è 8

Questi sono esempi di displays e di risposte esatte per il Ten-Key Flicker. La risposta esatta è sempre la cifra la cui posizione nella tastiera corrisponde alla cifra diversa sullo schermo.

Figura 3.30 — Esempi di displays per il Ten-Key Flicker

— a secondo che questa sia giusta o sbagliata. Se non fate in tempo a dare una risposta, il programma vi dirà YOU'VE GOT TO BE FASTER — DOVETE ESSERE PIÙ VELOCE —. Se avete sbagliato la risposta, il programma vi dirà quella esatta e vi mostrerà una volta ancora il quadro con tutti i numeri. Questa volta avete tanto tempo a disposizione quanto ne volete, il programma infatti attenderà un vostro input ad un

```
#Ten-Key Flicker

            GOSUB init
            repeat{
              GOSUB setdelay
              repeat{
                GOSUB pattern
                GOSUB show
                GOSUB keyin
                IF XX = 1 THEN
                  PRINT "THAT'S RIGHT!"
                else IF XX = 0 THEN
                  PRINT "THAT'S WRONG! etc."; EX$;
                else                                  #XX = -1
                  PRINT "BE FASTER etc."; EX$;
                DL = 2: GOSUB delay                   #attendere 2 secondi
                GOSUB show                            #poi far riapparire le ci-
                                                      fre sullo schermo
                GOSUB next                            #attendere l'input del
                                                      giocatore NX$
              }until (NX$ = "N" or "E")
            }until (NX$ = "E")
            END

100   GOSUB 430
110   GOSUB 420
120   GOSUB 340:GOSUB 200
130   GOSUB 230:IF XX=1 THEN PRINT "THAT'S RIGHT!":GOTO 160
140   IF XX=-1 THEN PRINT "YOU'VE GOT TO BE FASTER!
      — IT WAS ";EX$;:GOTO 160
150   PRINT "THAT'S WRONG — THE ANSWER IS ";EX$;
160   DL=2:GOSUB 300
170   GOSUB 200:GOSUB 390:IF NX$="E" THEN END
180   IF NX$="N" THEN 110
190   GOTO 120
```

Questa è la routine principale del gioco Ten-Key Flicker. È simile alle routines principali di altri giochi illustrati in questo libro.

Figura 3.31 — Il Ten-Key Flicker

solo carattere — sia esso "N" per fissare un nuovo limite di tempo, o un altro per dare il via ad una nuova partita.

Inoltre, se avete cercato di rispondere schiacciando un tasto, ma eravate appena appena un po' in ritardo, il programma interpreterà la vostra risposta come un segnale d'inizio di un'altra partita. Sarebbe possibile cambiare questo piccolo problema con una semplice variazione, ma questo comportamento dà una certa personalità al programma. (Potrete cambiarlo inserendo GET X$ dopo il GOSUB 200 sulla linea 170 nella Figura 3.31).

La Figura 3.31 illustra la routine principale del Ten Key Flicker. Questa routine segue gli stessi contorni generali di altre che abbiamo già esaminato precedentemente.

Nella Figura 3.32 vediamo la routine di display del quadro, che usa le stringhe L$ e B$. Queste stringhe contengono i caratteri per muovere il cursore del Pet. Sull'Apple, si può ottenere lo stesso display usando i comandi HTAB e VTAB.

Nella Figura 3.33 vediamo la routine che accetta l'input del giocatore mentre controlla il tempo. Questa routine usa la variabile XX per codificare i tre risultati possibili:

1: Risposta esatta

0: Risposta sbagliata

—1: Nessuna risposta nel tempo permesso

```
#Mostrare le cifre

show        PRINT "clr"
            PRINT A(7); L$; A(8); L$; A(9); B$;
                  A(4); L$; A(5); L$; A(6); B$;
                  A(1); L$; A(2); L$; A(3);
            RETURN

200*  PRINT CHR$(147)
210   PRINT A(7) : L$; A(8); L$; A(9); B$; A(4); L$; A(5); L$; A(6); B$;
      A(1); L$; A(2); L$; A(3);
220   RETURN
```

Questa routine mostra le cifre sullo schermo.

* La versione per il TRS-80 usa CLS (vedere Figura 2.12)

Figura 3.32 — Display delle cifre nel Ten-Key Flicker

Nella Figura 3.34 vediamo una routine di ritardo, che ha la stessa funzione della routine nella Figura 3.27. Questa routine conta gli scatti; quella nella Figura 3.27 paragona l'orario.

```
#Accettare l'input del giocatore

keyin       DL = DC
            TX$ = TIME$
            repeat {
              GET X$
              IF X$ < > "" THEN {
                IF X$ = EX$ THEN
                  XX = 1
                else
                  XX = 0
                break
                }
              else if DL < = 0 then
                XX = – 1
                break
                }
              else if TX$ < > TIME$ THEN
                TX$ = TIME$
                DL = DL – 1
                }
              }
            PRINT "clr"
            RETURN

230   DL=DC:TX$=TIME$
240*  GET X$:IF X$<>"" THEN 280
250   IF DL<=0 THEN XX=–1:GOTO 290
260   IF TX$=TIME$ THEN 240
270   TX$=TIME$:DL=DL–1:GOTO 240
280   XX=0:IF X$=EX$ THEN XX=1
290** PRINT CHR$(147):RETURN
```

Questa routine accetta un input di un solo carattere dal giocatore, se avviene entro un tempo limite, cioè prima che l'orologio abbia scattato DC volte.

---

* Nella versione per il TRS-80 di questa linea, GET X$ è stato sostituito da X$=INKEY$. In una versione per l'Apple sarà necessario usare un approccio completamente diverso, poichè il comando GET X$ attende finchè X$<>"".

** Nella versione per il TRS-80, CLS viene usato (vedi Figura 2.12).

Figura 3.33 — Input che il giocatore deve inserire in un tempo limite per il Ten-Key Flicker

Nella Figura 3.35 vediamo la routine che crea il display dei numeri che appariranno sullo schermo. Questa routine sceglie tre numeri a caso nella gamma compresa tra l'1 e il 9:

— il numero "principale" che comparirà in otto posizioni

— la posizione per il numero diverso

— il numero diverso.

Se il numero diverso è lo stesso che quello principale, allora la risposta del giocatore deve essere zero. Se il numero diverso differisce da quello principale, allora il numero del giocatore deve essere lo stesso del secondo scelto a caso.

Nella Figura 3.36 vediamo le routines per accettare il prossimo codice di gioco, e le due routines di inizializzazione. Compresa nella inizializzazione c'è una definizione di funzione:

```
DEF FNR9(X) = INT(RND(1)*9) + 1
```

Questa viene usata dal programma nella Figura 3.35 per scegliere i numeri nella gamma da 1 a 9.

È ovviamente possibile apportare dei miglioramenti al gioco, per esempio:

- Quando si mostra la risposta esatta, mostrare il numero diverso lampeggiante.

```
#Attendere finchè l'orologio non scatta DL volte

delay       IF DL < > 0 THEN
              FOR I = 1 TO DL
                TX$ = TIME$
                repeat{   }until (TX$ < > TIME$)
                NEXT I
            RETURN

300   IF DL=0 THEN RETURN
310   FOR I=1 TO DL:TX$=TIME$
320   IF TX$=TIME$ THEN 320
330   NEXT I:RETURN
```

Questa routine attende che l'orologio scatti DL volte. Questa è un'alternativa al metodo usato nella Figura 3.27.

Figura 3.34 — Routine di tempo per il Ten-Key Flicker

- Aggiungere il suono, così da rendere il programma più antipatico e testardo.
- Sincronizzare l'inizio del display. (Vedere un suggerimento simile con il gioco delle Carte).

TIMER

Questo non è un gioco, ma un timer per il gioco. Per esempio, potrete usarlo per segnare il tempo nelle mosse degli scacchi.

```
#Creare il disegno delle cifre

pattern     N1 = FNR9(1)                    #scegliere tre numeri
                                            interi
            N2 = FNR9(1)                    #compresi tra 1 e 9
            N3 = FNR9(1)
            FOR I = 1 TO 9                  #fissare tutti i 9 numeri
                                            a N1
              A(I) = N1
              NEXT I
            A(N2) = N3                      #poi cambiare l'N2 a N3
            IF N1 = N3 THEN                 #fissare una risposta ad
                                            un solo carattere
              EX$ = "0"                     #che ci si può aspettare
                                            dal giocatore
            else
              EX$ = RIGHT$(STR$(N2), 1)
            RETURN

340* N1=FNR9(1):N2=FNR9(1):N3=FNR9(1)
350  FOR I=1 TO 9:A(I)=N1:NEXT I:A(N2)=N3
360  IF N1=N3 THEN EX$="0":GOTO 380
370  EX$=RIGHT$(STR$(N2), 1)
380  RETURN
```

Questa routine creare il disegno delle cifre che il giocatore deve riconoscere. Sullo schermo apparirà un quadrato di 9 cifre (3 file di 3 cifre ciascuna). Se tutte le cifre sono uguali, il giocatore dovrà rispondere con zero. Altrimenti, ci sarà una cifra diversa dalle altre, e il giocatore dovrà inserire la cifra che si trova nella stessa posizione sulla tastiera di quella diversa sullo schermo.

* La discussione 1 che appare nella chiamata di funzione viene ignorato (Figura 3.36).

Figura 3.35 — Come creare il disegno delle cifre sullo schermo per il Ten-Key Flicker

Il programma (come vediamo nella Figura 3.37) mostra un numero nell'angolo sinistro in alto dello schermo, normalmente è zero. Appena schiacciate la sbarretta

```
#Accettare il codice per la prossima partita

next        GOSUB onech
            NX$ = X$
            RETURN

onech       repeat {
              X$ = STR$(RND(1))
              GET X$
            } until (X$ < > "")

#Chiedere l'intervallo di tempo

setdelay    INPUT "DELAY INTERVAL";DC
            RETURN

#Inizializzare

init        DIM A(9)
            L$ = "sixteen cursor right characters"
            B$ = "one cursor left character followed
                  by ten cursor down characters"
            DEF FNR9(X) = INT(RND(1)*9) + 1        #ignorare la X
            RETURN
```

```
390   GOSUB 400:NX$=X$:RETURN
400*  X$=STR$(RND(1)):GET X$:IF X$="" THEN 400
410   RETURN
420   INPUT "DELAY INTERVAL";DC:RETURN
430   DIM A(9)
440** L$="":FOR LL=1 TO 16:L$=L$+CHR$(29):NEXTLL
450** BS=CHR$(157):FOR LL=1 TO 10:B$+CHR$(17):NEXTLL
460*  DEF FNR9(X)=INT(RND(1)*9)+1
470   RETURN
```

* Nella versione per il TRS-80 di queste linee, il termine RND (0) compare invece di RND(1), e GET X$ viene sostituito da X$=INKEY$.

** Nella versione per il TRS-80 queste linee sono

```
440   L$="":FOR LL=1 TO 27:L$=L$+CHR$(25):NEXT LL
450   B$=CHR$(24):FOR LL=1 TO 6:B$+CHR$(26):NEXTLL
```

Figura 3.36 — Diverse routines per il Ten-Key Flicker

dello spazio, il numero aumenterà una volta al secondo, cioè il programma mostrerà quanti secondi sono passati da quando avete schiacciato la sbarretta. Se la schiacciate ancora, si fermerà. Se la rischiacciate una terza volta, ricomincerà a contare da dove l'avete fermato. Se volete riazzerarla, dovete schiacciare R, mentre non sta contando.

Ci sono diversi modi per migliorare questo programma:

- Usando il formato HH:MM:SS.
- Dargli due timers in due angoli opposti dello schermo. Schiacciando la sbarretta si inizierà un conteggio, poi rischiacciandola si fermerà il primo e si farà partire il secondo.
- Dargli un tempo limite, e far suonare un allarme o farne scattare uno visivo quando si raggiunge questo tempo limite.
- Dare una versione del Timer che usi la caratteristica TI del Pet.
- Integrare l'uso del timer nel programma dell'Orologio di cui abbiamo parlato prima.

SOMMARIO

Lo scopo di questo capitolo è stato di fornire degli esempi di tecniche usate per manipolare l'elemento tempo nei giochi. Abbiamo iniziato con il descrivere l'uso del-

```
100   TX=0:GOSUB 190
110*  GET X$:IF X$="" THEN 110
120   IF X$="R" THEN TX=0:GOSUB 190:GOTO 110
130   IF X$="" THEN GOSUB 150:GOTO 110
140   GOTO 110
150   TX$=TIME$
160*  GET X$:IF X$<>"" THEN RETURN
170   IF TX$=TIME$ THEN 160
180   TX=TX+1:GOSUB 190:GOTO 150
190*  IF TX=0 THEN PRINT CHR$(147):GOTO 200
195*  PRINT CHR$(19)
200   PRINT TX:RETURN
```

* La versione per il TRS-80 usa X$=INKEY$ invece di GET X$. Il PRINT "clr" viene sostituito da CLS e il carattere "home" viene rappresentato da CHR$(28). Una versione per l'Apple necessiterebbe di un approccio completamente diverso perchè il comando GET X$ nell'Apple attende finchè X$ <> "".

Figura 3.37a — Forma in BASIC per il Cronometro

l'orologio nel Pet. Poi abbiamo esaminato quattro giochi diversi che illustrano modi diversi in cui l'elemento tempo può entrare a far parte dei giochi.

L'Orologio illustra le tecniche usate per mantenere un display dell'orario costante mentre si svolgono altre attività. Il gioco delle Carte e Ten Key Flicker illustrano l'uso dell'orologio per poter misurare l'azione a velocità diverse. Il Timer fornisce il modello per un tipo di misurazione "a cronometro".

```
#Cronometro

            TX = 0                              #iniziare il conteggio zero

            GOSUB showcount
            repeat {
              GET X$                            #attendere il comando
              IF X$ = "R".THEN {                #R fa ricominciare il conteggio da zero

                TX = 0
                GOSUB showcount
                }
              else IF X$ = "space" THEN         #lo spazio fa continuare il conteggio

                GOSUB count
              }
count       repeat {
              TX$ = TIME$
              repeat {
                GET X$
                IF X$ < > "" THEN               #qualsiasi comando ferma il conteggio

              RETURN
                }until (TX$ < > TIME$)
              TX = TX + 1                       #aumentare una volta al secondo

              GOSUB showcount
              }
showcount   IF TX = 0 THEN                      #se si ricomincia daccapo

              PRINT "clr"                       #pulire lo schermo
            else
              PRINT "home"
            PRINT TX
            RETURN
```

Questo è il programma che fa funzionare il Cronometro.

Figura 3.37b — Forma in Free BASIC per il Cronometro

CAPITOLO 4

GIOCHI CON LE DATE

I Giorni della Settimana

Il bambino nato di Lunedì ha la pelle chiara,
quello di Martedì è aggraziato,
quello di Mercoledì è sfortunato,
quello di Giovedì andrà lontano,
quello di Venerdì sa amare e donare,
quello di Sabato lavorerà sodo,
e un bambino che è nato di Domenica
sarà onesto, saggio, buono e allegro.

di Autore ignoto; da *"Le Tradizioni del Devon"* di Bray

Ogni programmatore dovrebbe conoscere a fondo le tecniche usate per l'input e l'output delle date. I giochi illustrati in questo capitolo, Compleanno e Calendario, sono stati studiati per spiegare queste tecniche.

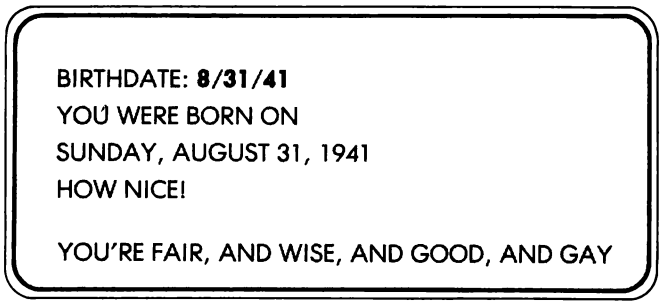
COMPLEANNO

Questo gioco vi insegnerà:

- A determinare il giorno della settimana di qualsiasi data. (Questo sarà utile anche per scopi commerciali).
- A trovare un formato utile per l'input della data.

Il programma inizia con chiedere la vostra data di nascita (Vedi Figura 4.1). Voi rispondete con l'inserire la data nella forma "Mese, giorno, anno", dove il mese, il giorno e l'anno sono rappresentati da numeri separati per mezzo di trattini o spazi. Se l'anno che volete inserire è composto di due cifre, allora il programma aggiungerà autòmaticamente 1900. Nell'esempio usato nella Figura 4.1, il giocatore ha specifi-

cato 1941 inserendo "41". (Se aveste voluto indicare 41A.D., avreste dovuto inserire "041"). Il programma risponde con "TU SEI NATO DI" seguito dal giorno della settimana e dalla data nella forma illustrata nella Figura 4.1. Poi aggiunge un commento



Questo è il display che appare sullo schermo per il gioco del Compleanno. Il programma attende che il giocatore schiacci un tasto, così da pulire lo schermo e ricominciare.

Figura 4.1 — Display per il Compleanno

```
#Compleanno

          GOSUB init
          repeat {
            GOSUB clearscreen              #iniziare con lo schermo vuoto

            DP$ = "BIRTHDATE:"             #discussione per data in
            GOSUB datein                   #chiedere una data
            GOSUB day                      #convertire la data in un giorno della settimana

            GOSUB daypatter                #recitare la poesia ecc.
            GOSUB onech                    #attendere un impulso
            }

100   GOSUB610
110   GOSUB510:DP$="BIRTHDATE:":GOSUB120:GOSUB300:
      GOSUB400:GOSUB520:GOTO110
```

Questa è la routine principale per il Compleanno.

Figura 4.2 — Il Compleanno

che cambia a seconda del giorno della settimana (per es. "MONDAY'S CHILD IS FAIR OF FACE").

Infine, il programma aspetta che il giocatore schiacci qualsiasi tasto della tastiera, e pulisce lo schermo ed è così pronto a ricominciare.

IL PROGRAMMA DEL COMPLEANNO

Il Programma del Compleanno viene mostrato nelle Figure dalla 4.2 alla 4.11. Molte delle subroutine verranno poi usate anche in altri giochi con le date.

La routine principale, illustrata nella Figura 4.2, è abbastanza semplice, e non abbiamo dunque bisogno di commentarla. La sua struttura è simile a quella di altre routines principali di cui abbiamo già parlato in questo libro.

La routine di input della data illustrata nella Figura 4.3 è studiata per uso generico.

Accetta come discussione la stringa di suggerimento che userà quando domanda l'input della data. La routine di esame della stringa della data viene illustrata nella Figura 4.4.. Prima di tutto la routine fissa la lunghezza della stringa nella variabile L. Questa lunghezza viene usata dalla routine che è stata chiamata per dividere la stringa in tre parti. Poi la routine verifica che sia stata inserita una data giusta. Nel caso affermativo, fissa XX a 1 e ritorna il mese, il giorno e l'anno (con l'aggiunta di 1900 se necessario) nelle variabili MO, DA e YR. Se la data inserita non è corretta, la routine ritorna XX = 0.

```
#Input della data — DA, MO, YR (giorno, mese, anno)

datein      repeat {
              PRINT DP$;                    #chiedere una data
              GOSUB stringin                #portare la risposta fino
                                            a RETURN
              GOSUB parsedate               #convertire in DA, MO,
                                            YR
            } until (XX = 1)
            RETURN

120   PRINT DP$;:GOSUB540:GOSUB140:IF XX<>1 THEN 120
130   RETURN
```

Questa è la routine di input della data. Essa accetta la stringa DP$ come discussione, così da poter poi essere chiamata ad accettare delle date per altri scopi. In Compleanno, DP$ viene fissato a "BIRTHDATE:" prima di chiamare questa routine.

Figura 4.3 — Input della data

```
#Trasformare la stringa XX$ in MO,DA,YR (mese, giorno, anno)

parsedate   L = LEN(XX$)
            IF L >= 5 THEN                          #ignorare se è troppo
                                                    corta
              GOSUB fields                          #spezzettare XX$ in
                                                    MO$, DA$, YR$
              MO = INT(VAL(MO$))
              DA = INT(VAL(DA$))
              YR = INT(VAL(YR$))
              IF LEN(YR$) = 2 THEN                  #un anno con due cifre
                                                    significa 19YR
                YR = YR + 1900
              GOSUB leapyear                        #se l'anno è bisestile
              IF XX = 1 THEN
                ML(2) = 29                          #allora Febbraio ne ha
                                                    29
              else
                ML(2) = 28                          #altrimenti 28
              IF 1 <= MO <= 12 AND YR >= 1 THEN     #controllare mese e an-
                                                    no
                IF 1 <= DA <= ML(MO) THEN           #controllare il giorno
                  IF NOT (YR = 1582 AND             #attenzione alla riforma
                                                    del calendario
                  MO = 10 AND 4 < DA < 15) THEN {
                    XX = 1
                    RETURN
                    }
              }
            XX = 0
            RETURN
```

```
140   L=LEN(XX$):IF L<5 THEN210
150   GOSUB220:MO=INT(VAL(MO$)):DA=
      =INT(VAL(DA$)):YR=INT(VAL(YR$))
160   IF LEN(YR$)=2 THEN YR=YR+1900
170   GOSUB480:ML(2) = 28 IF XX = 1 THEN ML(2) = 29
180   IF NOT(1<=MO AND MO<=12 AND YR>=1) THEN210
190   IF NOT (1<=DA AND DA<=ML(MO)) THEN210
200   IF NOT(YR=1582 AND MO=10 AND 4 <DA AND DA<15) THEN
      XX=1:RETURN
210   XX=0:RETURN
```

Questa routine ricerca nella stringa XX$ i tre campi che compongono il mese, il giorno e l'anno di una data posteriore al 1 Gennaio 1 A.D. Questo compito è più complicato di quello che sembra, perchè le date tra il 5 Ottobre 1582 e il 14 Ottobre 1582 non esistono.

Figura 4.4 — Esame della stringa della data

La routine illustrata nella Figura 4.5 spezza la stringa della data in tre segmenti separati da trattini o spazi.

```
#Ottenere MO$, DA$, YR$ da XX$ – MO$ = "" indica un formato errato

fields      XS = 0                                  #numero di separatori
                                                    esistenti
            FOR XL = 2 TO L – 1                     #non si può iniziare o fi-
                                                    nire con un separatore
              IF MID$(XX$,XL,1) =                   #se un carattere fa da
                                                    separatore,
              "/" or "" or "–" THEN
                XS = XS + 1                         #allora bisogna contarlo
                IF XS = 1 or 2 THEN
                  SP(XS) = XL                       #e ricordare dove era
                else {
                  MO$ = ""
                  RETURN
                  }
                }
              NEXT XL
            IF XS < > 2 THEN
              MO$ = ""
            else{                                   #usare i due separatori
                                                    per derivare
              MO$ = LEFT$(XX$, SP(1) – 1)           #tre campi
              DA$ = MID$(XX$, SP(1) + 1,
              SP(2) – SP(1) – 1)
              YR$ = MID$(XX$, SP(2) + 1)
              }
            RETURN

220   XS=0:FOR XL=2 TO L–1:ZZ$=MID$(XX$,XL,1)
230   IF ZZ$<>"/" AND ZZ$<>"" AND ZZ$<>"–" THEN260
240   XS=XS+1:IF XS=1 OR XS=2 THEN SP(XS)=XL:GOTO260
250   MO$="":RETURN
260   NEXT XL:IF XS<>2 THEN MO$="":GOTO290
270   MO$=LEFT$(XX$,SP(1)–1):DA$=MID$(XX$,SP(1)+
      +1,SP(2)–SP(1)–1)
280   YR$=MID$(XX$,SP(2)+1)
290   RETURN
```

Questa routine spezzetta la stringa XX$ in tre campi, separati da trattini o spazi. I tre campi vengono inseriti in tre variabili di stringa: MO$, DA$, YR$. Non viene controllata la validità della data, a parte per il controllo effettuato sull'esistenza dei tre campi nel formato che viene inserito. La routine ritorna MO$="" se il formato è errato.

Figura 4.5 — Spezzettare la stringa della data

```
#Convertire MO,DA,YR in DW – giorno della settimana (0 = Domenica,..., 6 = Sabato)
#Versione Gregoriana – tenere presente il cambiamento di calendario nel 1582

day     YZ = YR mod 400                          #400 anni = esattamente 20.871 settimane
        DW = DZ                                  #1 Gennaio 2.001
        while (YZ >= 101){                       #aggiungere 5 giorni per ogni secolo
          YZ = YZ – 100
          DW = DW + 5
        }
        while (YZ >= 5){                         #aggiungere 5 giorni per ogni 4 anni
          YZ = YZ – 4
          DW = DW + 5
        }
        IF YZ = 0 THEN
          DW = DW – 2                            #2 giorni per l'A.D. 2000 (per esempio)
        else
          DW = DW + (YZ – 1)                     #1 giorno ciascuno per il 2002-2004
        GOSUB julian                             #JD = numero dei giorni in un anno
        DW = DW + JD – 1                         #aggiungere un giorno per ogni giorno
        IF (MO,DA,YR) <= (10,4,1582) THEN
          DW = DW + 3                            #aggiungere 3 per gli 11 giorni mancanti
        DW = DW (mod 7)
        RETURN
```

```
300  YZ=YR–400*INT(YR/400):DW=DZ
310  IF YZ<101 THEN330
320  YZ=YZ – 100:DW=DW+5:GOTO310
330  IF YZ<5 THEN350
340  YZ=YZ – 4:DW=DW+5:GOTO330
350  IF YZ=0 THEN DW=DW–2:GOTO356
352  DW=DW+YZ–1
356  GOSUB450:DW=DW+JD–1:IF YR>1582 THEN390
360  IF YR=1582 AND MO>10 THEN390
370  IF YR=1582 AND MO=10 AND DA>=15 THEN390
380  DW=DW+3
390  DW=DW–7*INT(DW/7):RETURN
```

Questa routine prende ogni data posteriore al 1 Gennaio 1 A.D. e determina il giorno della settimana in cui cade. Questa routine tiene conto del cambio del calendario decretato da Papa Gregorio XIII che fece si che al 4 Ottobre 1582 seguisse il 15 Ottobre 1562.

Figura 4.6 – Determinare il giorno della settimana

La routine che determina il giorno della settimana è illustrata nella Figura 4.6. Nonostante il fatto che ha un codice speciale per le date prima del cambio effettuato nel 1582, la routine dà dei risultati non esatti per le date prima del 1600. Come esercizio potreste cercare di risolvere questo problema che interessa sia la routine di cui sopra che quella descritta nella Figura 4.9. La routine di cui alla Figura 4.6 è più semplice di quello che sarebbe stata se si fosse usata una data base 1 Gennaio 2000, invece di 1 Gennaio 2001. Potreste cercare di immaginare il perchè.

La routine illustrata nella Figura 4.7 fornisce l'elemento gioco. È terribilmente semplice, e consiste interamente di comandi PRINT.

La routine alla Figura 4.8 mostra il computo della data "Julian". Questo è il numero che dà un numero in sequenza da 1 a 365 (o 366 in anni bisestili) per ciascun giorno dell'anno. La routine usa l'array J, che viene determinato dalla routine di inizializzazione illustrata nella Figura 4.11. J ha un inserimento per ogni mese. Il valore di questo inserimento è il numero totale dei giorni dall'uno gennaio fino al primo dei mesi scelti (in un anno non bisestile). Per esempio, J (1) = 0, poichè non ci sono giorni che precedono Gennaio. J(2) = 31, J(3) = 59 (cioè, 31 + 28), e così via.

La routine alla Figura 4.9 determina se un anno sia bisestile o no, e ritorna XX = 1 se l'anno è bisestile, XX = 0 se non lo è. La routine ha però un "problema" che viene

```
#Commentare sulla data di nascita

daypatter    PRINT: PRINT "YOU WERE BORN ON"
             PRINT DW$(DW);","; MO$(MO); DA; ","; YR
             PRINT "HOW NICE!"
             PRINT
             PRINT PM$(DS)
             RETURN

400   PRINT:PRINT "YOU WERE BORN ON"
410*  PRINT DW$(DW);",";MO$(MO);(STR$(DA);",";YR
420   PRINT "HOW NICE!"
430   PRINT
440   PRINT PM$(DW):RETURN
```

Questa è la routine che commenta sulla data di nascita del giocatore. L'array PM$ che è quello dei commenti, viene inizializzato dalle dichiarazioni DATA nella routine d'inizializzazione (Figura 4.11).

* La versione per l'Apple di questa linea differisce soltanto nella spaziatura della costante di stringa che precede la variabile YR.

Figura 4.7 — Commenti del programma sulla data di nascita

causato dal "riscrivere la storia in maniera non esatta". Cercate di trovare e di correggere questo errore. Un cambiamento corrispondente dovrà essere fatto alla routine di cui alla Figura 4.6. (Suggerimento: queste routine trattano il 1500 come se fosse il 1900).

```
#Numero del giorno nel contesto dell'anno: convertire (MO,DA,YR) in JD

julian      JD = J(MO) + DA
            IF MO > 2 THEN {
              GOSUB leapyear
              IF XX = 1 THEN
              } JD = JD + 1
            RETURN

450   JD=J(MO)+DA:IF MO<=2 THEN RETURN
460   GOSUB480:IF XX=1 THEN JD=JD+1
470   RETURN
```

Questa routine determina la data "Julian" nel contesto dell'anno. La variabile JD prende valore da 1 a 365 o 366, iniziando con 1 per il 1 Gennaio, e finendo con 365 o 366 per il 31 Dicembre.

Figura 4.8 — Data Julian

```
#L'anno I/R è bisestile? — XX = 1 se lo è.

leapyear    IF 4 divides YR AND                 #Ogni quattro anni c'è
            (100 does not divide YR OR          #un anno bisestile
            400 divides YR) THEN                #a parte quegli anni che
              XX = 1                            #non sono divisibili
            else                                #per 400.
              XX = 0
            RETURN

480   XX=0:IF YR<>4*INT(YR/4)THEN RETURN
490   IF YR=100*INT(YR/100)AND YR<>400*INT(YR/400)THEN RE-
      TURN
500   XX=1:RETURN
```

Questa routine determina se l'anno è o non è bisestile. Dà come risultato XX=1 se è bisestile, XX=0 se non lo è. Gli anni bisestili cadono ogni quattro anni, a parte quegli anni che non sono divisibili per 400 (per esempio 1700, 1800, 1900 non sono bisestili). Cercate di trovare l'"errore" in questa routine.

Figura 4.9 — Controllo dell'anno bisestile

Nella Figura 4.10 sono illustrate diverse routines di servizio che sono già apparse in versioni identiche o simili in altri programmi in questo stesso libro. La routine di input della stringa che è qui illustrata implementa il comando BASIC.

LINE INPUT XX$

```
#Stringa di input

stringIn    XX$ = ""                          #iniziare con una stringa nulla
            repeat {
              GOSUB onech                     #ottenere il carattere seguente
              IF X$ < > "delete" THEN {       #se non è errato
                PRINT X$;                     #ripeterlo
                IF X$ < > "return" THEN       #se non è un ritorno
                  XX$ = XX$ + X$              #aggiungere alla stringa
                else break                    #finire il ritorno
                }
              else IF LEN(XX$) < > 0 THEN {   #se è errato
                PRINT "delete string";        #cancellarlo dallo schermo
                XX$ = LEFT$(XX$,LEN(XX$) - 1) #e eliminarlo dalla stringa
                }
              }
            RETURN
```

```
540   XX$=""
550*  GOSUB520:IF X$=CHR$(20) THEN580
560   PRINT X$;:IF X$=CHR$(13) THEN600
570   XX$=XX$+X$:GOTO550
580   IF LEN(XX$)=0 THEN550
590** PRINT X$;:XX$=LEFT$(XX$,LEN(XX$)-1):GOTO550
600   RETURN
```

* Nelle versioni per l'Apple e per il TRS-80 di questa linea, il carattere per cancellare è CHR$(8).

** Nella versione per l'Apple, questa linea viene sostituita dalle seguenti:

```
590   PRINT X$;"";X$;:IF LEN(XX$)=1 THEN XX$ = "":GOTO550
595   XX$ = LEFT$(XX$,LEN(XX$)-1): GOTO550
```

Il BASIC per l'Apple non accetta una discussione a lunghezza zero nella funzione LEFT$, e il carattere dell'Apple non cancella quello precedente dallo schermo.

Figura 4.10a — Routines di servizio per il Compleanno

```
#Pulire lo schermo
clearscreen   PRINT "clr";
              RETURN
#Input ad un solo carattere
onech         repeat
                GET X$
                until (X$ < > "")
              RETURN

510   PRINTCHR$(147);:RETURN
520   GET X$:IF X$="" THEN520
530   RETURN
```

Queste routines di servizio sono simili ad altre in questo libro. (Vedi Figure 2.11 e 2.12)

Figura 4.10b — Routines di servizio per il Compleanno

```
#Inizializzazione per il Compleanno

610   READ DZ:DATA 1
620   DIM DW$(6),J(12),MO$(12),ML(12),PM$(6)
630   FOR XX=0 TO 6:READ DW$(XX),PM$(XX):NEXT XX
640   DATA SUNDAY, "YOU'RE FAIR, AND WISE, AND GOOD, AND GAY"
650   DATA MONDAY, "MONDAY'S CHILD IS FAIR OF FACE"
660   DATA TUESDAY, "TUESDAY'S CHILD IS FULL OF GRACE"
670   DATA WEDNESDAY, "WEDNESDAY'S CHILD IS FULL OF WOE"
680   DATA THURSDAY, "THURSDAY'S CHILD HAS FAR TO GO"
690   DATA FRIDAY, "FRIDAY'S CHILD IS LOVING AND GIVING"
700   DATA SATURDAY, "SATURDAY'S CHILD WORKS HARD FOR ITS LI-
      VING"
710   FOR XX=1 TO 12:READ MO$(XX):NEXT XX
720   DATA JANUARY,FEBRUARY,MARCH,APRIL,
      MAY,JUNE,JULY,AUGUST,SEPTEMBER,
730   DATA OCTOBER,NOVEMBER,DECEMBER
740   DT = 0 : FOR XX = 1 TO 12 : J(XX) = DT : READ ML(XX) : DT = DT +
      ML(XX) : NEXT XX
745   IF DT < >365 THEN STOP
750   DATA 31,28,31,30,31,30,31,31,30,31,30,31
760   RETURN
```

Questa è l'inizializzazione per il Compleanno. Le versioni in BASIC e in Free BASIC sono più o meno identiche, dunque abbiamo illustrato solo quella in BASIC.

Figura 4.11 — Inizializzazione per il Compleanno

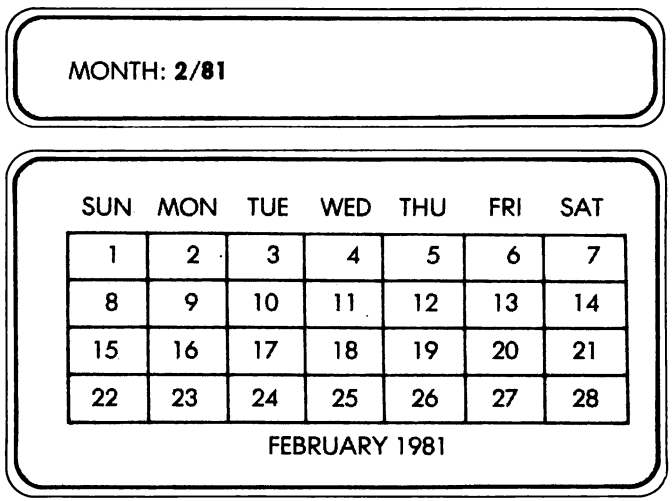
Pochi sistemi di computer non industriali hanno istruzioni LINE INPUT.

Nella Figura 4.11 mostriamo l'inizializzazione per le routines usate in Compleanno.

CALENDARIO

Questo gioco è stato progettato per mostrarvi come le routines sviluppate per il Compleanno possano essere usate in altri programmi. Il gioco del Calendario deriva dal Compleanno usando il processo di "cannibalizzazione" che abbiamo descritto nel Capitolo 2.

Il programma inizia col chiedervi di inserire un mese ed un anno nella stessa forma numerica usata per il Compleanno. Il programma poi risponde producendo un calendario per il mese richiesto (vedi Figura 4.12). Il calendario rimane sullo schermo finchè il giocatore schiacci qualsiasi tasto sulla tastiera. Lo schermo si pulisce e il programma è pronto a ricominciare e richiede un altro mese ed un altro anno.



Questo è il display che appare sullo schermo per il gioco del Calendario. Il giocatore specifica un mese ed un anno particolare ed il programma mostra sullo schermo un calendario per l'intero mese.

Figura 4.12 — Display per il Calendario

IL PROGRAMMA DEL CALENDARIO

Questo programma è stato tratto dal programma del Compleanno aggiungendo i programmi illustrati nelle Figure dalla 4.13 alla 4.17 e cancellando il programma prin-

```
# Calendario

        GOSUB init                              #inizializzazione per il
                                                Compleanno
        GOSUB calinit                           #inizializzazione per il
                                                Calendario
        repeat{
          GOSUB clearscreen                     #cominciare con lo
                                                schermo vuoto
          repeat{
            PRINT "MONTH:";                     #ottenere il mese e l'an-
                                                no ("MM / YY")
            GOSUB stringin
            XX$ = "1/" + XX$                    #sostituire con "1 / MM
                                                / YY"
            L = LEN(XX$)
            GOSUB fields
            XX$ = DA$ + "" + MO$ + "" + YR$     #far diventare "MM 1
                                                YY"
            GOSUB parsedate                     #controllare la validità
            }until (XX = 1)                     #XX = 1 se valida
          GOSUB day: SZ = DW                    #trovare l'inizio
          NL = 1                                #prima linea
          DL = ML(MO) - (7 - SZ)                #contiene 7 - SZ giorni
          while (DL > 0)
            DL = DL - 7:NL = NL + 1             #ogni linea in più prende
            }                                   7
                                                #fino all'ultima
          GOSUB clearscreen: GOSUB frame        #inserire NL linee di ca-
                                                selle
          ND = ML(MO): GOSUB daydisp            #stampare da 1 a ND
          GOSUB onech                           #lasciare sullo schermo
          }                                     finchè non viene schiac-
                                                ciato il tasto

        END
```

Questa è la routine principale per il Calendario. Questa routine usa diverse subroutines prese dal Compleanno.

Figura 4.13 — Il Calendario

cipale (Figura 4.2), la routine di input della data (Figura 4.3), e la routine del commento (Figura 4.7). Il programma del Calendario, a parte questi cambiamenti, è stato lasciato così com'era per la maggior parte. Nonostante quasi tutto il codice d'inizializzazione per il Compleanno (Figura 4.11) sia irrilevante per quanto riguarda il Calendario, è stato lasciato intatto, ed è stata aggiunta una routine di inizializzazione separata per il Calendario.

La routine principale per il Calendario (Figura 4.13) mostra diverse "contorsioni" che sono necessarie perchè le routines del Compleanno sono state usate per scopi leggermente diversi da quelli per i quali sono state progettate. Per esempio, dopo che il programma ha ottenuto un mese ed un anno nella forma "2/81" (cioè Febbraio 1981), prima di tutto converte la data in "1/2/81" (che vorrebbe normalmente significare 2 Gennaio 1981). Poi il programma chiama la routine ilustrata nella Figura 4.5 che spezza la data in tre segmenti: MO$ = "1", DA$ = "2", YR$ = "81".

Questi segmenti vengono poi riposizionati in una nuova stringa nella forma corretta: "2/1/81", cioè Febbraio 1, 1981. Questa stringa viene poi passata alla routine d'esame della stringa della data illustrata nella Figura 4.4 non appena il giocatore ha scritto "2/1/81".

Le routines prese dal Compleanno possono essere usate da qui in avanti. La routine illustrata nella Figura 4.6 viene chiamata per determinare il giorno della settimana in cui cade il 1 Febbraio 1981. Questo fa sì che il calendario possa essere redatto in maniera molto semplice, usando diverse chiamate alle routines illustrate nelle Figure 4.14 e 4.15. La prima di queste routines mostra la cornice del calendario (i giorni, le diverse caselle, e il mese e l'anno). La seconda routine riempie le caselle con le date.

La posizione del cursore fa sì che si possano usare diverse funzioni per il display della cornice e per riempire le caselle con le date. Entrambe le routines illustrate nelle Figure 4.14 e 4.15 chiamano una routine che mette il cursore in posizione per scorrere su una linea e su una colonna specificata nelle variabili LL e CC. (Il "curso-

```
100   GOSUB610:GOSUB900
105   GOSUB510
110   PRINT "MONTH: ";:GOSUB540:XX$ = "1/" + XX$:L = LEN(XX$):GO-
      SUB220
112   XX$ = DA$ + "" + MO$ + "" + YR$:GOSUB140:IF XX < > 1
      THEN110
115   GOSUB300:SZ = DW:NL = 1:DL = 1ML(MO) + SZ - 7
120   IF DL >0 THEN DL = DL - 7:NL = NL + 1:GOTO120
125   GOSUB510:GOSUB130:ND = ML(MO):GOSUB400
128   GOSUB520:GOTO105
```

Figura 4.13a — Forma in BASIC per il Calendario

re" è dunque la posizione alla quale apparirà il prossimo carattere. Di solito si tratta di un quadratino bianco lampeggiante sullo schermo. Questo quadrato viene anch'esso chiamato cursore). La routine illustrata nella Figura 4.15 usa i comandi di posizionamento del cursore per muovere quest'ultimo in successione in ciascuna casella del calendario che sia vuota e che debba ricevere un numero. La stringa di numeri a due cifre viene così stampata, usando il comando PRINT con il finale a "punto e virgola". Questo fa sì che nient'altro che già appare sullo schermo venga disturbato dalla stampa dei numeri.

Il display in momenti diversi della cornice e dei numeri (reso possibile dal posizionamento del cursore), porta ad un programma semplice e chiaro. L'alternativa sarebbe di combinare le funzioni di cui alle Figure 4.14 e 4.15 in una sola routine che costruisca le caselle con già i numeri dentro di esse, poi le stampa una linea per volta. Questo vorrebbe poi dire che la costruzione della stringa VL$ (fatta in anticipo durante la routine di inizializzazione) dovrebbe essere integrata con questa funzione. Cioè, tre compiti semplici che sono presentati in maniera facile e separata nella versione attuale, dovrebbero essere soppiantati da un programma che si presenterebbe molto più difficile da capire. Questo esempio aiuta a capire l'utilità del posizionamento del cursore nella progettazione di programma di display.

```
#Display della cornice
frame       PRINT HD$: PRINT TL$                    #nomi dei giorni e linea
                                                    superiore
            FOR LX = 1 TO NL
              FOR VV = 1 TO VS
                PRINT VL$                           #lati della cornice
                NEXT VV
              PRINT BL$                             #linea inferiore
              NEXT LX
            CC = TC: LL = LL(NL) + BH               #posizionare il cursore
            GOSUB cursor
            PRINT MO$(MO); YR                       #stampare il mese e
                                                    l'anno sotto la cornice
            RETURN

130  PRINT HD$:PRINT TL$:FOR LX=1 TO NL:FOR VV=1 TO VS:PRINT
     VL$:NEXT VV:PRINT BL$:NEXT LX
135* CC=TC:LL=LL(NL)+BH:GOSUB800:PRINT MO$(MO);YR;:RETURN
```

Questa routine mostra la cornice vuota del Calendario.

* La versione per l'Apple di questa linea differisce soltanto nell'apparire della costante di stringa "" tra le variabili MO$(MO) e YR.

Figura 4.14 — La cornice per il Calendario

La Figura 4.16 mostra la routine usata per il posizionamento del cursore alla linea LL, colonna CC. Su un Pet questo viene eseguito ripetendo la stampa dei caratteri "cursor right" e "cursor down", dopo una stampa iniziale del carattere "home". Questa funzione viene implementata sull'Apple usando dei comandi HTAB e VTAB. La versione del TRS-80 è simile a quella del Pet, ma con caratteri che muovono il cursore in differenti e diverse dimensioni dello schermo.

La Figura 4.17 illustra l'inizializzazione speciale usata per il Calendario. Questa routine viene chiamata in aggiunta a quella d'inizializzazione che è stata presa dal Compleanno.

```
#Mostrare i giorni da 1 a ND iniziando con la casella SZma

daydisp     DY = 1
            FOR LX = 1 TO NL
              LL = LL(LX)                          #linea per la LXma riga
              IF LX = 1 THEN
                ZC = SZ + 1                        #ZC = prima posizione
              else
                ZC = 1
              FOR CX = ZC TO 7
                CC = CC(CX)                        #colonna per questo numero

                GOSUB cursor
                PRINT RIGHT$("" + STR$(DY),2);     #2 caratteri al posto giusto

                DY = DY + 1                        #contare i giorni stampati

                IF DY > ND THEN
                  RETURN
                NEXT CX
              NEXT LX
            RETURN

400  DY = 1:FOR LX = 1 TO NL:LL = LL(LX):ZC = 1:IF LX = 1 THEN ZC =
     SZ + 1
405  FOR CX = ZC TO 7:CC = CC(CX) : GOSUB800 : PRINT RIGHT$("" +
     STR$(DY), 2);
410  DY=DY+1:IF DY >ND THEN RETURN
415  NEXT CX:NEXT LX:RETURN

  Questa è la routine che riempie le caselle del calendario con i numeri dei giorni.
```

Figura 4.15 — Display dei numeri dei giorni

SOMMARIO

Abbiamo dunque progettato due giochi (Compleanno e Calendario, per illustrare le tecniche importanti usate per l'input e l'output di date.

```
#Posizionamento del cursore — muoverlo alla linea LL, colonna CC
#Versione per il Pet

cursor      PRINT "home";                    #spostarsi alla linea 0,
                                             colonna 0
            CC = CC mod columns              #far si che le discussio-
                                             ni siano comprese
            LL = LL mod lines                #entro la gamma della
                                             grandezza dello scher-
                                             mo
            IF CC > 0 THEN                   #muoversi verso destra
                                             con CC
              FOR ZZ = 1 TO CC
                PRINT "right";
                NEXT ZZ
            IF LL > 0 THEN                   #muoversi verso il bas-
                                             so con LL
              FOR ZZ = 1 to LL
                PRINT "down";
                NEXT ZZ
            RETURN

800* PRINT CHR$(19); : CC = CC — 40 * INT(CC/40) : LL = LL — 24 *
     INT(LL/24)
805* IF CC>0 THEN FOR ZZ = 1 TO CC:PRINT CHR$(29);:NEXT ZZ
810* IF LL>0 THEN FOR ZZ = 1 TO LL:PRINT CHR$(17);:NEXT ZZ
815  RETURN
```

Questa è la routine di posizionamento del cursore per il Calendario. La versione per l'Apple è più semplice, perchè i comandi per il posizionamento del cursore sono già in esistenza in quel sistema.

---

* Per la versione dell'Apple, le linee 805 e 810 sono:

```
805  HTAB CC+1
810  VTAB LL+1
```

Per la versione per il TRS-80, i valori delle colonne e delle linee sono 64 e 16 e non 40 e 24. I caratteri "home", "right" e "down" (alla posizione di partenza, a destra e in basso) sono rappresentati da CHR$(28), CHR$(25) e CHR$(26).

Figura 4.16 — Posizionamento del cursore per il Calendario

```
#Inizializzazione speciale per il Calendario
calinit     READ BW,BH: DATA boxwidth,boxheight      #larghezza ed altezza
                                                     della cornice
            READ LZ,CZ:DATA boxline,boxcol           #angolo in alto a sinistra
                                                     della cornice
            FOR XX = 1 TO 6
              LL(XX) = LZ + (XX - 1) * BH            #fissare le linee per i
                                                     numeri
              NEXT XX
            FOR XX = 1 TO 7
              CC(XX) = CZ + (XX - 1) * BW            #fissare le colonne per i
                                                     numeri
              NEXT XX
            TL$ = "": BL$ = "": VL$ = ""             #linee in alto, in basso e
                                                     in mezzo
            FOR XX = 1 TO 7
              TL$ = TL$ + "four top line characters"
              BL$ = BL$ + "one lowerleft corner and three bottom line characters"
              VL$ = VL$ + "one vertical line character and three blanks"
              NEXT XX
            VL$ = VL$ + "one vertical line character"
            BL$ = BL$ + "one vertical linecharacter"
            HD$ = "SUN MON TUE WED THU FRI SAT"
            VS = BH - 1                              #numero di linee di VL$
                                                     per riga
            TC = CZ + 5                              #colonne per la parte
                                                     scritta in lettere
            RETURN
900*  READ BW,BH:DATA4,3
905   READ LZ,CZ:DATA3,1
910   FOR XX=1 TO 6:LL(XX)=LZ+(XX-1)*BH:NEXT XX
915   FOR XX=1 TO 7:CC(XX)=CZ+(XX-1)*BW:NEXT XX
920   TL$="":BL$="":VL$=""
925** FOR XX = 1 TO 7:TL$ = TL$ + "        ":VL$ = VL$ + " ":BL$ = BL$
      + "        ": NEXT XX
930** VL$=VL$+"[ ":BL$=BL$+"[ "
935   HD$=" SUN MON TUE WED THU FRI SAT"
940   VS=BH - 1:TC=CZ+7:RETURN
```

Questa è l'inizializzazione speciale per il Calendario. Questa routine viene chiamata in aggiunta alla routine d'inizializzazione per il Compleanno.

* Nella versione per il TRS-80 i due valori nella linea 900 sono 4 e 2.

** Nelle versioni per l'Apple e il TRS-80 la differenza in queste linee consiste nell'uso di trattini e di punti esclamativi per le linee verticali e orizzontali delle caselle del calendario.

Figura 4.17 — Inizializzazione per il Calendario

Il Compleanno mostra l'algoritmo usato per determinare il giorno della settimana in cui cade una certa data. Inoltre dà un esempio di routine di input di data con una tale flessibilità che sia possibile usarla per tutti.

Il Calendario usa la routine che compone il programma del Compleanno in modo da mostrare un calendario per un certo mese e anno. Il programma mostra il tipo di contorsione che si rende talvolta necessaria per adattare una routine già esistente ad una situazione nuova. Inoltre mostra anche le semplificazioni strutturali che possono risultare dall'uso dei comandi di posizionamento del cursore per ottenere delle funzioni di display logicamente indipendenti separatamente.

CAPITOLO 5

L'ESATTORE DELLE TASSE

Lasciatemi dire come sarà,
Uno per Lei, diciannove per me...
E se il cinque per cento Le pare poco,
Ringrazi che non me lo prendo tutto,
Perchè io sono l'Esattore delle Tasse

Canzone di George Harrison

Il gioco che andiamo a presentare in questo capitolo è vecchio come il cucco, ma la versione qui descritta probabilmente sfrutta meglio le capacità del vostro computer. L'Esattore delle tasse è un gioco di tipo diverso da quelli presentati prima in questo libro. Uno degli obiettivi è quello di indovinarne le regole.

ISTRUZIONI PER IL GIOCO

Si comincia con una domanda da parte del programma, egli vuole sapere la grandezza di un "tortino". Questa domanda allude al diagramma che forse avrete visto, in cui un dollaro d'argento veniva diviso come un tortino – una fetta per l'affitto, una per il mangiare, una per le tasse, ecc. ecc. Voi rispondete inserendo un numero intero.

Il tortino rappresentato sullo schermo non sarà rotondo. Sarà un array rettangolare di numeri. Infatti, voi, inserendo il numero intero N, avete risposto alla richiesta del computer di sapere la grandezza del tortino, esso consisterà dei numeri interi 1, 2, ..., N. Il valore minimo di N permesso è 4. Quello maggiore dipenderà dalla grandezza del vostro schermo, e viene dedotto da valori contenuti nelle dichiarazioni DATA nella routine di inizializzazione (Vedere Figura 5.12).

Nella Figura 5.1 vi è un esempio di gioco. Voi avete scelto la grandezza del tortino 15; i numeri 1, 2, 3...., 15 appaiono sullo schermo. Tutti i numeri a parte il numero 1 sono "al contrario". Non vi viene detto perchè, ma la maggior parte dei giocatori ne scopre subito la ragione.

Sotto al display di cui alla Figura 5.1 appaiono i totali attuali:

YOU: 0% TAXMAN:0% LEFT: 100%

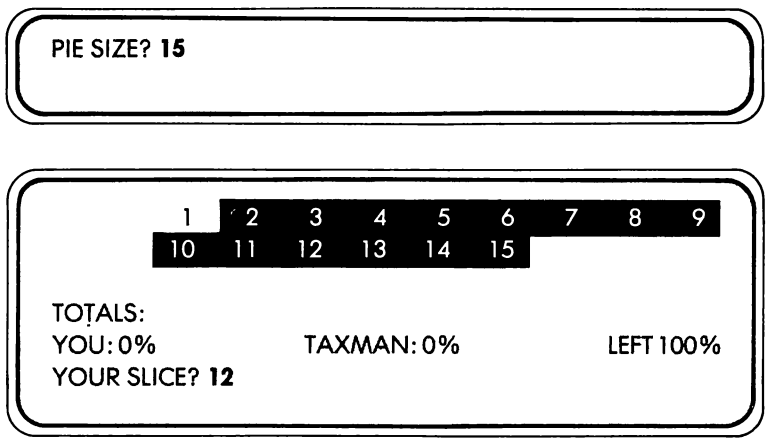
E più sotto appare la domanda:

YOUR SLICE?

Questa domanda vi suggerisce di inserire un numero tra 1 e 15. (Anche questo non ve lo dice nessuno, ma la maggior parte dei giocatori impara in fretta a non inserire numeri che non appaiano sullo schermo).

Nella Figura 5.1 avete inserito il numero 12. Lo schermo si pulisce, e nuove informazioni appaiono. Diversi numeri che facevano parte del "tortino" non appaiono più ed altri non appaiono più "al contrario". Ora sotto il "tortino" vi sono dei nuovi totali:

YOU: 10% TAXMAN: 13.3% LEFT: 76.7%



Prima di tutto il giocatore sceglie un tortino di 15. Poi il tortino viene mostrato intero, e il giocatore sceglie la fetta 12. Nella terza parte si vede che la scelta del giocatore ha avuto come risultato che il 10% è andato al giocatore e il 13.3% all'Esattore delle Tasse. Il restante 76.7% viene rappresentato da un tortino diverso: mancano alcuni numeri, ed altri non sono più in "reverse video". La scelta effettuata dal giocatore, 15 per la fetta seguente ci porta ad esaminare il quarto display, ma poi il giocatore commette lo sbaglio di scegliere 10, e l'Esattore delle Tasse si prende tutto il tortino rimasto.

Figura 5.1a — Esempi di display per l'Esattore delle Tasse

Il vostro compito è quello di capire come questi numeri si riferiscano alla fetta di tortino che avete scelto, ai numeri mancanti, e ai numeri che non sono più "al contrario". (Suggerimento: La somma dei numeri 1, 2, ... 15 è 120).

Per la vostra prossima fetta, avete scelto 15, e lo schermo vi mostra il nuovo "tortino" ed i nuovi totali. La vostra prossima scelta di 10 risulta nella confisca da parte dell'Esattore delle Tasse del resto del "tortino". I totali finali vengono scritti sulla penultima linea del display finale:

YOU KEPT: 22.5% TAXMAN GOT: 77.5%

Più sotto compare un riassunto del punteggio migliore delle mani precedenti:

THE RECORD IS 54% OF 30.

Questo significa che nella mano più fortunata, avete indicato un tortino di 30 e ne avete tenuto il 54%.

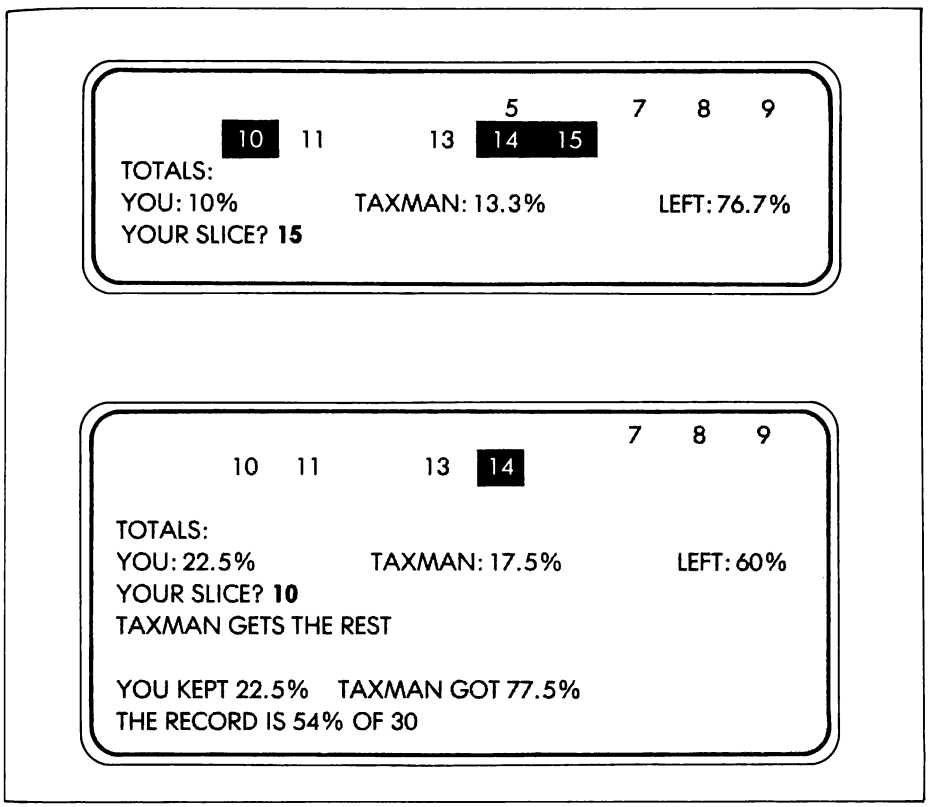


Figura 5.1b — Esempi di display per l'Esattore delle Tasse

Una volta che i totali finali ed il risultato migliore siano apparsi, il programma aspetta che il giocatore schiacci un qualsiasi tasto per pulire lo schermo ed essere pronto per ricominciare.

```
#L'Esattore delle Tasse
            GOSUB init                          #fissare la grandezza
                                                 dell'array e delle costan-
                                                 ti
            repeat{
              GOSUB turninit                    #fissare la grandezza
                                                 del tortino, inizializzare
                                                 HT
              IF N = 0 THEN
                break                           #N = 0 è il segnale per
                                                 smettere la partita
              repeat{
                GOSUB showpie                   #mostrare quello che è
                                                 rimasto del tortino
                IF CH > 0 THEN
                  INPUT "YOUR SLICE"; S         #lasciare che il giocato-
                                                 re ne scelga un pezzo
                GOSUB taxgrab                   #è il turno dell'Esattore
                                                 delle Tasse
                } until (PL = 0)                #è finito quando non c'è
                                                 più tortino
              GOSUB stats                       #mostrare i risultati fi-
                                                 nali
              }
            END
```

```
100  GOSUB530
110  GOSUB420:IF N=0 THEN END
120* GOSUB150:IF CH>0 THEN INPUT "YOUR SLICE"; S
130  GOSUB260:IF PL>0 THEN120
140  GOSUB360:GOTO110
```

Questa è la routine principale del programma dell'Esattore delle Tasse. Per cominciare ciascun turno di gioco, il giocatore sceglie una grandezza di tortino. Poi il giocatore e l'Esattore delle Tasse prendono un pezzo di tortino a turno finchè non ce ne rimane più. L'Esattore segue determinate regole che non vengono dette al giocatore. Un giocatore intelligente è in grado di indovinare le regole, ma bisogna tener conto che il conoscere le regole non è motivo di successo quando si gioca contro l'Esattore delle Tasse.

* La versione per l'Apple differisce soltanto nella spaziatura della costante di stringa.

Figura 5.2 — L'esattore delle Tasse

```
#Mostrare il tortino avanzato
showple     GOSUB clearscreen                    #iniziare con lo schermo vuoto
            CH = 0                               #contare la scelta del giocatore
            FOR YY = 1 TO N STEP LS              #display di una linea soltanto
              TP = min(N, YY + LS - 1)           #l'ultima linea può essere vuota
              FOR XX = YY TO TP                  #numeri individuali
                IF HT(XX) < 0 THEN
                  NM$ = L2$ + SZ$                #vuoto se non è possibile
                else {
                  NM$ = STR$(XX)                 #altrimenti il numero
                  IF XX < 10 THEN                #aggiunge degli spazi vuoti
                    NM$ = L2$ + NM$              #per aggiustare
                  else IF XX < 100 THEN          #in campo da tre cifre
                  } NM$ = L1$ + NM$
                IF HT(XX) > 0 THEN               #se l'inserimento non è zero
                  {RV = 1: CH = CH + 1}          #usare "reverse video", contare la scelta
                else
                  RV = 0                         #non usare "reverse video"
                GOSUB displaynum                 #mostrare il campo
                NEXT XX
              PRINT                              #a capo
              NEXT YY
            GOSUB totals                         #mostrare i totali sotto il tortino
            RETURN

150 GOSUB520:CH=0:FOR YY=1 TO N STEP LS:TP=YY + LS - 1:IF
    TP >N THEN TP = N
160 FOR XX=YY TO TP:IF HT (XX) < 0 THEN NM$=L2$ + SZ$:GOTO190
170 NM$=STR$(XX):IF XX< 10 THEN NM$=L2$ + NM$:GOTO190
180 IF XX < 100 THEN NM$=L1$ + NM$
190 RV=0:IF HT(XX) > 0 THEN RV = 1:CH = CH +1
200 GOSUB480:NEXT XX:PRINT:NEXT YY:GOSUB210:RETURN
```

Questa routine mostra il "tortino" che è un gruppo di numeri da 1 a N (che rappresentano la grandezza del tortino). I numeri già scelti dal giocatore o dall'Esattore non compaiono. Certi numeri compaiono in reverse video per ragioni che il giocatore deve indovinare.

Figura 5.3 — Mostrare la parte di tortino avanzata

IL PROGRAMMA DELL'ESATTORE DELLE TASSE

Il programma dell'Esattore delle Tasse è illustrato nelle Figure da 5.2 a 5.12. La sua struttura è simile a quella di altri programmi illustrati in questo libro.

La parte "segreta" del gioco è contenuta nelle subroutines illustrate nella Figura 5.6 e 5.8. Queste routines usano l'array HT per codificare informazioni riguardanti i numeri che appaiono sullo schermo, quali appaiono "al contrario" e quali il giocatore può scegliere. La posizione originale di HT, i suoi cambiamenti dopo ogni scelta da parte del giocatore e il metodo di calcolo del punteggio sono tutti basati sull'algoritmo aritmetico che appare in queste due routines.

La subroutine più complicata di tutto il programma è illustrata nella Figura 5.3. Questa è la routine che mostra il nuovo "tortino". Uno degli obiettivi del programmatore era quello di rendere la routine indipendente dall'attuale sistema BASIC che usa. E le tecniche usate per raggiungere questo scopo sono:

— La variabile LS (inserita nella routine di inizializzazione di cui alla Figura 5.12) è

```
#Mostrare i totali a questo punto della partita

totals      PK = FNP(KP/PZ): PK$ = LB$ + STR$(PK)   #percentuale da tenere
            PT = FNP(PY/PZ): PT$ = LB$ + STR$ (PT)  #percentuale da pagare
            PP = FNP(PL/PZ): PP$ = LB$ + STR$(PP)   #percentuale "rimasta
                                                     nel tortino"
            PRINT: PRINT "TOTALS:"
            PRINT "YOU:"; PK$ "% ";
              "TAXMAN:"; PT$; "%";
              "LEFT:"; PP$; "%"
            RETURN

210  PK=FNP(KP/PZ):PK$=LB$+STR$(PK)
220  PT=FNP(PY/PZ):PT$=LB$+STR$(PT)
230  PP=FNP(PL/PZ):PP$=LB$+STR$(PP)
240* PRINT:PRINT "TOTALS:":PRINT "YOU:";PK$;"% TAXMAN:";PT$;"%
     LEFT:";PP$;"%"
250  RETURN
```

Questa routine stampa le percentuali che appartengono al giocatore, all'Esattore e che sono rimaste nel tortino a questo punto della partita. La variabile di stringa LB$ viene usata per essere certi che il numero sia preceduto da uno spazio, senza tener conto di quale sistema BASIC il programma usi. LB$ viene determinata dalla routine d'inizializzazione nella Figura 5.12.

* La versione per l'Apple di questa linea differisce soltanto nella spaziatura nelle costanti.

Figura 5.4 — Mostrare i totali

stata inserita. Il valore di LS è il numero di cifre che appariranno in ciascuna linea del display del tortino. Nell'esempio illustrato nella Figura 5.1, LS ha valore 9.

– Le variabili L1$ e L2$, che sono entrambe state definite in termini della variabile LB$, vengono usate per fornire giustifiche esatte. LB$ viene definita nella routine d'inizializzazione (linee 550 e 560 nella Figura 5.12) dai comandi:

```
L = LEN(STR$(1))
LB$ = " ": IF L>1 THEN LB$ = " "
```

Questo fa si che LB$ sia la stringa senza valore alcuno in quei sistemi che forniscono un bianco portante per i numeri positivi (per esempio Pet e TRS-80), mentre in quei sistemi che non forniscono un bianco, LB$ consiste di un bianco singolo.

– Una chiamata per una routine separata (GOSUB 480 sulla linea 200 nella Figura 5.3) viene usata per produrre l'immagine "a rovescio". La routine a 480 (nella Figura 5.9) è dipendente dal sistema.

```
#Calcolo delle Tasse

taxgrab     IF CH > 0 THEN {                    #se c'è la possibilità di
                                                 scelta
              S = INT(S)
              IF 2 < = S < = N THEN              #e se la scelta del gio-
                                                 catore
                IF HT(S) > = 1 THEN {            #è valida
                  GOSUB playfair                 #allora seguire le regole
                  RETURN
                  }
              }
            PRINT "TAXMAN GETS THE REST."        #altrimenti...
            PY = PY + PL
            PL = 0
            RETURN

260   IF CH=0 THEN300
270   S=INT(S):IF S<2 OR S>N THEN300
280   IF HT(S)<1 THEN300
290   GOSUB310:RETURN
300   PRINT "TAXMAN GETS THE REST.":PY=PY+PL:PL=0:RETURN
```

Questa routine decide se dare o no la possibilità al giocatore di avere parte del tortino. Se l'Esattore non lo prende tutto, allora viene chiamata la subroutine illustrata nella Figura 5.6 per determinare la parte spettante all'Esattore.

Figura 5.5 — Calcolo delle Tasse

Dovreste essere in grado di capire il resto del programma dell'Esattore semplicemente leggendolo.

```
#Calcolare la parte che spetta all'Esattore

playfair    TX = 0                              #inizializzare la parte
                                                che spetta all'Esattore
            KP = KP + S: PL = PL - S            #la parte che spetta al
                                                giocatore è S
            HT(S) = - 1                         #levare la S
            IF INT(N/S) >= 2 THEN {             #i suoi multipli ora
              FOR YY = 2 TO INT(N/S)            #hanno tutti
                HT(YY*S) = HT(YY*S) - 1         #meno divisori
                NEXT YY
            FOR XX = INT(S/2) TO 1 STEP - 1     #trovare la parte che
                                                spetta all'Esattore
              IF XX is a divisor of S
              AND HT(XX) >= 0 THEN
                HT(XX) = - 1                    #un'altra per l'Esattore
                TX = TX + XX
                FOR YY = 2 TO INT(N/XX)         #aggiustare i suoi multi-
                                                pli
                  HT(YY*XX) = HT(YY*XX) - 1     #i divisori
                  NEXT YY
                }
              NEXT XX
            PY = PY + TX: PL = PL - TX          #aggiornare i totali da
                                                "pagare"
            RETURN

310  TX=0:KP=KP+S:PL=PL-S:HT(S)=-1:IF INT(N/S)<2 THEN330
320  FOR YY=2 TO INT(N/S):HT(YY*S)=HT(YY*S)-1:NEXT YY
330  FOR XX=INT(S/2) TO 1 STEP - 1:IF HT(XX) < 0 OR S <>
     XX*INT(S/XX) THEN350
340  HT(XX) = - 1:TX = TX + XX:FOR YY = 2 TO INT(N/XX):HT(YY * XX)
     = HT(YY * XX) - 1: NEXT YY
350  NEXT XX:PY=PY+TX:PL=PL - TX:RETURN
```

Questa routine calcola la parte che spetta all'Esattore. L'algoritmo usato in questa routine è segreto e il giocatore deve indovinarlo per poter giocare.

Figura 5.6 — La parte che spetta all'Esattore

```
#Mostrare le statistiche finali

stats .     PK = FNP(KP/PZ)                          #percentuale da "tene-
                                                     re"
            PT = 100 - PK                            #percentuale per le
                                                     "tasse"
            IF PK > BK THEN {                        #è un nuovo record?
              BK = PK: BZ = N
              NR = 1                                 #si
              }
            else
              NR = 0                                 #no
            PK$ = LB$ + STR$(PK)                     #trasformare le percen-
                                                     tuali in stringhe
            PT$ = LB$ + STR$(PT)                     #con spazi bianchi
            PRINT: PRINT "YOU KEPT";PK$;
            "% TAXMAN GOT";PT$;"%"
            IF NR = 1 THEN                           #fare il paragone con i
                                                     record precedenti
              PRINT "THAT'S A NEW RECORD!"
            else {
              BK$ = LB$ + STR$(BK)
              BZ$ = LB$ + STR$(BZ)
              PRINT "THE RECORD IS"; BK$; "% OF";BZ$
              }
            GOSUB onech                              #lasciare sullo schermo
                                                     finchè non viene schiac-
                                                     ciato un tasto
            RETURN


360   PK = FNP(KP/PZ):PT = 100 - PK:NR = 0:IF PK > BK THEN BK =
      PK:BZ = N:NR = 1
370   PK$ = LB$ + STR$(PK):PT$ = LB$ + STR$(PT)
380   PRINT:PRINT "YOU KEPT";PK$;"% TAXMAN GOT";PT$;"%"
390   IF NR = 1 THEN PRINT "THAT'S A NEW RECORD!":GOTO410
400   BK$ = LB$ + STR$(BK):BZ = LB$ + STR$(BZ):PRINT "THE RECORD
      IS";BK$"% OF";NZ$
410   GOSUB500:RETURN


  Questa routine mostra il risultato finale e lo paragona a quelli precedenti.
```

Figura 5.7 — Mostrare le statistiche finali

```
=Preparare per questa partita

turninit    GOSUB clearscreen                    #iniziare con lo schermo vuoto
            repeat{
              INPUT "PIE SIZE"; N                #lasciare che il giocatore inserisca la grandezza del tortino
              IF N = 0 THEN                      #N = 0 è il segnale per smettere la partita
                RETURN
              else
                N = INT(N)                       #è necessario avere numeri interi
            }until(4 <= N <= MX)
            PL = N*(N+1)/2: PZ = PL              #la somma di 1, 2, ... N
            FOR XX = 1 TO N                      #azzerare l'array "hit"
              HT(XX) = 0
              NEXT XX
            FOR XX = 1 TO INT(N/2)               #contare i divisori
              FOR YY = 2 TO INT(N/XX)
                HT(YY*XX) = HT(YY*XX) + 1
                NEXT YY
            KP = 0: PY = 0                       #inizializzare gli importi da tenere e da pagare

            RETURN
```

```
420  GOSUB520
430* INPUT "PIE SIZE";N:IF N = 0 THEN RETURN
440  N = INT(N):IF N < 4 OR N > MX THEN430
450  PL = N * (N + 1)/2:PZ = PL:FOR XX = 1 TO N:HT(XX) = 0:NEXT XX
460  FOR XX = 1 TO INT(N/2) : FOR YY = 2 TO INT(N/XX) : HT(YY * ) =
     HT(YY * XX) + 1 : NEXT YY : NEXT XX
470  KP = 0:PY = 0:RETURN
```

Questa routine permette al giocatore di inserire la grandezza del tortino da dividere con l'Esattore delle Tasse. La routine poi inizializza l'array HT, che viene usato per sapere quali numeri si possono usare e quali sono già stati usati.

---

* La versione per l'Apple di questa linea differisce soltanto nei valori nelle costanti.

Figura 5.8 — Preparare un nuovo tortino

```
#Mostrare NM$ — usando il reverse video se RV = 1
#Versione per il Pet

displaynum   IF RV = 1 THEN
               PRINT "rvs"; NM$; "off";
             else
               PRINT NM$;
             RETURN

480*  IF RV=1 THEN PRINT CHR$(18);NM$;CHR$(146);:RETURN
490   PRINT NM$;:RETURN
```

Questa è la prima di diverse routines che hanno versioni diverse per ogni sistema in BASIC. Quella per il Pet usa i codici di carattere "rvs" e "off" per mettere la stringa che deve essere mostrata in reverse video tra parentesi. La versione per l'Apple è molto simile, perchè mette tra parentesi un comando PRINT NM$; tra i comandi INVERSE e NORMAL. Per il TRS-80 è necessario usare un approccio diverso, perchè non ci sono comandi per il reverse video già installati nel sistema.

NOTA DEL TRADUTTORE: Per reverse video si intende una frase o dei numeri scritti in bianco su nero invece che nero su bianco che appaiono sullo schermo.

* La versione per l'Apple di questa linea è:

```
480          IF RV=1 THEN INVERSE: PRINT NM$;: NORMAL: RETURN
```

Figura 5.9 — Reverse Video

```
#Input ad un solo carattere
#Versione per il Pet

onech        repeat
               GET X$
               until (X$ < > "")
             RETURN

500   GETX$:IF X$="" THEN500
510   RETURN
```

Questa routine da un input ad un solo carattere per il Pet. La stessa routine funziona anche per l'Apple, ma non è necessario avere il loop, invece bisognerà inserire un codice per riconoscere il controllo C, perchè il comando GET dell'Apple non lo fa. Nel TRS-80 bisognerà usare un loop simile a questo, ma con la funzione INKEY$ invece di GET, perchè GET significa una cosa completamente diversa nel TRS-80.

Vedere la Figura 2.11 per le versioni per l'Apple e per il TRS-80.

Figura 5.10 — Input ad un solo carattere

SUGGERIMENTI PER MIGLIORARE IL PROGRAMMA E RENDERLO PIÙ COMPLETO

Ecco diversi metodi per migliorare questo gioco:

- Usare il controllo del cursore per alterare l'immagine del tortino invece di pulire lo schermo ed ottenere una nuova immagine.
- Registrare le percentuali maggiori e minori per ciascun "tortino".
- Registrare i risultati di ciascun giocatore e determinare così se e quanto il giocatore migliora. Fare dei commenti incoraggianti o di congratulazioni dopo ciascuna mano (o dopo diverse mani).

SOMMARIO

L'Esattore è un gioco che sfida il giocatore a scoprire le regole. Il programma mostra un modo di non dover essere legati alle piccole differenze che esistono tra i diversi sistemi BASIC. È certamente possibile portare delle migliorie al programma, specialmente per quanto riguarda il controllo del cursore e la registrazione dei risultati.

```
#Pulire lo schermo
#Versione per il Pet

clearscreen  PRINT "clr";
             RETURN

520    PRINT CHR$(147);:RETURN
```

Questa routine pulisce lo schermo. La versione per l'Apple usa il comando HOME. Quella per il TRS-80 usa il comando CLS.

Vedere la Figura 2.12 per le versioni per l'Apple e per il TRS-80.

Figura 5.11 — Pulire lo schermo

```
#Inizializzazione

init        READ LS: DATA linesize          #numeri/linea in display
            READ DL: DATA screenlines       #linee/quadro
            MX = LS * (DL – textunder       #sottrarre le linee per il
                                             testo in basso
            DIM HT(MX)                      #dimensionare l'HT in
                                             rapporto
            L = LEN(STR$(1))                #vedere se il BASIC in-
                                             serisce lo spazio vuoto
                                             portante
            SZ$ = ""                        #SZ$ = un numero vuo-
                                             to ad una cifra
            FOR LL = 1 TO L
              SZ$ = SZ$ + ""
              NEXT LL
            IF L > 1 THEN                   #LB$ assicura uno spa-
                                             zio vuoto portante
              LB$ = ""                      #LB$ è nullo se il BASIC
                                             fornisce uno spazio vuo-
                                             to
            else
              LB$ = ""
            L1$ = LB$ + ""                  #extra spazi vuoti por-
                                             tanti – a valore 2 cifre
            L2$ = L1$ + ""                  #valore 1 cifra
            DEF FNP(X) = INT(1000*X + .5)/10  #funzione per XX.X%
            RETURN

530*  READ LS:DATA 9:READ DL:DATA 24
540   MX=LS*(DL – 5):DIM HT(MX)
550   L=LEN(STR$(1)):SZ$="":FOR LL=1 TO L:SZ$=SZ$+"":NEXT LL
560   LB$="":IF L > 1 THEN LB$=""
570   L1$=LB$+"":L2$=L1$+""
580   DEF FNP(X)=INT(1000*X + .5)/10
590   RETURN
```

Questa è la routine d'inizializzazione per il programma dell'Esattore delle Tasse. Le variabili SZ$, LB$, L1$, L2$ sono determinate usando lo stesso codice per l'Apple, il Pet e Il TRS-80, ma non hanno gli stessi valori. Questo è un esempio di programmazione indipendente dal sistema.

---

* Nella versione per il TRS-80, i valori di LS e DL sono 15 e 16.

Figura 5.12 — Inizializzazione

CAPITOLO 6

PROGRAMMARE CON IL FREE BASIC

Nei capitoli precedenti abbiamo descritto diversi programmi in Free BASIC. Ogni programma appariva in due forme: Free BASIC e BASIC. Le nostre discussioni fino ad adesso si sono sempre riferite alla forma BASIC. Ora spiegheremo come usare il Free BASIC. Vi sarà necessario saperlo usare se volete continuare a leggere e capire questo libro, perchè nelle nostre discussioni a proposito dei programmi non ci riferiremo più al BASIC. D'ora innanzi si parlerà soltanto di Free BASIC. Inizieremo con l'esaminare il progetto di un programma.

Programmare in BASIC è semplice – così semplice che inganna. È come imparare a mettere dei chiodi ed a segare del legno; vi sarà facile imparare a farlo in fretta, ma questo non vi farà diventare un falegname. Un programmatore esperto, come un esperto falegname, deve saper fare ben di più che usare degli strumenti. L'artigiano abile, sia esso falegname o programmatore, deve saper usare l'intelligenza, e deve saper prevedere in anticipo. Sedersi davanti ad una tastiera ed iniziare a scrivere un programma è come costruire una casa senza il progetto.

TECNICHE DI PROGETTAZIONE DI UN PROGRAMMA

Le tre tecniche principali che i programmatori usano per progettare e per descrivere i propri programmi sono:

- Schematizzazione delle operazioni (Flowcharts)
- Descrizioni verbali degli algoritmi
- Pseudocodici

Useremo il Programma dell'Addizione nel Capitolo 1 per illustrare queste tecniche. Poichè probabilmente già conoscete la schematizzazione delle operazioni, non di-

scuteremo questa tecnica in dettaglio. Nella Figura 1.1 vediamo uno schema delle operazioni dell'Addizione presentato nella Figura 1.2.

Una descrizione verbale degli algoritmi è uno schema che non usa figure. Le frasi incasellate dello schema diventano "passi" numerati e le frecce dello schema diventano passi numerati del tipo:

"Go back to step 39"

Nella Figura 6.1 vediamo una descrizione verbale di un algoritmo che corrisponde allo schema del programma dell'Addizione.

Nella Figura 6.2 vediamo lo pseudocodice che un programmatore potrebbe aver scritto prima di scrivere il programma dell'Addizione. Mentre il contenuto dello pseudocodice è lo stesso dello schema e della descrizione verbale, la sua forma è leggermente diversa. Prima di tutto la struttura del loop del programma viene indicata con la parola "repeat" seguita dal gruppo di azioni che vanno ripetute. Queste azioni sono messe tra parentesi. Notate che la versione pseudocodice non contiene nè frecce nè numeri di passi. Nella rappresentazione che usa lo pseudocodice non vi sono analogici del comando BASIC GOTO. Dei "super comandi" del tipo "repeat" forniscono delle strutture che si uniscono e che si diramano ma che non possono essere implementate come loops FOR...NEXT. Dunque GOTO non è necessario nella rappresentazione che usa lo pseudocodice.

L'esperienza acquisita con linguaggi "strutturati" come il Pascal e il C mostra che questo tipo di progettazione di programmi, conosciuta come "programmazione senza GOTO", porta ad ottenere programmi che sono più semplici da scrivere, da far partire e da capire rispetto a quelli progettati con comandi GOTO.

1. Scegliere due numeri ad una cifra, N1 e N2.
2. Chiedere "Cosa fa N1 + N2?"
3. Se il giocatore risponde correttamente, passare al punto 6.
4. Dire "Sbagliato, riprova".
5. Ritornare al punto 2.
6. Dire "Giusto, prova ad indovinarne un altro".
7. Ritorna al punto 1.

Questa è una descrizione verbale dell'algoritmo per il programma dell'Addizione che abbiamo presentato nel Capitolo 1. La sequenza delle operazioni inizia al punto 1 e continua con i diversi punti in ordine numerico, a parte quando si trova davanti a punti come il 3, il 5 o il 7, che alterano esplicitamente questa sequenza.

Figura 6.1 — Descrizione dell'Addizione usando un algoritmo verbale

FREE BASIC

Il Free BASIC è una forma di pseudocodice. È molto più preciso di quello mostrato nella Figura 6.2, poichè il Free BASIC usa dei comandi BASIC invece delle descrizioni verbali del passo.

Il Free BASIC è stato sviluppato dall'autore per ottenere due scopi principali:

- Per liberare il programmatore di BASIC da numeri di linee BASIC
- Per far sì che le tecniche di progettazione di un programma che non usino GOTO si possano applicare al BASIC.

Nella Figura 6.3 vediamo il modo in cui il Free BASIC è stato usato in questo libro. Una descrizione del Free BASIC appare per prima in ciascuna figura. Nella 6.3 vediamo la descrizione di un programma semplice nella forma di un loop infinito. Il programma prende i valori di diametro da una lista di dati DATA e stampa poi i valori di circonferenza corrispondenti.

Sotto alla descrizione in Free BASIC vi sono i comandi in BASIC che derivano dal Free BASIC. La traduzione da Free BASIC in BASIC è stata fatta a mano, usando delle semplici regole meccaniche. Nella maggior parte dei casi, due programmatori diversi, tradurrebbero una certa descrizione in Free BASIC in identici programmi BASIC.

Poichè il Free BASIC è stato progettato per essere tradotto a mano in BASIC, non è stato necessario includere i meccanismi formali che sono necessari per far sì che un programma di computer sia in grado di eseguire la traduzione. Mentre esaminiamo il programma illustrato nella Figura 6.3 in dettaglio, vedremo anche degli esempi

```
repeat {
   pick two single-digit numbers N1 and N2
   repeat {
      ask "What is N1 + N2?"
      if answer is incorrect
         say "Wrong, try again"
      else
         say "Right, try another"
      } until (answer is correct)
}
```

Questa è una versione in pseudocodice del programma dell'Addizione. I "punti" sono uguali a quelli dello schema delle operazioni (Figura 1.1) e della descrizione usando un algoritmo verbale (Figura 6.1), ma la sequenza delle operazioni viene mostrata in maniera diversa.

Figura 6.2 — Pseudocodice per l'Addizione

della "informalità" del Free BASIC che rende la traduzione eseguita con una macchina impossibile.

Ora discuteremo le convenzioni del Free BASIC.

Uso delle lettere minuscole

La prima cosa da notare a proposito del Free BASIC è l'uso di lettere minuscole per rappresentare gli elementi non-BASIC del Free BASIC, e l'uso delle lettere maiuscole per rappresentare del materiale che sarà trasportato, senza cambiarlo, nel programma BASIC. Per esempio, nella Figura 6.3, vediamo che le lettere minuscole sono state usate per le parole "repeat" e "pi". La parola "repeat" segna una delle strutture di controllo del Free BASIC. (Discuteremo tutte le strutture di controllo del Free BASIC in seguito). La parola "pi" è un esempio dell'informalità del Free BASIC: è un

```
repeat {
  READ D
  C = pi * D
  PRINT "DIAMETER:"; D; " CIRCUMFERENCE: "; C
  }
DATA 4,6,10
```

```
10  READ D
20  C=3. 14159*D
30  PRINT "DIAMETER: ";D; " CIRCUMFERENCE: ";C
40  GOTO 10
50  DATA 4,6,10
```

Questa Figura mostra una descrizione in Free BASIC per un programma molto semplice. Sotto alla descrizione in Free BASIC c'è un gruppo di comandi in BASIC derivato dal Free BASIC. Se si fa un paragone tra le due versioni, vediamo che:

- Il Free BASIC è "GOTO-less" (senza GOTO) e non usa numeri di linea.
- Nel Free BASIC viene usato un sistema di "dentellatura" per poter fornire le informazioni necessarie alla comprensione della struttura del programma.
- Vengono usate lettere minuscole per comporre le descrizioni in Free BASIC che non hanno parti in BASIC.
- Vengono usati dei parametri, cioè dei nomi simboli per le costanti, nel Free BASIC, ma non viene data nessuna indicazione dei valori di ciascuno di questi parametri.

Figura 6.3 — Un programma semplice in Free BASIC

nome simbolico per la costante che apparirà nel programma BASIC (3.14159 nella Figura 6.3).

Tuttavia, da nessuna parte nel programma Free BASIC abbiamo definito "pi : = 3.14159" che sarebe necessaria per far sì che un programma di computer possa derivare il programma BASIC dalla descrizione in Free BASIC come per la Figura 6.3. Non abbiamo fornito nessun meccanismo di questo tipo, poichè tale formalità non sarebbe di utilità maggiore per il Free BASIC usato in questo libro.

Una costante alla quale è stato dato un nome simbolico è chiamata un "parametro"del programma. Questo nome deriva dalla terminologia delle statistiche matematiche. Un parametro di una distribuzione statistica è una costante che si trova nella formula che definisce la distribuzione. Valori diversi della costante danno luogo a distribuzioni diverse ma correlate. Allo stesso modo, nel programma descritto nella Figura 6.3, valori diversi del parametro "pi" danno luogo a programmi diversi ma correlati. Un valore di 3.14 dà luogo a un gruppo diverso di valori di circonferenza rispetto a quelli generati da un valore di 3.14159.

Capoversi

Notate che la descrizione del programma in Free BAJSIC che appare nella Figura 6.3 usa i capoversi per riflettere la struttura del programma. Esso consiste di due comandi: il "repeat" e il "DATA". Dunque le parole "repeat" e "DATA" sono le uniche parole che appaiono al margine sinistro della pagina. I tre comandi che passano fra il comando "repeat" sono spostati da questa posizione di capoverso; se uno qualsiasi di questi comandi contenesse a sua volta degli altri comandi, allora anche questi sarebbero spostati. Notate inoltre che la parentesi di chiusura compare da sola su una linea, spostata di tanto quanto lo sono i comandi che essa delimita.

Queste convenzioni per i capoversi sono studiate per facilitare la presentazione visiva del materiale. Infatti, il significato della descrizione in Free BASIC sarebbe uguale anche se non ci fossero i capoversi, o se le parentesi fossero piazzate sulla stessa linea dei comandi che racchiudono.

Commenti

La maggior parte dei linguaggi per programma fanno sì che il programmatore possa includere dei commenti nel codice che funge da fonte al programma. Un commento è un aiuto esterno in più per il lettore che serve a chiarificare lo scopo o il significato di un comando. Poichè il Free BASIC è la forma del programma BASIC che è studiata per essere letta e capita, è necessario far sì che sia possibile aggiungerci dei commenti. Nel Free BASIC, qualsiasi testo compreso tra # e la fine di una linea è un commento. Esempi di commenti compaiono in tutti i programmi dei giochi descritti in questo libro.

TRADURRE DAL FREE BASIC IN BASIC

Non esiste un programma che traduca la descrizione del programma scritta in Free BASIC in comandi in BASIC. Questo vuol dire che dovete tradurvela da soli. Diamo un'occhiata alle regole del Free BASIC e discutiamo le tecniche di traduzione che potreste usare.

Il comando "Repeat"

La forma generale del comando "repeat" è:

```
repeat instruction until (condition)
```

Per esempio,

```
repeat X = X + 1 until (X = 9)
```

sarebbe tradotto nel codice BASIC seguente:

```
10 X = X + 1
IF X <> 9 THEN 10
```

In genere, il significato di

```
repeat instruction until (condition)
```

è: eseguire il comando ripetutamente, seguendo ciascuna esecuzione con una prova della condizione. Se questa si dimostra vera, allora bisogna smettere di eseguire il comando. Per qualsiasi comando e qualsiasi condizione, la forma generale del comando da ripetere può essere tradotto nel seguente codice BASIC:

```
N instruction
  IF NOT condition THEN N
```

dove N è il prossimo numero di linea disponibile. Se il vostro computer non riconosce l'operatore logico NOT (tutti i computer più usati lo fanno), o se preferite modificare il programma BASIC, potete trasformare "NOT condition" in una condizione equivalente. Nell'esempio di cui sopra, abbiamo usato "X <> 9" invece di "NOT X = 9".

La definizione generale che abbiamo dato qui fa sì che sia possibile ripetere soltanto un solo comando. Le parentesi giocano lo stesso ruolo nelle descrizioni in Free

BASIC e nelle espressioni aritmetiche. Per esempio, in una espressione aritmetica, se desideriamo moltiplicare un termine per 5 possiamo scrivere

```
5 * term
```

Se il termine che vogliamo moltiplicare per 5 è la variabile A, allora possiamo scrivere

```
5 * A
```

ma se il termine è A + B + C, allora dobbiamo scrivere

```
5 * (A + B + C)
```

Le parentesi fanno sì che tutto quello che è compreso dentro di esse deve essere trattato come un unico termine nell'espressione. Allo stesso modo quando scriviamo delle istruzioni tra parentesi, come nella Figura 6.3, tutto quello che è compreso entro di esse deve essere trattato come un solo comando nella descrizione in Free BASIC.

Il comando "repeat" nella Figura 6.3 non comprende la parte "until (condition)". Nella Figura 1.3 vediamo una descrizione in Free BASIC del programma dell'Addizione.

Il comando:

```
150 IF A <> N1 + N2 THEN ... GOTO 120
```

nella Figura 1.2 corrisponde alla parte

```
until (A = N1 + N2)
```

del comando "repeat" nella Figura 1.3.

Notate che la condizione che segue la parola "until" è sempre entro parentesi. Questo perchè così ci sia possibile sapere sempre esattamente quali sono i termini che compongono la condizione. Infatti in molti casi si presenterebbero delle ambiguità se non si usassero le parentesi, soprattutto quando la condizione è composta da diversi termini. Le condizioni che seguono un "until" nel comando "repeat" sono uguali a quelle usate in una dichiarazione IF, ma le parentesi non sono necessarie in questo ultimo caso, perchè BASIC sa sempre che la condizione è sempre ciò che è compreso tra l'IF e il THEN.

Un altro modo di terminare un comando "repeat" è con l'uso di un comando "break". Quest'ultimo fermerà la ripetizione del comando, essendone lui stesso un sottocomando. Un esempio dell'uso di un "break" è spiegato nella routine principale del programma di Accoppiamento. (Vedi Figura 7.16 nel Capitolo 7). In molti casi l'u-

so del comando "break" può essere evitato riorganizzando la struttura del programma. La riorganizzazione spesso rappresenterà un miglioramento, poichè la necessità di usare un comando "break" può essere segno di cattiva organizzazione del programma.

If... then... else

Più di 2.000 anni fa Aristotele determinò le regole della logica che sono poi diventate lo strumento di ragionamento di tutto il mondo Occidentale. Una di queste regole è quella del "centro escluso", che dice: "qualsiasi asserzione può essere sia vera che falsa — deve essere una dei due, non può essere tutt'e due". Questa regola fa da modello alla costruzione IF...THEN...else del Free BASIC. La forma generale di questa costruzione è

IF condition THEN instruction 1 else instruction 2

La parte "else instruction 2" non è però obbligatoria. L'effetto di questo comando è che se la "condizione" è vera, allora il comando "1" viene eseguito; se la "condizione" è falsa, allora il comando "2" viene eseguito. Il risultato è dunque che uno ed uno soltanto dei due comandi viene eseguito. (Se la "condizione" è falsa, e non esiste il comando 2, allora non si esegue nessun comando). Come il comando "repeat" anche questi comandi possono essere uno solo o una serie di comandi contenuti in parentesi.

Nella Figura 1.3, in cui vediamo una descrizione in Free BASIC del programma dell'Addizione, vi è un esempio di questo tipo di costruzione e degli usi convenzionali dei capoversi e del raggruppamento. In questa figura, la "condizione" è "A = N1 + N2", il "comando 1" è "PRINT "THAT'S RIGHT etc." e il "comando 2" è "PRINT "THAT'S WRONG etc.".

Notate l'uso delle lettere maiuscole e minuscole con la costruzione IF...THEN...else. Avremmo potuto scrivere if...then...else (tutto in minuscolo) per distinguere la costruzione Free BASIC da quella BASIC IF...THEN. Tuttavia, l'IF...THEN del Free BASIC IF...THEN...else viene sempre tradotto direttamente nel BASIC IF...THEN. L'uso del Free BASIC che noi facciamo sottolinea questo concetto scrivendo IF...THEN in maiuscolo. Poichè la maggior parte dei piccoli computer con sistemi BASIC non permette l'uso di ELSE con IF...THEN (il TRS-80 ha ELSE, il Pet e l'Apple no), abbiamo usato le lettere minuscole per scrivere l'else.

I casi

La dichiarazione del caso, una forma utilissima della dichiarazione IF, si basa su

un'altra tecnica preferita da Aristotele, l'enumerazione delle possibilità. La forma generale di questa dichiarazione del caso è

```
IF case
        condition 1 THEN instruction 1
        condition 2 THEN instruction 2


                .
        condition n THEN instruction n
else
        instruction n + 1
```

La parte "else instruction n + 1" di questa dichiarazione non è obbligatoria.

La dichiarazione del caso è di grande aiuto per la chiarificazione del programma.

Quando vedete la dichiarazione di cui sopra, sapete che solo uno, ed uno soltanto, dei comandi che vi compaiono, sarà eseguito. Il comando eseguito sarà quello corrispondente alla prima condizione della lista che si riveli vera. Se tutte le condizioni solo false, allora il comando da eseguire sarà n+1.

Nella Figura 6.4 vediamo un programma Free BASIC che usa una dichiarazione del caso. In questa dichiarazione ci sono tre condizioni ed una parte "else". Quello che il resto di questo programma di gioco fa resta a voi scoprirlo. Infatti, l'obiettivo del gioco (chiamato "Provatelo se ci riuscite") è quello di scoprire che cosa fa il programma. La Figura 6.5 mostra un programma BASIC derivato dalla Figura 6.4. Se avete già esperienza come programmatore, o se avete già letto delle pubblicazioni al riguardo, vi sarà già capitato di dover decifrare dei programmi simili a questo del "Provatelo se ci riuscite".

Mentre

L'ultima struttura di controllo che introdurremo è usata per implementare i loops che iniziano con una prova. La forma generale di una struttura "while" è:

```
while (condition) instruction
```

Il comando in una struttura "while" non sarà eseguita se la condizione non è vera. (Nella Figura 4.6 nel Cap. 4 vediamo un programma che usa una costruzione di questo tipo).

Subroutines

L'obiettivo principale del Free BASIC è quello di liberare il programmatore di BASIC dall'aver il problema di pensare continuamente ai numeri delle linee. Le strutture

di controllo che abbiamo introdotto eliminano la necessità di usare i numeri delle linee nelle dichiarazioni IF e GOTO. Rimane soltanto un problema: come eliminare i numeri delle linee dai comandi GOSUB. La soluzione che abbiamo scelto è quella di usare dei numeri di linee simbolici per le linee iniziali delle subroutines. (Se vi sembra necessario usare un GOTO nel vostro programma, potete piazzare un numero di linea simbolico sulla linea alla quale volete riferirvi). Questi numeri di linee simbolici vengono chiamati con un nome (in lettere minuscole naturalmente) alla sinistra del comando, esattamente allo stesso posto dove comparirebbe un numero di linea.

```
INPUT A$, B$
IF case
  A$ = "A" THEN
    IF B$ = "1" THEN
      PRINT "A1"
    else
      PRINT "AX"
  A$ = "B" THEN
    repeat
      INPUT C$
      until (B$ = C$)
  A$ = "C" THEN
    IF VAL(B$) < =0 THEN
      PRINT "???"
    else {
      FOR J = 1 TO VAL(B$)
        PRINT "*";
        NEXT J
      repeat {
        INPUT C$
        IF C$ = "Q" THEN
          break
        else
          PRINT "XYZ"
      } }
  else
    PRINT "HA HA"
END
```

Questo programma appartiene ad un gioco chiamato "Provalo se ci riesci". Lo scopo del gioco è di trovare cosa fa il programma. Vi è possibile operare il programma e leggere la lista (In un'altra versione meno semplice il giocatore può operare il programma ma senza leggere la lista in Free BASIC, solo quella in BASIC). L'abilità necessaria a giocare questo gioco vi servirà in situazioni di vita reale.

Figura 6.4 — Un programma d'esempio

Traduzione meccanica

In tutto questo libro vi sono diversi programmi per i quali abbiamo fornito sia una descrizione in Free BASIC che dei comandi in BASIC. Questi programmi forniscono degli esempi pratici della traduzione del Free BASIC in BASIC. In molti casi la traduzione è stata eseguita meglio a mano che usando un programma computerizzato. Per ciascuna delle costruzioni usate in questo capitolo, c'è una traduzione meccanica "ovvia" in BASIC. Nella Figura 6.6 vediamo l'esempio.

FREE BASIC, PROGRAMMAZIONE STRUTTURIZZATA E PASCAL

Il Free BASIC non è altro che un BASIC "strutturizzato". Avrete certamente sentito parlare di "programmazione strutturizzata". Questo termine deriva dalle "Annotazioni sulla Programmazione Strutturizzata" di Edsger W. Dijkstra (apparse in "Programmazione strutturizzata" di Dahl e altri; Academic Press, 1972). In questa monografia Dijkstra ha sottolineato due punti:

- I programmi senza GOTO sono più semplici da capire, e più spesso corretti rispetto a quelli con comandi GOTO.
- Inoltre si possono ottenere diversi benefici dall'arrangiamento dei programmi in "strati", ciascuno rappresentante un livello diverso di astrazione e ognuno usa un "sistema ideale" per ciascun livello di astrazione diverso.

```
 10   INPUT A$,B$
 20   IF A$ <> "A" THEN 40
 30   IF B$ = "1" THEN PRINT "A1":GOTO 110 ELSE PRINT "AX":GOTO
      110
 40   IF A$ <> "B" THEN 60
 50   INPUT C$:IF B$ <> C$ THEN 50 ELSE 110
 60   IF A$ <> "C" THEN 100
 70   IF VAL(B$) <=0 THEN PRINT "???":GOTO 110
 80   FOR J = 1 TO VAL(B$):PRINT "*";:NEXT J
 90   INPUT C$:IF C$ = "Q" THEN 110 ELSE PRINT "XYZ":GOTO 90
100   PRINT "HA HA"
110   END
```

Questo è il gioco "Provalo se ci riesci" che è illustrato nella Figura 6.4. È scritto in BASIC per il TRS-80 che permette di avere delle clausole ELSE in dichiarazioni IF. Il programma sarebbe molto più complicato su di un Apple o su di un Pet, poichè questi sistemi non permettono clausole ELSE.

Figura 6.5 — BASIC per il gioco di "Provalo se ci riesci"

Il primo di questi punti è abbastanza semplice da capire. Quale risultato, diversi libri, articoli e seminari hanno cercato di insegnare la "programmazione strutturizzata", ma in effetti si sono concentrati sulle tecniche di programmazione senza GOTO.

Uno degli obiettivi principali del Free BASIC è quello di portare i benefici ottenuti con la programmazione senza GOTO a tutti i programmatori BASIC, in questo senso

Free BASIC		**BASIC**
repeat	nnn	instruction
instruction		IF NOT condition THEN nnn
until (condition)		
while (condition)	nnn	IF NOT condition THEN mmm
instruction		instruction
		GOTO nnn
	mmm	...
IF condition THEN		IF NOT condition THEN nnn
instruction 1		instruction 1
else		GOTO mmm
instruction 2	nnn	instruction 2
	mmm	...
IF case		
condition 1 THEN instruction 1		IF NOT condition 1 THEN nn2
		instruction 1
.		GOTO mmm
condition 2 THEN instruction 2	nn2	IF NOT condition 2 THEN nn3
		instruction 2
		GOTO mmm
.		.
condition n THEN instruction n	nnn	IF NOT condition n THEN eee
		instruction n
		GOTO mmm
else	eee	instruction n + 1
instruction n + 1	mmm	...

Una semplice rispondenza tra le strutture di controllo del Free BASIC e i comandi del BASIC viene qui illustrata. Un computer con un programma semplice potrebbe tradurre un sistema nell'altro.

Figura 6.6 — Rispondenza tra il Free BASIC e il BASIC

il Free BASIC sostiene ed aiuta le tecniche di programmazione strutturizzata.

Il secondo punto è più difficile da comprendere, e la tecnica qui suggerita è più complicata. I programmi illustrati in questo libro non sono stati progettati con l'idea di "stratificare" o usando un "sistema ideale" d'astrazione; dunque, in questo senso, i programmi illustrati in questo libro non parlano delle tecniche della programmazione strutturizzata. In effetti, invece, abbiamo usato la tecnica della progettazione "sotto-sopra". Le tecniche di strutturizzazione sottosopra vengono ampiamente discusse in questo libro in diversi capitoli. (Per esempio, vedi Cap. 7, il Gioco dell'Accoppiamento).

Il Pascal è un linguaggio di programmazione progettato per facilitare la programmazione senza GOTO. Le strutture di controllo del Pascal sono praticamente identiche a quelle usate nel Free BASIC, cosicchè una descrizione in Free BASIC di un programma sembra un incrocio "ibrido" tra il BASIC e il Pascal. Dunque, molte delle descrizioni in Free BASIC riportate in questo libro, potrebbero formare la base per dei programmi in Pascal. Vi sono, però, due ostacoli a questa traduzione:

- Non tutte le caratteristiche del BASIC sono ottenibili in Pascal "standard".
- I programmi in Pascal necessitano di una "dichiarazione" di tutti i dati e di tutte le subroutines. In Pascal, inoltre, dovete spiegare prima di usare un determinato oggetto, che cosa esso sia, e in seguito, il Pascal controllerà continuamente che questo oggetto non venga usato nella maniera non pre-determinata.

SOMMARIO

Vi sono tre tecniche principali che sono usate dai programmatori per progettare e descrivere i loro programmi: la schematizzazione delle operazioni, le descrizioni verbali degli algoritmi e gli pseudocodici. Il Free BASIC è una versione di pseudocodice che è così accurata che è paragonabile ad un linguaggio da computer. Il Free BASIC usa una convenzione che assegna ruoli diversi alle lettere minuscole e maiuscole. Le lettere maiuscole vengono usate per parti della descrizione in Free BASIC che apparirà parola per parola nel programma BASIC. Le lettere minuscole vengono usate per tutte le altre parti della descrizione in Free BASIC. L'uso principale delle lettere minuscole si vede nei parametri del programma, e nelle parole chiave usate in specificazioni di strutture di controllo. I capoversi vengono usati nelle descrizioni in Free BASIC per riflettere la struttura del programma. Il loro uso non è obbligatorio e ciascun programmatore è libero di adottare capoversi diversi da quelli usati dall'autore.

Le strutture di controllo usate in Free BASIC sono

- repeat...until
- IF...THEN...else
- IF case...else
- while

Il comando "break" fa sì che sia possibile terminare improvvisamente quei loops che sono definiti da costruzioni "repeat" e "while".

Queste caratteristiche eliminano dunque il bisogno di comandi GOTO nelle descrizioni in Free BASIC. Inoltre, numeri di linee simbolici sono usati in comandi GOSUB per chiamate di subroutines. Questo è il passo finale nella liberazione dei programmatori BASIC dall'uso di numeri di linee — questo era l'obiettivo principale del Free BASIC.

Il Free BASIC inoltre sostiene l'aspetto del non uso del GOTO nella programmazione strutturizzata. Le sue strutture di controllo sono le stesse di quelle usate nel Pascal, cosicchè una descrizione in Free BASIC può essere tradotta facilmente in Pascal. Gli ostacoli principali di queste traduzioni sono la mancanza di caratteristiche specifiche del Pascal, e la sua necessità di "dichiarazioni" delle subroutines e dei dati.

CAPITOLO 7

IL GIOCO DEGLI ACCOPPIAMENTI

In questo capitolo presentiamo il Gioco degli Accoppiamenti, un gioco vero e proprio, che spiega diversi punti importanti riguardanti la programmazione di sistemi di computer interattivi. Una descrizione di come si gioca viene presentata per prima, e poi presenteremo i programmi che vengono usati per il gioco.

L'obbiettivo del gioco è quello di "accoppiare" membri di due gruppi diversi che si dimostrino compatibili. I gruppi possono essere composti di uomini e donne, venditori e compratori, animali e padroni, lavoratori e disoccupati, e così via. La compatibilità viene determinata da una serie di domande, e poi paragonando le risposte di ciascun giocatore con le preferenze dichiarate da ciascun membro dell'altro gruppo. Per esempio, se i due gruppi usati sono uomini e donne ad una festa da ballo, allora lo scopo del gioco è quello di trovare coppie compatibili. Questo si fa prima di tutto dando un nome ai due gruppi, UOMINI e DONNE. Poi si prepara un gruppo di domande con scelte multiple per ciascun gruppo, e si dà un peso numerico a ciascuna domanda. Questo peso numerico viene usato per determinare il grado al quale le risposte di un giocatore si "accoppiano' con le preferenze dichiarate dai membri dell'altro gruppo.

Proviamo a fare una partita d'esempio. Si comincia con l'apparire sullo schermo di un "::" nell'angolo a sinistra in alto. Nella Firura 7.1 vediamo i comandi che possono essere inseriti in risposta all'apparire del "::". Ciascuno consiste una sola lettera o cifra.

LA FASE DI COSTRUZIONE DEL GIOCO

La prima fase del gioco è quella di costruzione dello stesso, in cui si inseriscono i nomi dei gruppi e le domande a risposta multipla. Nella Figura 7.2 vediamo un dialogo d'esempio che potrebbe seguire l'inserimento di un comando "I". Tutte le risposte del giocatore sono seguite da RETURN (o ENTER). Le risposte appaiono in neretto, così da distinguerle dalle domande e dagli output del computer.

Nella Figura 7.2 il giocatore ha inserito "I", e il programma ha risposto con "GROUP 1" per indicare che ora il giocatore deve inserire il nome del primo gruppo. Il giocatore inserisce "MEN" e il programma risponde con "Q1 FOR MEN" per far si che il giocatore inserisca la prima domanda per il gruppo 1. Se il giocatore avesse inserito "DOGPATCH" come nome per il gruppo 1, il programma avrebbe chiesto "Q1 FOR DOGPATCH". Per la prima domanda per il gruppo MEN, il giocatore ha inserito "WHOM DO YOU MOST RESPECT?". Notate che il giocatore è stato in grado di scrivere questa stringa senza metterla tra virgolette, nonostante contenga degli spazi bianchi. Questo suggerisce che è stato usato il comando LINE INPUT. Più tardi, quando parleremo del programma, vedremo che LINE INPUT viene simulato da una subroutine.

Ora il programma inizia a suggerire dei numeri, e il giocatore risponde specificando le scelte possibili per la prima domanda. Nel nostro esempio, dopo che cinque

COMANDO	AZIONE
I	*Inizializzare:* Il gioco ricomincia da capo. Il programma chiede i nomi dei due gruppi, poi accetta un lotto diverso di domande per ciascun gruppo.
L	*Load:* Questo comando non viene usato nella versione del programma illustrata in questo libro. Un nome del gruppo e un lotto di domande per quel gruppo vengono inseriti da cassette o dischi.
N	*Redazione dei nomi:* I nomi dei gruppi vengono mostrati, e vengono accettati dei nomi sostitutivi.
Q	*Smettere:* Il suggerimento "[ ]" viene mostrato e si inizia la fase di gioco.
S	*Registrare:* Questo comando non viene usato nella versione del programma illustrata in questo libro. Un nome del gruppo e un lotto di domande per quel gruppo vengono registrati su cassette o dischi.
1	*Redazione del Gruppo 1:* Il suggerimento "*" viene mostrato, e i comandi per la redazione delle domande del Gruppo 1 vengono accettati.
2	*Redazione del Gruppo 2: Il suggerimento* "*" viene mostrato, e i comandi per la redazione delle domande del Gruppo 2 vengono accettati.

Questi sono i comandi che possono essere inseriti in risposta al suggerimento "::". I comandi I, L e S portano a dei compiti precisi. I comandi Q, 1, e 2 portano a routines di selezione di ulteriori comandi.

Figura 7.1 — Risposte al suggerimento "::"

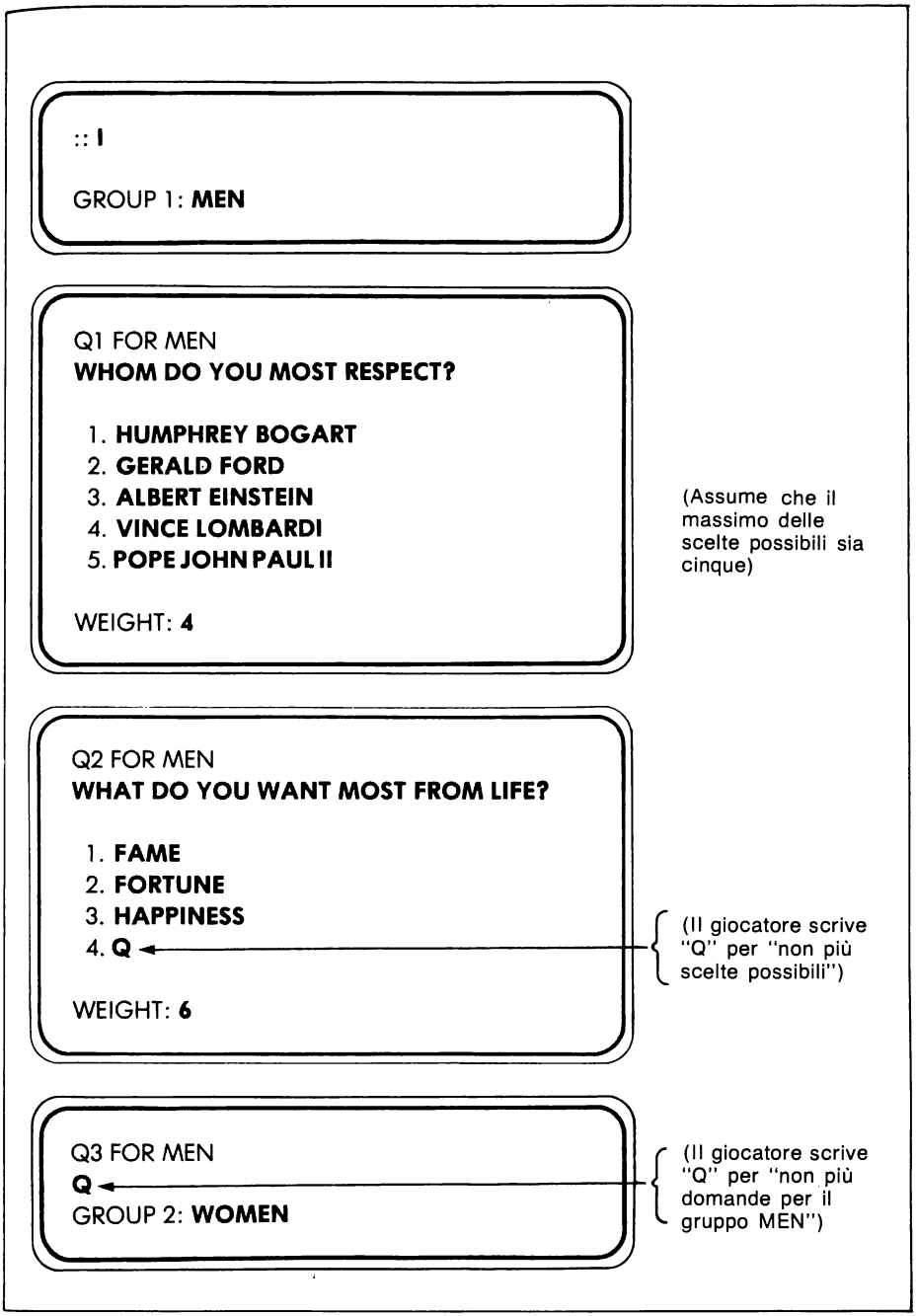


Figura 7.2 — Dialogo di inizializzazione

scelte sono state inserite, il programma chiede "WEIGHT:". Questo segnala al giocatore che è il momento di inserire il peso numerico assegnato a questa domanda. Nella Figura 7.2 il programma si ferma dopo cinque scelte, perchè il numero massimo è stato fissato a cinque da una dichiarazione DATA nella subroutine di inizializzazione.

Discuteremo come fissare questo ed altri parametri in seguito, ma notate che se avete un piccolo computer (per esempio con solo 8K di RAM) dovrete assegnare valori molto bassi al numero massimo di scelte per ciascuna domanda, al numero delle domande e al numero dei giocatori, perchè questi parametri rientrano nelle dichiarazioni di dimensione per gli array multi-dimensionali.

Dopo che il giocatore ha inserito un peso numerico di 4 per la prima domanda, il programma suggerisce "Q2 FOR MEN," e il giocatore risponde con la domanda seguente: "WHAT DO YOU WANT MOST FROM LIFE?". Ancora una volta il program-

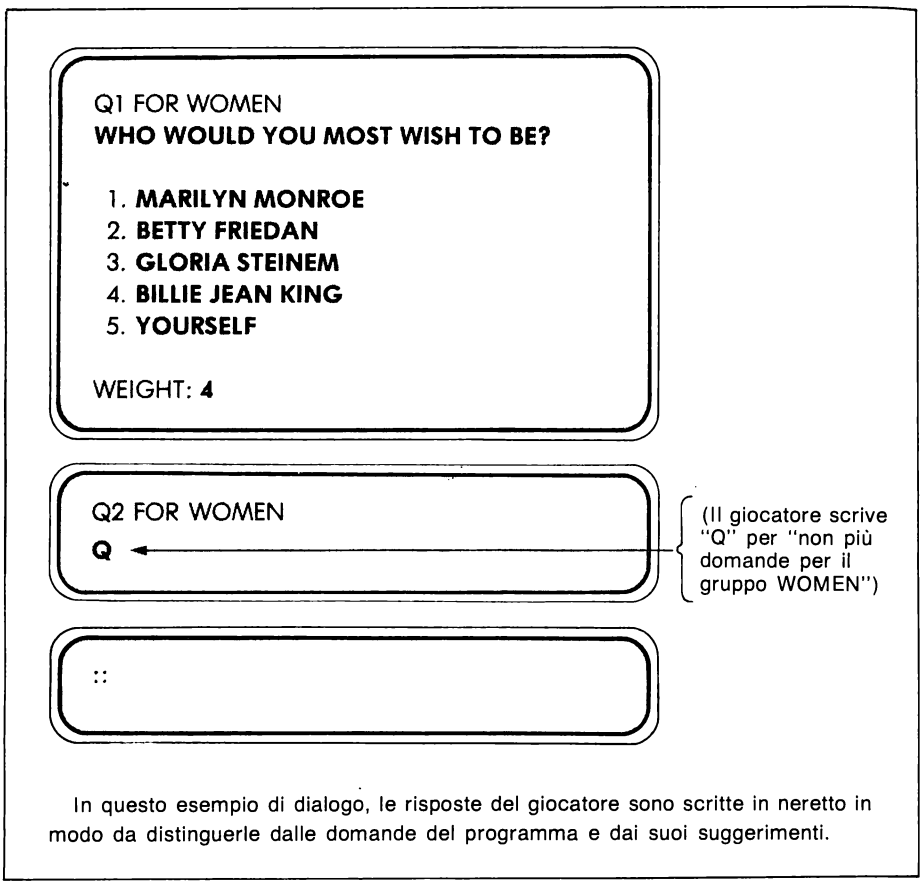
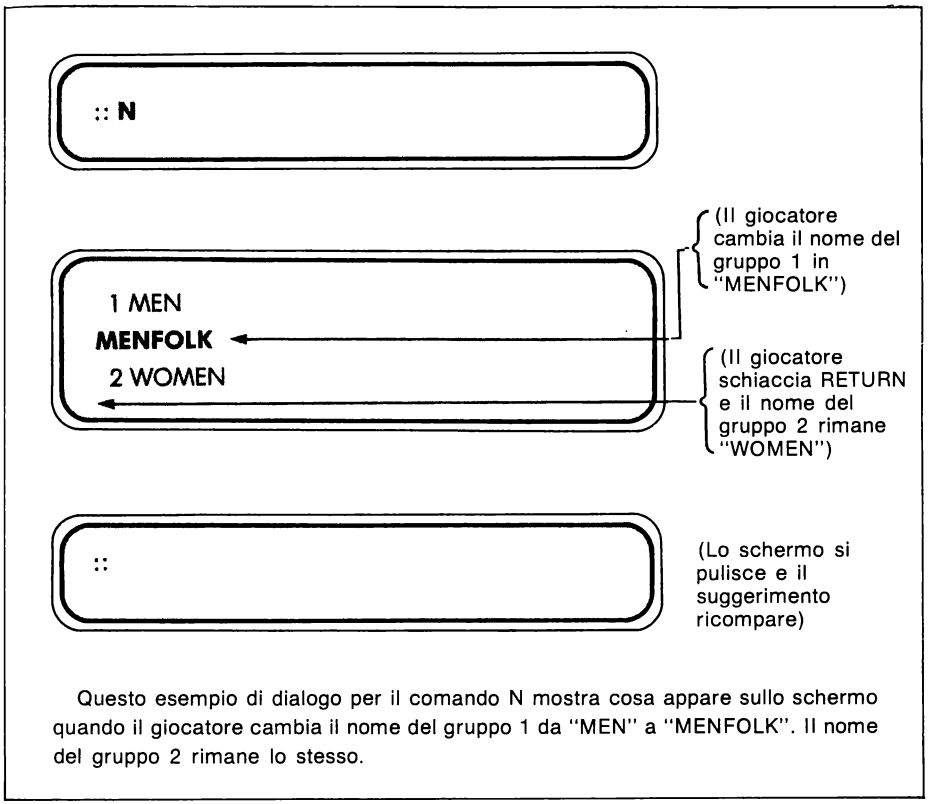


Figura 7.2a — Dialogo di inizializzazione

ma richiede una specifica delle risposte possibili mostrando i numeri delle scelte, ma questa volta, dopo aver inserito tre scelte, il giocatore risponde con un "Q".

In tutto questo Gioco degli Accoppiamenti, Q viene usato spesso come comando di "fermata". In questo caso specifico significa "Non ho intenzione di inserire nessuna altra scelta per questa domanda. Andiamo avanti" Il programma risponde con "WEIGHT:" e il giocatore inserisce 6. Poi il programma risponde con "Q3 FOR MEN" Il giocatore inserisce "Q" un'altra volta; questa volta "Q" però significa "Non voglio più inserire domande per il gruppo 1". Il programma dunque procede con il gruppo 2. Il dialogo per il gruppo 2 é simile a quello per il gruppo 1.

Nella Figura 7.2 non si vede però un aspetto del display che è difficile da rappresentare nello scritto. Tre volte durante il dialogo si è usato "Q". In modo da ridurre le possibilità di confusione, il programma rimuove il carattere Q dallo schermo se l'input del giocatore consiste di un carattere singolo "Q" seguito da RETURN. Dunque il display attuale sarà qualcosa come quello illustrato nella Figura 7.2 ma la Q finale sarà rimpiazzata da spazi bianchi.



Questo esempio di dialogo per il comando N mostra cosa appare sullo schermo quando il giocatore cambia il nome del gruppo 1 da "MEN" a "MENFOLK". Il nome del gruppo 2 rimane lo stesso.

Figura 7.3 — Dialogo di redazione del nome del gruppo

Vediamo come i comandi N, 1 e 2 possono essere usati per modificare il gruppo di domande inserite nella Figura 7.2 (Vedi Figura 7.1) poichè i comandi S e L non sono implementati nella versione degli Accoppiamenti illustrata in questo libro, non li includeremo nella nostra discussione.

Il comando N

Nella Figura 7.3 vediamo un dialogo d'esempio per il comando N. Nell'esempio, due nomi di gruppi vengono mostrati a turno. Ogni volta che un nome viene mostrato, il giocatore può inserire o un RETURN, che lascerà il nome del gruppo senza cambiarlo, o può inserire un nuovo nome. In questo esempio, il nome del gruppo 1 è stato cambiato da "MEN" a "MENFOLK', mentre il nome del gruppo 2 rimane inalterato.

COMANDO	AZIONE
A	*Aggiungere:* Al giocatore è permesso di aggiungere una domanda alla fine del lotto di domande per il gruppo. Il programma suggerisce dando il numero della domanda e della scelta, poi ne richiede il peso numerico.
B	*Prima:* Il programma chiede al giocatore il numero della domanda che seguirà quella nuova da inserire. Poi richiederà le informazioni relative alla domanda, solo per la funzione aggiungere.
D	*Cancellare:* Il programma chiede il numero di una domanda e la cancella, muovendo le domande seguenti di un posto, così da non lasciare spazi vuoti in ogni gruppo di domande.
E	*Redigere:* Il programma domanda il numero di una domanda e mostra la dichiarazione che le compete, e lascia che il giocatore decida di cambiarla o lasciarla uguale. Poi il programma passa fra tutte le risposte e quindi il giocatore decide di cambiarle o di cancellarle o di lasciarle uguali. Inoltre il programma accetta delle altre risposte alle domande (fino ad un massimo prestabilito). Ed infine il peso numerico assegnato a ciascuna domanda viene mostrato dal programma, e il giocatore può lasciarlo uguale o cambiarlo.
Q	*Smettere:* Una volta che la redazione del gruppo sia completa, lo schermo viene pulito e appare il "::".

Questi sono i comandi che si possono inserire in risposta al "::" che appare quando sia stato dato il comando "1" o quello "2".

Figura 7.4 — Risposte al suggerimento "*"

```
:: 2
```

```
*E
QUESTION NUMBER: 1
WHO WOULD YOU MOST WISH TO BE?
 1. MARILYN MONROE
BO DEREK                                   (Cambia la 1ª risposta)
 2. BETTY FRIEDAN                          (Schiaccia RETURN)
 3. GLORIA STEINEM
D                                          (Cancella la 3ª risposta)
 3. BILLIE JEAN KING
ELIZABETH HOLTZMAN                         (Cambia la 3ª risposta nuova)
 4. YOURSELF                               (Schiaccia RETURN)
 5.                                        (Schiaccia RETURN)
WEIGHT:  4  :  7                           (Cambia il peso numerico)
```

```
*E
QUESTION NUMBER: 1
WHO WOULD YOU MOST WISH TO BE?

 1. BO DEREK
 2. BETTY FRIEDAN
 3. ELIZABETH HOLTZMAN
 4. YOURSELF
 5.

WEIGHT:  7:
```

(Il giocatore controlla la domanda cambiata richiedendo la funzione di redazione e schiacciando RETURN ogni volta)

```
*
```

Questo esempio di dialogo mostra come si possa redigere la prima domanda inserita per il gruppo 2 nella Figura 7.2. Le risposte del giocatore vengono mostrate in neretto. (A parte quelle in risposta a "::" e "*", tutte le risposte del giocatore terminano con il tasto RETURN).

Figura 7.5 — Redazione delle Domande

Il comando 1 e quello 2

I comandi 1 e 2 sono simili tra di loro. In entrambi i casi, lo schermo si pulisce, il nuovo suggerimento "*" appare, e il programma attende che il giocatore inserisca un altro comando con un solo carattere.

Nella Figura 7.4 vediamo i comandi che si possono inserire a questo punto. Essi fanno parte di un gruppo specifico — il gruppo 1 se il comando "1" è stato usato per inserire questo tipo di domanda, o il gruppo 2 se si è usato il comando "2".

Il comando E

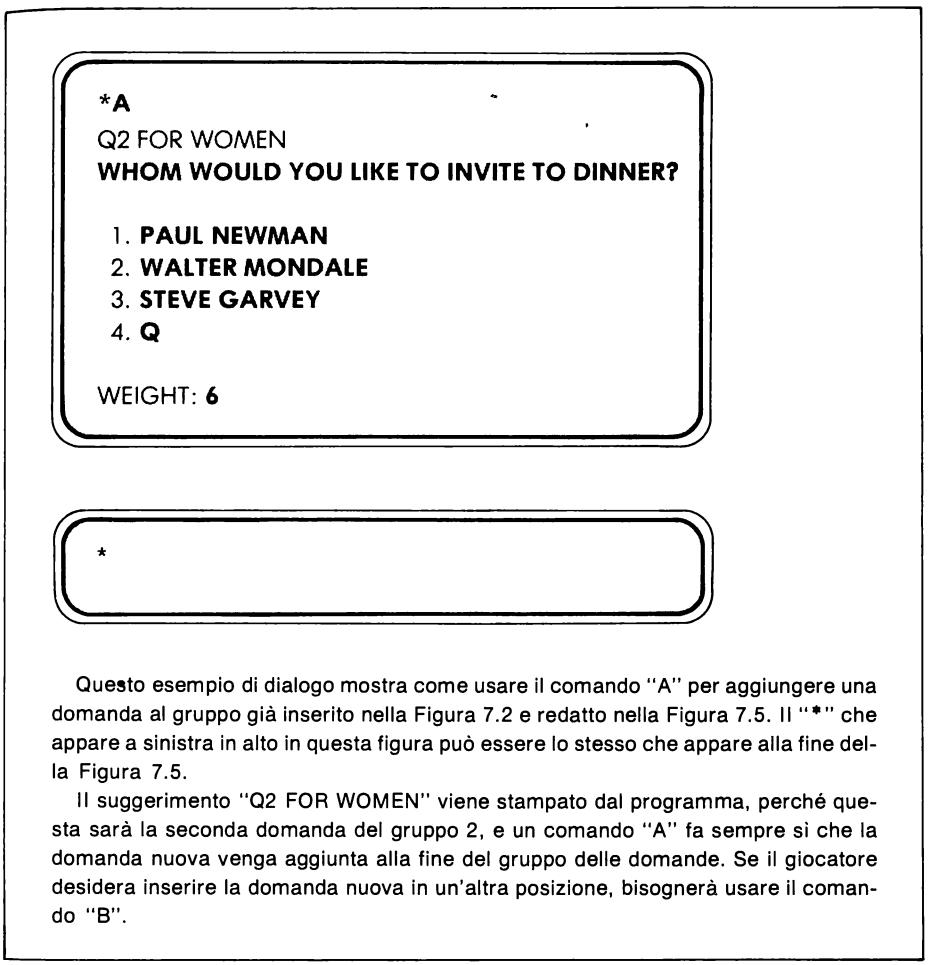
La Figura 7.5 illustra un esempio dell'uso del comando E. Qui il giocatore prepara la domanda che è stata inserita per il gruppo 2 nella Figura 7.2. La domanda è lasciata identica. La prima risposta a scelta viene rimpiazzata da una nuova, la seconda rimane uguale, e la terza viene eliminata, così le risposte che in origine erano la numero quattro e la cinque sono adesso rispettivamente la numero tre e la numero quattro. La nuova numero tre (prima la numero quattro) viene cambiata, e la nuova numero quattro (prima la numero cinque) rimane uguale. Poichè ora ci sono solo quattro scelte possibili, il programma suggerisce di inserirne una quinta, ma il giocatore schiaccia il tasto RETURN, così non è più possibile inserire altro. Allora il programma mostra il peso numerico relativo alla domanda, e il giocatore inserisce un nuovo valore.

I comandi A e B

I comandi A e B sono usati per incorporare delle domande in più nel gruppo di domande già formulate per ciascun gruppo. Il comando A viene usato per raggiungere una domanda alla fine dell'intero gruppo. (Vedi Figura 7.6). Il programma, che già conosce il numero delle domande del gruppo, suggerisce un numero di domande più alto. Il comando B viene usato per inserire una domanda nel gruppo delle domande già esistenti. (Vedi Figura 7.7). A questo punto, il programma non sa dove deve inserire la domanda, dunque comincia con il chiedere al giocatore il numero che precederà la nuova domanda. Per esempio, se il giocatore specifica che l'inserzione deve avvenire prima della domanda numero due, dunque la nuova domanda diventa la numero due, e quella che era la numero due diventa la numero tre, la numero tre diventa la numero quattro e così via. Sia con il comando A che con quello B, una volta determinato il numero della domanda, tutte le altre informazioni vengono inserite allo stesso modo del comando I (Figura 7.2).

L'uso della lettera B (invece che I) per specificare il comando di inserzione della domanda necessita di qualche spiegazione. Una delle ragioni a favore dell'uso di B

invece che di I è che se il giocatore dimenticasse di usare i comandi 1 e 2 e inserisse il comando di inserimento in risposta al "::" (un errore abbastanza frequente e facile da fare), allora "B" darebbe luogo ad un rigetto senza danni del comando (poichè non esiste B come risposta a "::"). D'altra parte, "I" darebbe luogo ad una chiamata non desiderata del comando di inizializzazione, che causerebbe l'inizio da zero di tutto il gioco. La seconda ragione per l'uso di B invece di I è che il valore mnemonico di B (che sta per "before" 'prima') aiuta il giocatore a ricordare che la nuova domanda sarà inserita prima della domanda specificata nel comando. È facile dimenticare se l'inserzione deve andare prima o dopo la domanda specificata, specialmente se avete usato un sistema di editing che segue delle convenzioni diverse per l'inserzione.



Questo esempio di dialogo mostra come usare il comando "A" per aggiungere una domanda al gruppo già inserito nella Figura 7.2 e redatto nella Figura 7.5. Il "*" che appare a sinistra in alto in questa figura può essere lo stesso che appare alla fine della Figura 7.5.

Il suggerimento "Q2 FOR WOMEN" viene stampato dal programma, perché questa sarà la seconda domanda del gruppo 2, e un comando "A" fa sempre sì che la domanda nuova venga aggiunta alla fine del gruppo delle domande. Se il giocatore desidera inserire la domanda nuova in un'altra posizione, bisognerà usare il comando "B".

Figura 7.6 — Aggiungere una domanda

```
*Q
```

(Il giocatore finisce la redazione delle domande per il gruppo 2)

```
:: 1
```

(Il giocatore finisce la redazione delle domande per il gruppo 1)

```
*B
AHEAD OF QUESTION NUMBER: 1
Q1 FOR MEN
HOW OLD ARE YOU?

 1. UNDER 25
 2. 25-38
 3. 39
 4. OLD ENOUGH TO KNOW BETTER
 5. AT THE DANGEROUS AGE

WEIGHT: 9
```

```
*
```

Questo esempio di dialogo illustra l'uso del comando "B". Il giocatore ha finito di redigere le domande per il gruppo 2; per esempio "*" all'inizio può essere lo stesso di quello alla fine della Figura 7.6. Poi il giocatore usa il comando "Q" per ritornare all'"::", e poi usa il comando "1" per generare il "*" per la redazione delle domande per l'intero gruppo 1.

Se l'archivio delle domande per il gruppo 1 è lo stesso di quello inserito nella Figura 7.2 allora le domande uno e due che sono state inserite diventano ora la due e la tre, e la domanda qui inserita diventa la numero uno.

Figura 7.7 — Inserire una domanda

Il comando D

Il comando finale che viene usato per l'editing della domanda è il comando D, che sta per "delete" (cancellare) (Vedi Figura 7.8). Il giocatore specifica il numero della domanda che va cancellata, e il programma fa il resto. La domanda viene rimossa dal gruppo, e ciascuna domanda che ha un numero maggiore di quella cancellata avrà il numero ridotto di uno. Dunque, se la domanda numero quattro viene cancellata, la numero cinque diventa la nuova numero quattro, la sei diventa la nuova numero cinque, e così via.

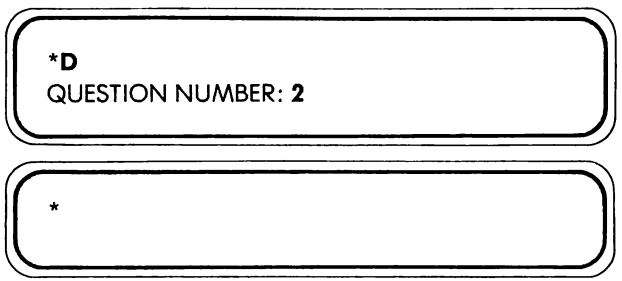
Ciò completa la nostra discussione sul set di comandi che sono usati per l'editing dei file della domanda del gruppo 1 e del gruppo 2. Questi comandi sono un set completo di funzioni di editing (nel senso che tutto ciò che è necessario viene fornito), ma vi sono mancanti parecchie caratteristiche assolutamente auspicabili. Alcune di esse sono suggerite come possibili miglioramenti alla fine del capitolo.

LA FASE DI GIOCO

La fase successiva è la fase di gioco vera e propria, alla quale si arriva impostando il comando Q (quit) in risposta a (prompt) "::". In questa fase il programma segnala un (prompt) "[ ]" pronto, e accetterà comandi costituiti da un singolo carattere, usati per stabilire e modificare le scelte del giocatore e per scegliere coppie compatibili di giocatori. La figura 7.9 mostra i comandi che possono essere dati in risposta al (prompt) "[ ]".

Il comando N

Nelle Figure 7.10 e 7.11 vediamo un esempio del dialogo che ha luogo quando si



Questo è un esempio di dialogo usato per cancellare una domanda dal suo gruppo. Se l'interazione di cui sopra avviene immediatamente dopo quella illustrata nella Figura 7.7, allora la domanda "WHOM DO YOU MOST RESPECT?", che era stata inserita come domanda numero uno nella Figura 7.2 ed è poi diventata domanda numero due dopo l'inserzione che abbiamo visto nella Figura 7.7, sarà cancellata dalle domande del gruppo 1. La domanda "WHAT DO YOU WANT MOST FROM LIFE?" diventa così la numero due ancora una volta.

Figura 7.8 — Cancellare una domanda

chiama il comando N. Il programma inizia col mostrare i nomi dei due gruppi, e chiede poi al giocatore di specificare quale dei due inserendo un 1 o un 2. Poi il programma chiede il nome del giocatore. Una volta dato il nome, al giocatore vengono presentate le domande pertinenti al gruppo da lui specificato. Il giocatore deve scegliere una risposta a ciascuna delle domande. Dopo aver dato il proprio nome e dopo aver scelto le risposte alle domande presentategli, (Figura 7.10), il giocatore deve dare delle preferenze che serviranno per determinare la sua compatibilità con un membro dell'altro gruppo. Nell'esempio di cui alla Figura 7.11, il giocatore può assegnare un valore di 2, o 1, —1 o —2 a ciascuna risposta. I valori sono definiti nelle dichiarazioni DATA nella routine di inizializzazione del programma, cosicchè questo aspetto del gioco possa essere facilmente alterato. Il programma usa il formato

```
VALUE n      CHOICE:
```

per domandare la scelta, se esiste, alla quale il giocatore voglia assegnare il valore n.

COMANDO	AZIONE
E	*Redazione:* Il contenuto dell'inserimento del giocatore viene mostrato sullo schermo, e il giocatore può fare dei cambiamenti o aggiungere o eliminare delle parti.
L	*Load:* Questo comando non viene usato nella versione del programma che abbiamo illustrato in questo libro. Un gruppo di inserimenti effettuati dal giocatore viene inserito usando dei dischi o delle cassette.
N	*Nuovo giocatore:* Si inserisce una parte nuova in un certo gruppo. L'inserimento consiste del nome del giocatore, delle risposte del giocatore che appartengono al gruppo, e le preferenze del giocatore per le risposte di membri che appartengono all'altro gruppo. Tutti questi dati vengono inseriti dal nuovo giocatore in risposta al suggerimento del programma.
P	*Coppia:* Due lotti di giocatori appartenenti all'altro gruppo vengono mostrati: quelli le cui risposte meglio si accoppiano a quelle del giocatore, e quelli le cui preferenze meglio si accoppiano con quelle del giocatore.
Q	*Smettere:* Il gioco è finito. Ritornare al "::" per iniziare una nuova partita.
S	*Registrare:* Questo comando non viene usato nella versione del programma che appare in questo libro. Gli inserimenti del giocatore possono essere registrati su dei dischi o delle cassette.

Figura 7.9 — Risposte al suggerimento "[ ]"

```
[ ] N

 1. MEN
 2. WOMEN
GROUP: 1
```

(Il giocatore è nel gruppo MEN)

```
NAME: JOHN
```

(Il nome del giocatore è JOHN)

```
HOW OLD ARE YOU?
 1. UNDER 25
 2. 25-38
 3. 39
 4. OLD ENOUGH TO KNOW BETTER
 5. AT THE DANGEROUS AGE
CHOICE: 2
```

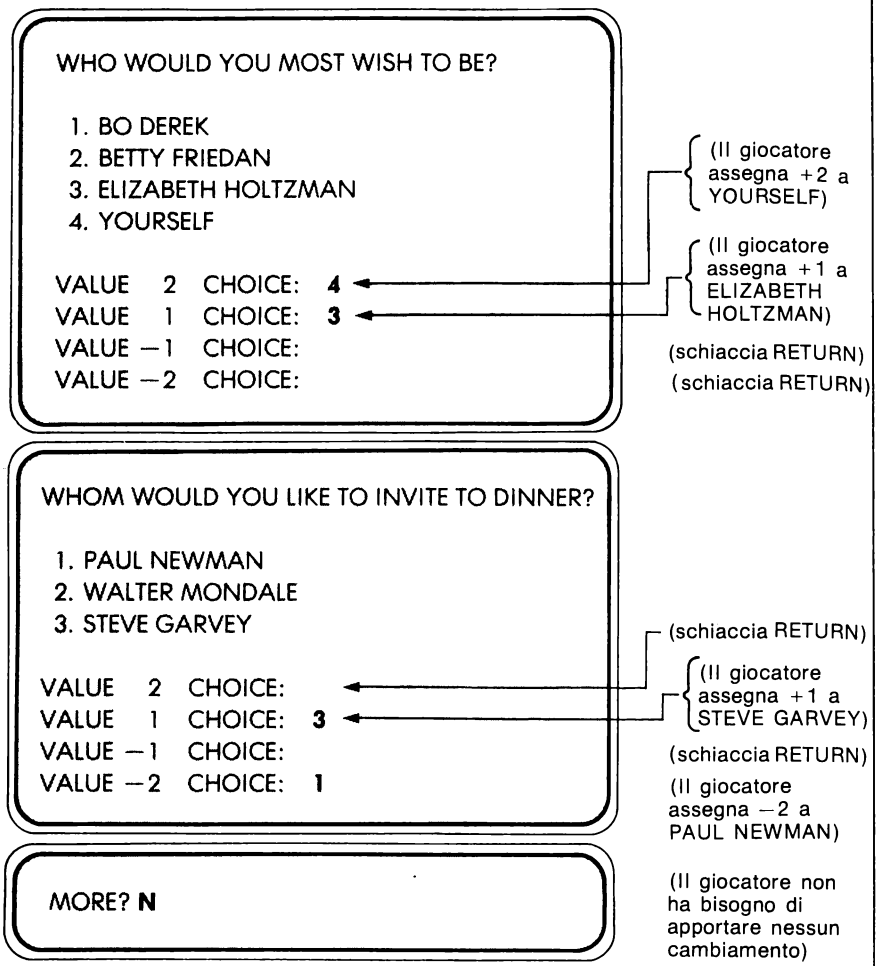
(Ha un'età compresa tra i 25 e i 38 anni)

```
WHAT DO YOU WANT MOST FROM LIFE?
 1. FAME
 2. FORTUNE
 3. HAPPINESS
CHOICE: 3
```

(Quello che desidera di più è HAPPINESS)

Questo esempio mostra la parte iniziale del dialogo del nuovo giocatore. Prima di tutto vengono mostrati i nomi dei due gruppi, e il giocatore specifica di far parte dei "MEN". Poi il programma domanda il nome del giocatore. La risposta è JOHN. Poi il programma mostra le diverse domande che formano il gruppo 1 e JOHN dà una risposta. POi il dialogo procede fino al punto in cui JOHN deve dare informazioni su come un membro del gruppo WOMEN dovrebbe rispondere alle domande del gruppo 2 per essere compatibile. Questo dalogo viene illustrato nella Figura 7.11.

Figura 7.10 — Le risposte di JOHN



Questo esempio mostra la parte del dialogo del nuovo giocatore a partire da dove termina la Figura 7.10. JOHN, che fa parte del gruppo dei MEN esamina le domande che verranno poste ai membri del gruppo 2 (WOMEN) e ha diritto ad assegnare dei punteggi positivi o negativi alle scelte possibili per ciascuna domanda. In questo esempio ci sono quattro punteggi possibili (2, 1, −1, −2) e JOHN può assegnare una scelta diversa a ciascuno. Il numero dei punteggi (4) e i numeri 2, 1, −1, −2 appaiono nelle dichiarazioni DATA nella routine d'inizializzazione. Le scelte a cui non è stato assegnato nessun punteggio hanno valore zero. Questi punteggi vengono usati per misurare la compatibilità di ciascuna risposta dei giocatori del gruppo 2 con le preferenze di JOHN.

Figura 7.11 — Le preferenze di JOHN

[ ] **N**
1. MEN
2. WOMEN
GROUP: **2**

NAME: **SUSAN**

WHO WOULD YOU MOST WISH TO BE?

1. BO DEREK
2. BETTY FRIEDAN
3. ELIZABETH HOLTZMAN
4. YOURSELF
CHOICE: **2**

WHOM WOULD YOU LIKE TO INVITE TO DINNER?

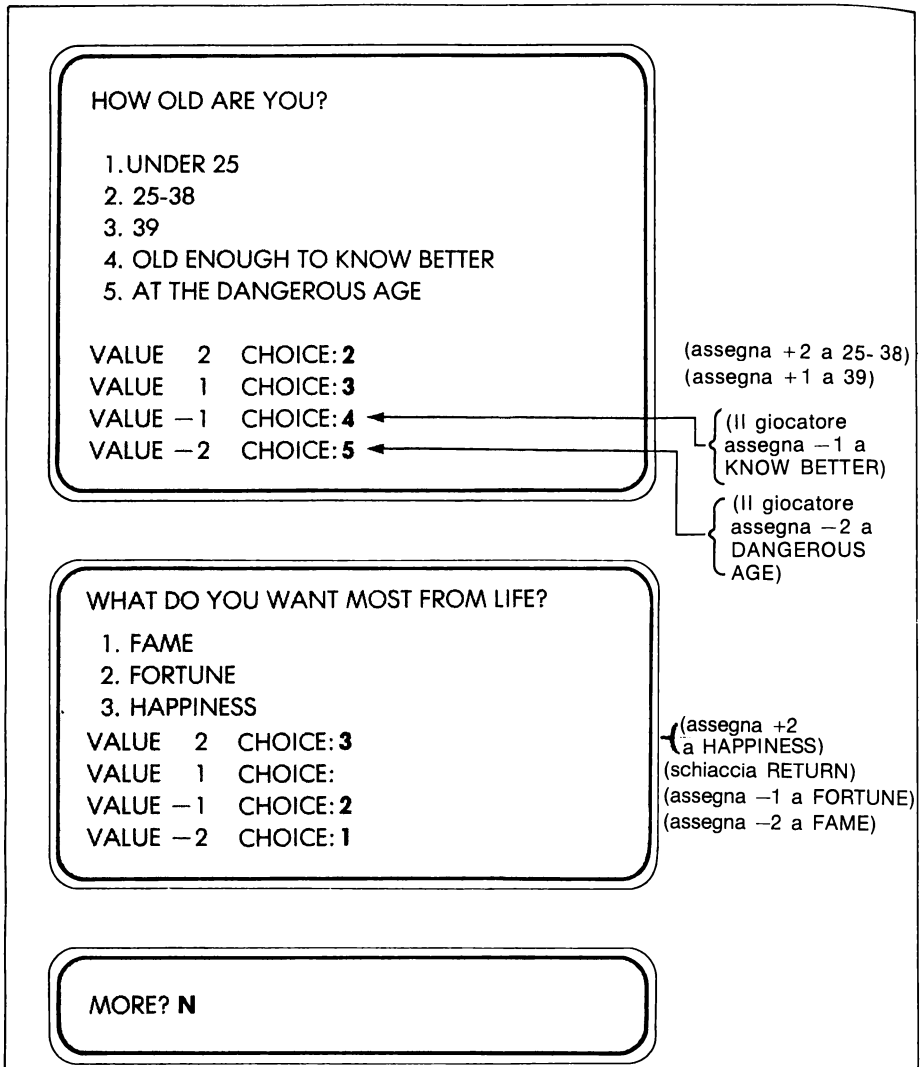
1. PAUL NEWMAN
2. WALTER MONDALE
3. STEVE GARVEY
CHOICE: **1**

Questa è la prima parte dell'inserimento del nuovo giocatore che fa parte del gruppo 2. Qui vediamo come le risposte di SUSAN si sono piazzate rispetto alle preferenze di JOHN:

Domanda 1:	(PUNTEGGIO:	0)	×	(PESO: 7)	=	0
Domanda 2:	(PUNTEGGIO:	−2)	×	(PESO: 6)	=	−12
TOTALE						−12

Nella Figura 7.13 le preferenze di SUSAN vengono usate per confrontarle con le risposte di JOHN.

Figura 7.12 — Le risposte di SUSAN



In questo esempio di dialogo, SUSAN ha inserito delle preferenze per le risposte date dai membri del gruppo 1 (MEN). Qui vediamo come le risposte di JOHN si sono piazzate rispetto alle preferenze di SUSAN:

Domanda 1:	(PUNTEGGIO:	0)	×	(PESO:	9)	= 18
Domanda 2:	(PUNTEGGIO:	2)	×	(PESO:	6)	= 12
TOTALE						30

Figura 7.13 — Le preferenze di SUSAN

```
[ ] P
 1. MEN
 2. WOMEN
GROUP: 1
```

(Il giocatore fa parte del Gruppo 1)

```
 1. JOHN
 2. BILL
 3. HARRY
 4. GEORGE
NUMBER: 1
```

(Il nome del giocatore è JOHN)

```
WOMEN BEST FOR JOHN

    20      BETTY
    20      CAROL
     2      ANN
     0      JANE
    -5      SALLY
   -12      SUSAN

WOMEN JOHN IS BEST FOR

    30      SUSAN
    30      JANE
    30      ANN
    30      CAROL
     6      SALLY
     0      BETTY
```

Questo esempio mostra come usare il comando "P" per ottenere due liste: quella dei giocatori dell'altro gruppo che si sono avvicinati di più ai desideri di JOHN, e quella dei giocatori dell'altro gruppo i cui desideri corrispondono o si avvicinano di più alle risposte di JOHN. Il numero che appare alla sinistra di ciascun nome è il punteggio corrispondente.

Figura 7.14 — Accoppiamento

In modo da illustrare il metodo di punteggio che abbiamo usato, nelle Figure 7.12 e 7.13 vediamo un altro esempio del dialogo che segue un comando N. Questa volta il giocatore appartiene all'altro gruppo. Le risposte di questo giocatore vengono paragonate con le preferenze del primo (Figura 7.11 e 7.12). Per poter conteggiare il punteggio delle risposte del giocatore B contro quelle del giocatore A, si dà un valore numerico a ciascuna delle domande a cui risponde il giocatore B e questi valori vengono poi sommati per formare un punteggio totale. Il valore assegnato a ciascuna domanda si trova moltiplicando il valore che il giocatore A ha specificato in precedenza per ogni scelta, moltiplicato il peso numerico assegnato alla domanda nella fase di preparazione del gioco.

Il comando P

Il comando P viene usato per ottenere un display di due liste di nome (vedi Figura 7.14):

- quei membri dell'altro gruppo le cui risposte si accoppiano ai desideri del giocatore
- quei membri dell'altro gruppo ai quali meglio si adattano le risposte del giocatore.

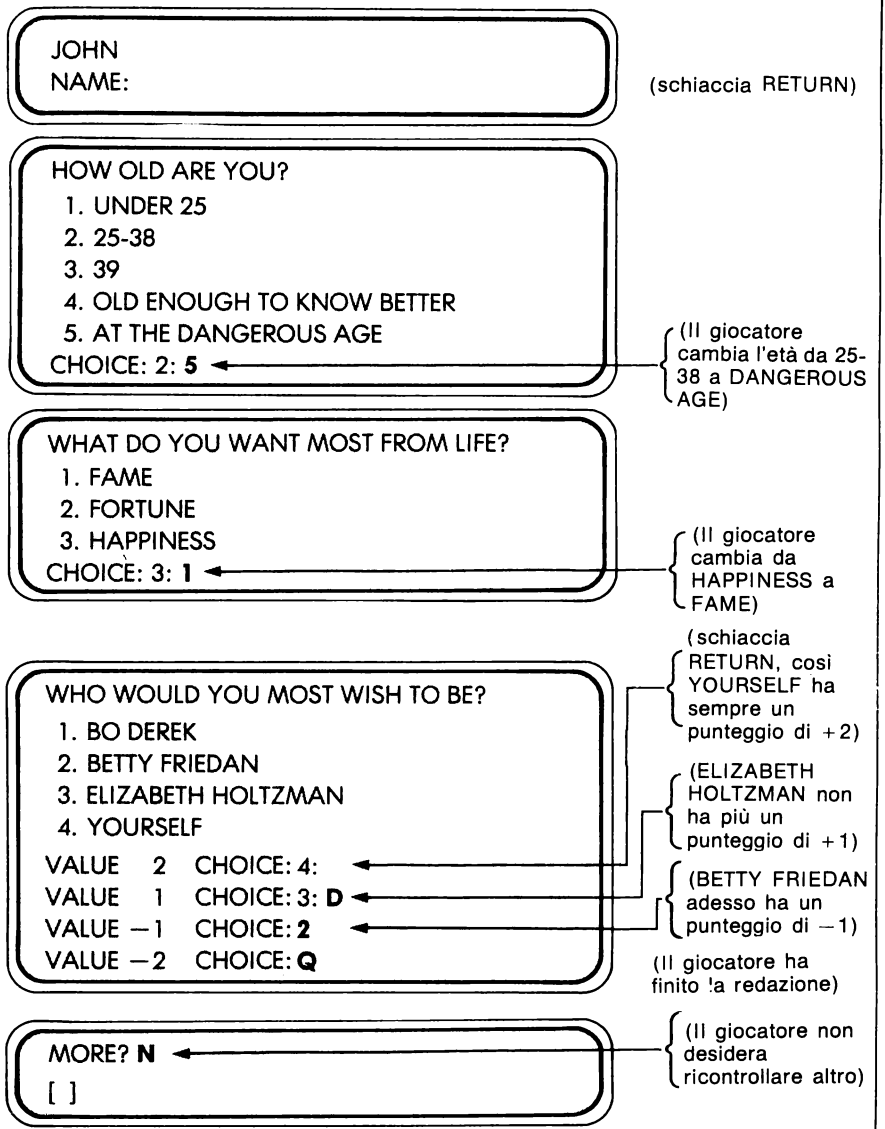
Queste informazioni rimangono sullo schermo finchè il giocatore non schiacci qualsiasi carattere della tastiera. Dopo di che riappare il "[ ]".

Il comando E

Il comando E può essere usato in risposta a "[ ]" per cambiare un inserimento già effettuato. Prima di poterlo fare, però, bisognerà determinare l'identità del giocatore. Prima di tutto il programma richiede il gruppo al quale il giocatore appartiene, e vengono così elencati tutti gli appartenenti a questo gruppo, poi il programma chiede il numero corrispondente al giocatore (vedi Figura 7.14). Nell'esempio che abbiamo riportato, sono elencati solo alcuni nomi, ma ce ne potrebbero essere molti di più, persino troppi perchè ci possano stare tutti sullo schermo. In questo caso viene stampata una lista parziale e il programma domanderà "NUMBER", se viene schiacciato il tasto RETURN o se viene inserito qualcosa che non sia un numero, sullo schermo appariranno le informazioni seguenti.

Dopo aver determinato l'identità del giocatore, si ripete il dialogo. Ogni volta compaiono le informazioni adeguate e poi il programma accetta il cambiamento. Questa è la stessa funzione di editing che si sarebbe inserita se la risposta alla domanda "MORE?" (vedi Figura 7.11 e 7.13) fosse "Y". Potete vedere un esempio di questo tipo di dialogo nella Figura 7.15.

Questo conclude così la nostra presentazione dei comandi per giocare agli Accoppiamenti. Mentre i comandi sono lunghi e complicati, il gioco in sè e per sè è abbastanza semplice.



Questo esempio di dialogo mostra come il giocatore JOHN che fa parte del gruppo 1, fa delle scelte nuove per entrambe le domande del gruppo 1. JOHN cancella una scelta e ne aggiunge un'altra quando specifica le preferenze per le risposte alla prima domanda del gruppo 2.

Figura 7.15 — Redazione dell'inserimento del giocatore

IL PROGRAMMA DEL GIOCO DEGLI ACCOPPIAMENTI

Il gioco degli Accoppiamenti è implementato dal programma illustrato nelle Figure che vanno dalla 7.16 alla 7.59. Vi sono più di quaranta subroutines (persino escludendo le "matrici" usate per funzioni di archivio esterno), e più di 255 linee di comandi BASIC, derivate da più di 500 linee di descrizioni in Free BASIC. A causa di tutte queste subroutines, il programma del gioco degli Accoppiamenti è un ottimo esempio del tipo di subroutine con passaggio di discussione che è tipica del BASIC: l'uso di discussioni globali "implicite". Questa convenzione è terribilmente rigida e porta facilmente ad errori, ma se dovete imparare a programmare in BASIC, dovete saper usarla senza commettere errori.

Il programma del gioco degli accoppiamenti è così vasto che può essere chiamato un programma "vero e proprio"; spiega diverse tecniche e principii della strutturizzazione e della progettazione dei programmi che sono difficili da trovare in programmi più piccoli. Ecco alcuni degli aspetti della programmazione che potrete imparare studiando il programma del gioco degli accoppiamenti.

- Concetti di progettazione
  - Modalità
  - Struttura "sottosopra"
  - Isolamento delle funzioni
  - Possibilità di alterazioni
  - Uso delle matrici per definire la struttura del programma

- Tecniche di programmazione
  - Progettazione di documentazione necessaria sia al programmatore che alla persona che userà il computer in seguito
  - Progettazione di sistemi di interazione
  - Correzione degli errori

- Tecniche di programmazione specifiche
  - Progettazione di funzioni di editing
  - Algoritmi di inserimento e di cancellatura
  - Progettazione di funzioni di controllo degli archivi
  - Risparmio di spazio necessario

Ora parleremo di questi punti in dettaglio.

```
#Il gioco degli Accoppiamenti

      GOSUB init                                  #fissare gli arrays e le
                                                  costanti
      repeat{
        GOSUB create                              #preparare o redigere
                                                  l'archivio delle domande
        repeat{                                   #giocare
          GOSUB clearscreen
          PRINT "[]";                             #mostrare il segnale di
                                                  suggerimento
          GOSUB onech: PRINT X$                   #ottenere e ripetere il
                                                  comando
          IF case
                X$ = "E" THEN GOSUB editplayer    #redigere un inserimen-
                                                  to di un giocatore già e-
                                                  sistente
                X$ = "N" THEN GOSUB newplayer     #aggiungere un nuovo
                                                  giocatore
                X$ = "P" THEN GOSUB pairup        #mostrare gli accoppia-
                                                  menti per un giocatore
                X$ = "L" THEN GOSUB loadplayers   #riempire un archivio
                                                  per il giocatore
                X$ = "S" THEN GOSUB saveplayers   #registrare un archivio
                                                  per il giocatore
                X$ = "Q" THEN                     #finire la partita
                  PRINT "REALLY?"
                  GOSUB onech
                  IF X$ = "Y" THEN
                      break
                  }
                  }
          }
      }
100   GOSUB2260
110   GOSUB1420
120   GOSUB2200:PRINT"[]";:GOSUB2180:PRINTX$
130   IFX$="E"THENGOSUB210:GOTO120
140   IFX$="N"THENGOSUB260:GOTO120
150   IFX$="P"THENGOSUB970:GOTO120
160   IFX$="L"THENGOSUB2230:GOTO120
170   IFX$="S"THENGOSUB2240:GOTO120
180   IFX$<>"Q"THEN120
190   PRINT "REALLY?";:GOSUB2180
200   IFX$="Y"THEN110
205   GOTO120
```

Questo è il loop principale del gioco degli Accoppiamenti. La subroutine "create" contiene una selezione di comando simile per riempire un archivio di domande. Il loop più interno del programma qui illustrato ha comandi per giocare, per riempire un archivio per i giocatori e ottenere degli stampati delle coppie.

Figura 7.16 — Gli Accoppiamenti

Concetti generali di progettazione

La prima categoria che fa parte della lista qui sopra viene chiamata Concetti Generali di Progettazione. Sotto questo titolo raggrupperemo tutti gli aspetti della progettazione che valgono per tutti i programmi.

Modularità Questo termine non ha una definizione precisa. Qui viene usato per segnalare il fatto che il programma del Gioco degli Accoppiamenti è stato diviso in piccole unità (anche evidenziato dalla sua descrizione in più di quaranta figure autonome). Ciascuna di queste unità ha un compito ben preciso, e sono interattive rispetto ad altre unità del programma. Questa loro aderenza ad interfacce ben definite di altre unità del programma si dimostra particolarmente importante nel BASIC, poichè ciascun programma può riferirsi a ogni veriabile.

La struttura "sottosopra" La struttura "sottosopra" viene illustrata dall'arrangiamento del programma del gioco degli Accoppiamenti. Il programma "principale" compare per primo, poi vengono le subroutine che sono state chiamate dal programma principale, poi le subroutine chiamate dalle subroutines del programma principale, e cosi via. Naturalmente, poichè la struttura che abbiamo appena descritto è una struttura ramificata a due dimensioni (cioè un "albero"), mentre le pagine di un libro si susseguono l'una all'altra in ordine lineare, l'arrangiamento vèro e proprio è solo simile a quello descritto.

Questo tipo di arrangiamento del programma è utile per il lettore, poichè rende il programma più facile da capire. Ciascun "livello" del programma è composto di chiamate a programmi di livello più basso che compiono azioni sempre più specializzate.

Il grado di astrazione viene ridotto a ciascun livello del programma: al livello più alto il programma si occupa di concetti astratti come l'addizionare i giocatori o il trovare l'accoppiamento di giocatori ideale; a quello più basso, vengono usati dei comandi in BASIC specifici per manipolare delle variabili specifiche.

Tuttavia, il termine "sottosopra" si riferisce non solo alla struttura del programma. Si riferisce anche all'ordine nel quale si sviluppano le parti del programma. Infatti, il programma del Gioco degli Accoppiamenti è stato sviluppato quasi nello stesso ordine in cui le unità del programma appaiono in questo libro.

Nonostante abbiamo provveduto in questo libro a dare diversi esempi di progettazione di un programma e del suo sviluppo, non ci è possibile coprire tutti gli aspetti in materia. Per allargare le vostre conoscenze sullo sviluppo della struttura "sottosopra" e diversi metodi di sviluppo, vi consigliamo di consultare le diverse opere di Dijkstra Wirth, Yourdon ed altri sulle tecniche di sviluppo dei programmi. Per esempio, How to Manage Structured Programming di Ed Yourdon contiene una presentazione particolarmente chiara di tutto questo materiale.

```
#Redazione di un inserimento di un giocatore già esistente — comando "E"

editplayer   GOSUB identity                    #chiedere (PP,GP)
             IF XX = 0 THEN                    #può scappare se c'è
                                               errore
               RETURN
             GOSUB othergroup                  #GC = "altro" gruppo
                                               per GP
             GOSUB loadwork                    #inserire in spazio di la-
                                               voro
             repeat {
               GOSUB changes                   #dare i cambiamenti al-
                                               l'archivio
               GOSUB askok                     #domandare se c'è altro
                                               da fare
             } until (OK = 1 OR OK = -1)       #aggiornare o abortire
             IF OK = 1 THEN
               GOSUB update                    #aggiornare l'inseri-
                                               mento proveniente dallo
                                               spazio di lavoro
             RETURN

210  GOSUB1310:IFXX=0THENRETURN
220  GOSUB1290:GOSUB340
230  GOSUB460:GOSUB510:IFNOT(OK = 1OROK = - 1)THEN230
240  IFOK = 1THENGOSUB380
250  RETURN
```

La routine "editplayer" è la routine di redazione per un inserimento di un giocatore. La routine "identity" ottiene un indice PP per il giocatore e un indice GP per il gruppo a cui appartiene tramite il dialogo con il giocatore. La routine "othergroup" computa l'indice GC e lo controlla con il gruppo. La routine "loadwork" prende gli elementi dell'array che corrispondono al giocatore (PP,GP) e li piazza in uno spazio dove possono essere manipolati senza alterare gli arrays. La routine "changes" passa sopra i dati, così da permettere che essi siano redatti. Poi viene chiamata la routine "askok", questa usa la variabile OK per specificare: "changes OK, update the file" (i cambiamenti apportati sono OK, aggiornare l'archivio) OK = 1, oppure "more editing needed" (è necessario redigerli di più) OK = 0, oppure "abort the editing process" (abortire l'intero processo di redazione) OK = −1. Infine, la routine "update" riscrive il contenuto dell'intero spazio che è stato redatto nell'array da dove proveniva. Se il processo viene abortito prima che venga chiamata "update", gli arrays non saranno alterati.

Figura 7.17 — Redazione di un inserimento di un giocatore

Isolamento delle funzioni Non solo è importante che i programmi siano spezzettati in unità piccole e maneggevoli con compiti e interfacce ben definite, ma è anche importante che sia possibile localizzare le informazioni facilmente.

Per esempio, le subroutines "unpackwish" e "packwish" (Vedi Figura 7.32) sono le uniche in cui in tutto il programma si faccia uso specifico della struttura attuale della codifica di un singolo numero della scelta da parte del giocatore dei valori di ogni preferenza. (Vedi Figura 7.11 e 7.13). Al di fuori di queste routines, il numero codificato viene trattato come un numero indecifrabile, mentre l'accoppiamento da par-

```
#Fare l'inserimento di un nuovo giocatore

newplayer   GOSUB askgroup                  #ottenere il gruppo del
                                            nuovo giocatore (GP)
            GOSUB othergroup                #GC è l'altro gruppo
            GOSUB clearwork                 #inizializzare la zona di
                                            lavoro
            repeat {                        #ottenere nuovi dati, poi
                                            controllarli
              GOSUB changes
              GOSUB askok
            } until (OK = 1 OR OK = -1)     #finchè il giocatore ap-
                                            prova o smette
            IF OK = 1 THEN                  #se approva
              GOSUB storenew                #allora creare un inseri-
                                            mento di array
            RETURN

260         GOSUB1260:GOSUB1290:GOSUB300
270         GOSUB460:GOSUB510:IFNOT(OK = 1OROK = -1)THEN270
280         IFOK = 1THENGOSUB490
290         RETURN
```

La routine "newplayer" crea un inserimento di un giocatore. La routine "askgroup" trova il gruppo a cui il giocatore appartiene (GP), e "othergroup" computa il numero dell'altro gruppo (GC). Poi la routine "clearwork" inizializza l'area di lavoro (cioè la riempie di stringhe di zero e spazi vuoti). Il loop di chiamata per "changes" e per "askok" è identico a quello in "editplayer". Questo perchè "changes" e le sue subroutines sono progettate in modo da redigere degli inserimenti già esistenti o da crearne di nuovi. Se il giocatore è soddisfatto dal contenuto dello spazio, allora la routine "storenew" viene chiamata per dare degli indici d'array all'inserimento e per conservarlo.

Figura 7.18 — Creare un inserimento di un giocatore

te del giocatore tra scelte e valori di ogni preferenza viene rappresentato come un array di numeri di scelta che corrisponde all'array dei valori di ogni preferenza.

La routine "packwish" prende un numero codificato e genera un array di numeri di scelte. Le funzioni di traduzione che passano avanti e indietro tra queste due forme dell'allocazione delle scelte da parte del giocatore vengono isolate nelle due routines illustrate nella Figura 7.32. Questo è un ottimo esempio di isolamento delle funzioni. Vedremo in seguito un altro esempio nell'isolamento della funzione del conteggio dei punti tra i due giocatori rispetto alla funzione del punteggio. (Vedi Figura 7.34).

Possibilità di alterazioni La possibilità di effettuare delle alterazioni è un altro dei concetti generali di progettazione che viene ben illustrato nel programma del gioco degli Accoppiamenti. Quando si progettano dei programmi per computer bisogna ricordare ed applicare la corollaria della Legge di Murphy:

> *Se uno degli aspetti di un programma è assolutamente inalterabile, qualcuno troverà certamente una ragione impellente per cui esso vada cambiato, senza tener conto di quello che la specifica originale raccomanda.*

```
#Pulire lo spazio di lavoro così da poter accogliere un inserimento per il gruppo GP
(l'altro gruppo è GC)

clearwork   WN$ = ""                          #cancellare il nome
            FOR QX = 1 TO NQ(GP)              #cancellare le risposte
                                              alle domande GP
              WA(QX) = 0
              NEXT QX
            FOR QX = 1 TO NQ(GC)              #cancellare i desideri
                                              per le risposte GC
              WW(QX) = 0
              NEXT QX
            RETURN

300   WN$=""
310   FORQX=1TONQ(GP):WA(QX)=0:NEXTQX
320   FORQX=1TONQ(GC):WW(QX)=0:NEXTQX
330   RETURN
```

La routine "clearwork" pulisce lo spazio di lavoro, che consiste del nome del giocatore WN$, dell'array delle risposte del giocatore WA, e l'array dei desideri del giocatore WW.

Figura 7.19 — Pulire lo spazio di lavoro

O, in poche parole:

Se qualcosa non deve essere cambiata lo sarà.

Senza dover spiegare oltre perchè sia utile avere la possibilità di effettuare delle alterazioni, esaminiamo alcune delle tecniche principali usate nel programma del gioco degli Accoppiamenti.

La prima e più importante tecnica usata per permettere delle alterazioni è quella di ottenere un programma scritto chiaramente, che sia ben strutturato e ben documentato.

Un altro punto importante è l'evitare l'uso di costanti numeriche nel programma, in modo da poter lasciare spazio per eventuali alterazioni e cambiamenti. Nella maggior parte dei programmi BASIC la possibilità maggiore di applicare questa tecnica è la specifica di limiti dei loops e di dimensioni degli arrays. Nel programma del gioco degli Accoppiamenti, le dimensioni degli arrays e i limiti del loop più altro sono quasi sempre variabili; nelle poche occasioni in cui non lo sono, sono dei parametri (cioè delle costanti simboliche). I limiti del loop più basso, invece, sono quasi sempre specificati da costanti (di solito 1 o 0). Questo sistema è più facile da capire di quello che

```
#Riempire lo spazio di lavoro con l'inserimento del giocatore (PP,GP); l'altro gruppo
è GC

loadwork    WN$ = NM$(PP,GP)                    #inserire il nome
            FOR QX = 1 TO NQ(GP)                #inserire le risposte alle
                                                 domande GP
              WA(QX) = A(QX,PP,GP)
              NEXT QX
            FOR QX = 1 TO NQ(GC)                #inserire i desideri per
                                                 le risposte GC
            WW(QX) = W(QX,PP,GP)
              NEXT QX
            RETURN

340  WN$=NM$(PP,GP)
350  FORQX=1TONQ(GP):WA(QX)=A(QX,PP,GP):NEXTQX
360  FORQX=1TONQ(GC):WW(QX)=W(QX,PP,GP):NEXTQX
370  RETURN
```

La routine "loadwork" riempie lo spazio di lavoro con l'array del nome del giocatore, NM$, l'array delle risposte del giocatore, A, e l'array dei desideri del giocatore W. I valori relativi al giocatore ed al gruppo vengono dati da PP e GP.

Figura 7.20 — Riempire lo spazio di lavoro

usa variabili per limiti più bassi, ed un limite minimo di 1 o 0 raramente necessita di cambiamenti.

Un'altra tecnica usata per consentire delle alterazioni future è quella di identificare quegli aspetti di un programma che potrebbero essere soggetti a cambiamenti, e di far sì che tali cambiamenti possano essere effettuati in una sola parte del programma. Per esempio, nel programma del gioco degli Accoppiamenti, il numero massimo delle scelte che viene associato a ciascuna domanda viene rappresentato dalla variabile MC. Questa variabile viene determinata nella subroutine di inizializzazione (vedi Figura 7.59) dai due comandi:

```
READ MQ, MC, MP
DATA maxquestions, maxchoice, maxplayers
```

(Nel programma in BASIC, i valori numerici, invece dei parametri, compaiono nella dichiarazione dei dati (DATA statement). Questa è l'unica parte dove bisognerà effettuare delle alterazioni nel caso in cui bisognerà cambiare il numero massimo delle scelte, poichè attraverso tutto il programma, MC è stata usata nelle dichiarazioni di dimensioni e nei limiti dei loops. Se si fossero usate delle costanti, avrebbero dovuto essere tutte ritrovate e poi cambiate. Inoltre, se il programma fosse pieno di costanti numeriche e se il numero massimo delle scelte dovesse essere cambiato, diciamo, da 3 a 5, la terza scelta che dovrebbe essere cambiata deve essere prima di tutto i-

```
#Aggiornare l'inserimento del giocatore (PP,GP) dallo spazio di lavoro; l'altro è GC.

update      UP = PP: UG = GP: UC = GC          #discussioni per "store-
                                                work"
            GOSUB storework
            RETURN

380   UP=PP:UG=GP:UC=GC:GOSUB420:RETURN
```

"Update" è la routine che fa da compagna a "loadwork". La sequenza di redazione è: GOSUB loadwork, redazione, GOSUB update. La routine "storework" (anche se dal nome può apparire una compagna di "loadwork") usa le discussioni UP, UG per il giocatore e UC per l'"altro" gruppo. La ragione è che "storework" viene chiamata usando discussioni diverse da "storenew"; se PP e GP fossero le discussioni usate da "storework", allora "update" dovrebbe cambiare il valore di PP. Poichè PP ha un suo significato entro la chiamata della routine "update", il cambiamento di PP avrebbe effetto sulla chiamata per "update". Nel BASIC è difficile evitare questi effetti sulle subroutines, ma bisogna a tutti i costi evitarli.

Figura 7.21 — Aggiornare un inserimento di un giocatore dallo spazio di lavoro

dentificata e distinta da una terza che non deve cambiare, e poi alterata. Inoltre, potreste aver scritto

```
IF X > 2
```

invece di

```
IF X > = 3
```

e in questo caso, cambiare il numero massimo delle scelte da 3 a 5 vorrebbe anche significare cambiare la numero 2 in numero 4. Se la dichiarazione avesse incluso anche

```
IF X > MC – 1
```

il cambiamento sarebbe avvenuto automaticamente.

```
#Conservare un nuovo inserimento per il gruppo GP

storenew    UG = GP: UC = GC                    #usare GP e GC attuali
            IF NP(GP) < MP THEN {               #se c'è spazio per un al-
                                                tro giocatore
              NP(GP) = NP(GP) + 1               # aggiungerlo
              UP = NP(GP)
              GOSUB storework
              }
            else                                #altrimenti
              PRINT "SORRY, NO ROOM."           # non aggiungerlo
            RETURN

390   UG=GP:UC=GC
400   IFNP(GP)<MPTHENNP(GP)=NP(GP)+1:UP=NP(GP):GOSUB420:
      RETURN
410   PRINT"SORRY, NO ROOM.":RETURN
```

La routine "storenew" viene chiamata dopo la creazione di un inserimento di un nuovo giocatore. Dà l'indice seguente libero al nuovo giocatore e chiama "storework" per conservare i dati riguardanti il giocatore che si sono raccolti nello spazio di lavoro.

Questa routine mostra un difetto comune di programmazione, il programmatore non ha realizzato in anticipo che sarà necessario portar a termine una prova per sapere se esista lo spazio necessario per conservare un nuovo inserimento (Questa prova deve essere all'inizio della routine "newplayer"). Dunque, il giocatore compie l'intero processo di risposta di domande e di scelta di preferenze, soltanto per sentirsi dire in seguito un qualcosa che il programma avrebbe potuto e dovuto conoscere dall'inizio — SORRY, NO ROOM — spiacente, non c'è posto.

Figura 7.22 — Un programma difettoso per conservare un inserimento di un giocatore

La routine d'inizializzazione di cui alla Figura 7.59 fa sì che il cambiamento avvenga in un altro modo. Tutte le variabili il cui valore dovrebbe cambiare sono state determinate da dichiarazioni DATA. In contrasto, notate che la variabile GM, che rappresenta il numero di gruppi in cui i giocatori sono divisi, viene determinata dal comando

```
GM = 2
```

Una dichiarazione dei compiti viene usata invece di una di dati (DATA) e la costante 2 viene usata invece di un parametro come "maxgroup". Questo perchè il numero dei gruppi non può essere cambiato senza alterare l'intera struttura del gioco. In tutto il programma scriviamo la variabile GM invece di "2" perchè desideriamo che il programma risulti chiaro e facile da capire, e non perchè il valore di GM possa essere alterato.

A questo punto abbiamo spiegato a sufficenza le possibilità di future alterazioni del programma del gioco degli Accoppiamenti. Via via che descriviamo il programma nei suoi particolari, vi troverete davanti a diversi casi in cui sono state usate delle variabili per questo scopo. Diversamente, vi è una costante che viene usata in tutto il programma: la costante stringa "Q". Forse troverete interessante di sforzarvi a cercare di cambiare il carattere di fermata "Q" (usato in diversi punti del dialogo) con qualcosa d'altro. Sapete per esempio perchè si è usata una costante invece di una variabile?

```
#Conservare il contenuto dello spazio di lavoro nell'inserimento del giocatore
(UP,UG); l'altro gruppo è UC.

storework   NM$(UP,UG) = WN$                    #conservare il nome
            FOR QX = 1 TO NQ(UG)                #conservare le risposte
              A (QX,UP,UG) = WA(QX)
              NEXT QX
            FOR QX = 1 TO NQ(UC)                #conservare i desideri
              W(QX,UP,UG) = WW(QX)
              NEXT QX
            RETURN

420   NM$(UP,UG) =WN$
430   FORQX=1TONQ(UG):A(QX,UP,UG) =WA(QX):NEXTQX
440   FORQX=1TONQ(UC):W(QX,UP,UG) =WW(QX):NEXTQX
450   RETURN
```

La routine "storework" conserva l'intero contenuto dello spazio di lavoro nell'array del nome del giocatore NMS, nell'array della risposta A, e in quello dei desideri W. I valori del giocatore e del gruppo vengono dati da UP e UG.

Figura 7.23 — Conservare il contenuto dello spazio di lavoro

L'uso delle matrici per definire la struttura del programma L'ultimo dei concetti di progettazione che toccheremo è l'uso delle matrici per definire la struttura del programma. Questo non è che un altro aspetto del processo di sviluppo "sottosopra", che necessita di definizione e di controllo delle interazioni dei moduli a livello più alto

```
#Chiedere i cambiamenti o le aggiunte da apportare ai dati di un giocatore del grup-
po GP, l'altro gruppo è GC.

changes     GOSUB changename                    #nome del giocatore
            FOR QX = 1 TO NQ(GP)                #risposte del giocatore
              GOSUB changechoice
              IF X$ = "Q" THEN                  #Q = nessun altro cam-
                                                biamento
                break
              NEXT QX
            IF X$ <> "Q" THEN
              GOSUB clearscreen
              PRINT "preference request"; GN$(GC)
              GOSUB onech
              FOR QX = 1 TO NQ(GC)              #desideri del giocatore
                GOSUB changewish
                IF X$ = "Q" THEN                #Q = nessun altro cam-
                                                biamento
                  break
                NEXT QX
            RETURN

460   GOSUB550
470   FORQX=1TONQ(GP):GOSUB680:IFX$<>"Q"THENNEXTQX
480   IFX$="Q"THENRETURN
485   GOSUB2200:PRINT"NOW GIVE YOUR PREFERENCES
      FOR ANSWERS BY";GN$(GC)
490   GOSUB2180:FORQX=1TONQ(GC):GOSUB880:IFX$ <>
      "Q"THENNEXTQX
500   RETURN
```

La routine "changes" chiede al giocatore se ci sono dei cambiamenti da apportare o delle aggiunte da fare all'inserimento esistente nello spazio allocato. (Quando viene chiamata da "newplayer", la routine può solo domandare se ci sono delle aggiunte da fare, perchè lo spazio è vuoto). La struttura della routine è simile a quella di "clearwork", "loadwork" e "storework". La routine "changename" viene chiamata per il nome del giocatore; "changechoice" viene chiamata per le risposte del giocatore alle domande del gruppo GP; "changewish" viene chiamata per le preferenze del giocatore rispetto alle risposte del gruppo GC.

Figura 7.24 — Richiesta dati di cambiamenti da apportare agli inserimenti

prima di effettuare la codifica e il controllo di quelli a livello più basso. In modo da controllare le interazioni dei moduli che non sono stati ancora completamente codificati e controllati, il programmatore elimina le porzioni incomplete di questi moduli e le sostituisce con matrici cioè dei programmi più piccoli le cui interazioni sono simili a quelle delle porzioni incomplete dei moduli, ma il cui comportamento funzionale è solo rudimentale.

Per esempio, i programmi che si occupano di immagazzinaggio esterno di domande o di informazioni relative al giocatore, non sono altro che programmi senza valore che rispondono con un "EXTERNAL STORAGE IS NOT IMPLEMENTED" (L'IMMAGAZZINAGGIO ESTERNO NON E' FUNZIONANTE) ogni qualvolta vengono chiama-

```
#Chiedere se il giocatore desidera rivedere lo spazio per apportare altri cambiamenti

askok       GOSUB clearscreen
            PRINT "MORE?"
            GOSUB onech                          #risposta di un solo carattere

            IF case
              X$ = "Q" THEN OK = -1              #Q = abortire: non conservare lo spazio
              X$ = "Y" THEN OK = 0               #Y = sì, rivedere
            else
              OK = 1                             #altrimenti, OK; nessun cambiamento

            REUTRN
```

```
510   GOSUB2200:PRINT"MORE?";:GOSUB2180:OK=1
520   IFX$="Q"THENOK=-1
530   IFX$="Y"THENOK=0
540   RETURN
```

La routine "askok" domanda al giocatore se è necessario fare dell'altro editing. Se il giocatore risponde "Y", questa routine risponde OK = 0, segnalando così al programma che si desidera chiamare un'altra volta "changes". Se il giocatore risponde "Q", questa routine risponde OK = −1, per segnalare così che l'intero processo è stato abortito e che il contenuto dello spazio deve essere abbandonato senza essere riportato nell'inserimento del giocatore. Qualsiasi altra risposta fa sì che si abbia OK=1. In questo caso, il programma chiamerà "update" o "storenew" per conservare le informazioni racchiuse nello spazio.

Questo gruppo di scelte che la routine fornisce è essenziale per avere un sistema senza intoppi di un ciclo di redazione.

Figura 7.25 — Chiedere se ci sono altri cambiamenti da apportare

ti. Delle matrici più elaborate – potenzialmente molto più utili nel controllo del sistema restante – potrebbero aver creato una domanda rimasta in memoria ogni volta che si richiedeva un carico. Durante il processo di controllo si affrontava il problema da un punto diverso. Il programmatore iniziava con lo scrivere delle matrici per la porzione del programma relativa alla creazione dello stesso (Figura 7.45 - 7.56). Queste matrici facevano sì che si potessero inserire delle domande e delle informazioni relative al giocatore usando solo un minimo di editing. Poi, dopo che si fosse perfezionata la parte del programma relativa al gioco, le matrici venivano allargate ed usate per formare le versioni di cui alle Figure dalla 7.45 alla 7.56. Poichè esisteva un modo di inserire domande e informazioni relativa al giocatore all'inizio del processo di controllo, le matrici per i carichi dell'immagazzinaggio esterno non sono state allargate per creare degli archivi fasulli.

Tecniche di programmazione

Abbiamo parlato di concetti di progettazione che si possono usare in tutte le situa-

Passaggi usati per la creazione del gioco degli Accoppiamenti e per la sua documentazione

- L'autore ha concepito e smussato l'idea originale per il programma. La forma finale è stata decisa prima di iniziare lo sviluppo del programma stesso.
- Le descrizioni in Free BASIC che compaiono nelle Figure dalla 7.16 alla 7.59 sono state prima scritte con matrici molto più semplici per la formulazione delle domande e per le routine di editing delle Figure dalla 7.45-7.56.
- Una traduzione manuale dal Free BASIC al BASIC è stata fatta dall'autore.
- Il programma è stato reso a prova d'errore, e sono stati migliorati sia i formati dello schermo che i dialoghi. Poi dei veri programmi di editing hanno sostituito le matrici. Tutti i cambiamenti sono prima di tutto stati apportati alle descrizioni in Free BASIC, e poi incorporati nel programma in BASIC.
- È stata compilata una lista in BASIC, e il Free BASIC e il BASIC sono stati combinati nelle Figure dalla 7.16 alla 7.59. Allo stesso tempo è stato redatto il materiale descrittivo delle Figure, e questo esame nel dettaglio ha portato al cambiamento di alcuni particolari, prima nel Free BASIC, e poi incorporati nel programma in BASIC.
- È così stato preparato un programma di esempio, durante il suo svolgimento sono state preparate le figure dalla 7.1 alla 7.15 e il testo che le accompagna. Questo processo ha poi portato a diversi cambiamenti nel programma, e le figure dalla 7.16 alla 7.59 sono state cambiate seguendo il programma.

Figura 7.26 – Creazione del gioco degli Accoppiamenti e sua Documentazione

zioni, ma vi sono anche diverse tecniche di programmazione spiegate nel gioco degli Accoppiamenti che possono trovare applicazione in una vasta gamma di programmi.

Progettazione di documentazione necessaria sia al programmatore che alla persona che userà il computer in seguito Forse vi sorprenderà sapere che la progettazione di documentazione necessaria sia al programmatore che alla persona che userà il computer in seguito è uno dei compiti primari da eseguire quando si crea un pro-

```
#Routine della redazione del nome

changename GOSUB clearscreen
           IF WN$ = "" THEN {                    #se non c'è nessun nome nell'archivio
             PRINT "NAME:";                      # ottenerne uno
             repeat {
             GOSUB stringin
               WN$ = SS$
               } until (WN$ <> "")
           }
           else                                  #se c'è un nome nell'archivio
             PRINT WN$                           # mostrarlo sullo schermo
             PRINT "NAME:";
             GOSUB stringin                      #ottenere un nuovo nome
             IF SS$ <> "" THEN
               WN$ = SS$                         #"RETURN" = non cambiare
             }
           RETURN

550  GOSUB2200:IFWN$<>""THEN590
560  PRINT"NAME:";
570  GOSUB610:WN$=SS$:IFWN$=""THEN570
580  RETURN
590  PRINTWN$:PRINT"NAME:";:GOSUB610:IFSS$<>""
     THENWN$=SS$
600  RETURN
```

La routine "changename" mostra al giocatore la stringa che c'è nell'archivio del nome del giocatore e accetta un cambiamento (o con un RETURN non apporta nessun cambiamento).

Figura 7.27 — Redazione del Nome del giocatore

```
#Routine di input della stringa — SS$ = tutti i caratteri sono stati scritti fino a RE-
TURN

Stringin    SS$ = ""                                  #inizializzare SS$
            repeat {
              GOSUB onech                             #attendere il carattere
              IF X$ = "return" THEN break             #fatto se RETURN
              else IF X$ < > "delete" THEN{           #se non è errato
                PRINT X$;                             # ripeterlo
                SS$ = SS$ + X$                        # aggiungere a SS$
                }
              else IF LEN(SS$) < > 0 THEN{            #se ce n'è uno
                PRINT "delete string";                # rimuovere dallo
                                                      schermo
                SS$ = LEFT$(SS$,LEN(SS$) — 1)         # rimuovere da SS$
              } }
            IF SS$ = "Q" THEN                         #rimuovere Q dallo
                                                      schermo
              PRINT "delete string";
            else IF SS$ < > "" THEN PRINT             #ripetere RETURN se la
                                                      linea non è nulla
            RETURN
```

```
610   SS$=""
620   GOSUB2180:IFX$=CHR$(13)THEN660
630*  IFX$<>CHR$(20)THENPRINTX$;:SS$=SS$ + X$:GOTO620
640** IFLEN(SS$) <>0THENPRINTX$;:SS$=LEFT$(SS$,LEN(SS$)
      —1):GOTO620
660*  IFSS$="Q"THENPRINTCHR$(20);:RETURN
665   IF SS$ <> ""THENPRINT
670   RETURN
```

La routine "stringin" è molto simile al comando LINE INPUT SS$.

Quando il programma si trova davanti al carattere (RUBOUT, back arrow, DEL) che è usato per cancellare, deve controllare se ci sono dei caratteri, in SS$ da cancellare. Questa parte è più complicata nella versione per l'Apple, perchè LEFT$(SS$,0) non è possibile in BASIC per l'Apple (Vedere Figura 4.10).

La routine ha due caratteristiche che rendono il suo comportamento diverso da quello di LINE INPUT. Se SS$ = "Q", allora il Q viene cancellato dallo schermo. Se SS$ = "", allora non verrà ripetuto il carattere RETURN; altrimenti si userà un comando PRINT per iniziare una nuova linea.

---

* Nella versione per l'Apple, CHR$(8) rappresenta il carattere per cancellare, e CHR$(8);" ";CHR$(8) rappresenta la stringa per cancellare. Nella versione per il TRS-80, CHR$(8) rappresenta la stringa per cancellare.

** Vedere le note per la Figura 4, per la versione per l'Apple di questa linea.

Figura 7.28 — Input della stringa

gramma. Sfortunatamente di solito questo viene sempre lasciato per ultimo, e mal fatto.

Nella Figura 7.26 vediamo i diversi passi che si sono seguiti nella creazione e nella

```
#Routine per la redazione delle risposte per un membro del gruppo GP; GC = l'altro
gruppo

changechoice FOR QX = 1 TO NQ(GP)
               GOSUB clearscreen
               GQ = GP: GOSUB showquestion          #mostrare la domanda
               PRINT "CHOICE:";
               IF WA(QX) <>0 THEN                   #mostrare la scelta, se
                                                    ce n'è
                 PRINT WA(QX); ":"
               repeat{                              #ottenere Q, D, RE-
                                                    TURN o
                 GOSUB onech                        # un numero di scelta
                                                    valido
                 IF case
                   X$ = "Q" or "return" THEN{       #Q = niente più editing
                     break                          #RETURN = domanda
                                                    seguente
                   X$ = "D" THEN                    #D = eliminare la scelta
                     WA(QX) = 0
                     break
                   }
                 else{                              #numero di scelta valido
                   NN = VAL(X$)                     # selezionare quella
                                                    scelta
                   IF 1 <= NN <= NC(QX,GP) THEN{
                     WA(QX) = NN
                     break
                     }
                   }
                 }
               IF X$ = "Q" THEN
                 break
               NEXT QX
             RETURN
```

La routine "changechoice" presenta le risposte attuali del giocatore, se ce n'è, per ogni risposta. Poi chiede un input. Se riceve un RETURN, significherà nessun cambiamento da apportaré, se riceve un D eliminerà la risposta di cui ci si occupa al momento, e se riceve un Q non effettuerà altri cambiamenti, naturalmente un numero di scelta valido significherà che la selezione di quella scelta va bene.

Figura 7.29 — Alterare le scelte del giocatore

documentazione del gioco degli Accoppiamenti. Notate il modo in cui la programmazione e la documentazione procedono di pari in passo e vengono mantenuti consistentemente allo stesso livello.

La ragione principale per lo sviluppo del Free BASIC è di liberare il programmatore di BASIC dall'uso dei numeri di linee. Il processo illustrato nella Figura 7.26 mostra quanto questo sia importante e come sia stato ottenuto con successo. Questo processo necessita però di ulteriori cambiamenti del programma che possono essere effettuati abbastanza facilmente, poichè il programma in BASIC può essere rinumerato e i comandi che compaiono nelle figura possono essere sostituiti senza causare problemi al testo che le accompagna. La progettazione, lo sviluppo e la documentazione del programma sono stati completamente liberati dall'uso dei numeri di linee. Questi ultimi vengono infatti solo usati nella traduzione da Free BASIC in BASIC, un processo meccanico che potrebbe poi essere persino eseguito da un programma di computer.

Lo scopo dei comandi in BASIC che compaiono nelle Figure dalla 7.16 alla 7.59 è quello di fornire degli esempi concreti del processo di traduzione dal Free BASIC al BASIC. Altrimenti, questi comandi in BASIC, come il codice che accompagna una lista FORTRAN o PL/I, sono solo necessari per facilitare certe forme, del resto poco usate, di controllo dei programmi per verificare l'esistenza di errori.

La discussione di cui sopra sottolinea l'importanza del linguaggio di programmazione nel processo di documentazione. Proprio come la documentazione che sarà poi utile alla persosna che userà il programma in seguito, deve essere preparata con l'aiuto di esempi veri e propri di comportamento del programma, la documentazione necessaria al programmatore deve scaturire dal manoscritto del programma. Poichè

```
680   FORQX=1TONQ(GP):GOSUB2200
690*  GQ = GP:GOSUB 1240:PRINT "CHOICE:";: IFWA(QX) <>
      0 THEN PRINTWA(QX); ": ";
700   GOSUB2180
710   IFX$ = "Q"ORX$ = CHR$(13)THEN750
720   IFX$="D"THENWA(QX)=0:GOTO750
730   NN = VAL(X$):IF1 < = NNANDNN < = NC(QX,GP) THENWA(QX) =
      NN:GOTO750
740   GOTO700
750   IFX$="Q"THENRETURN
760   NEXTQX:RETURN
```

* La versione per l'Apple di questa linea differisce solamente nella spaziatura nella costante di stringa.

Figura 7.29a — BASIC per alterare le scelte

il BASIC non può fornire informazioni rigurdanti la struttura del programma, la tecnica di descrizione del programma in Free BASIC è stata usata in questo libro come la fonte di linguaggio di programmazione. Dunque tutti i cambiamenti effettuati sono stati prima di tutto applicati alla descrizione in Free BASIC, e poi al programma in BASIC. Questo fa sì che la documentazione per il programmatore, che viene derivata dalle descrizioni in Free BASIC, riflette sempre lo stato del programma. Questo è un

```
#Domandare le preferenze del giocatore — nell'array FT

askwish     FOR FX = 1 TO NF
              PRINT: PRINT "VALUE"; FV(FX);" CHOICE";
              IF FT(FX) <> 0 THEN
                PRINT FT(FX);
              PRINT ": ";
              repeat {
                GOSUB onech
                IF X$ = "Q", "N" or "return" THEN
                  break
                else IF X$ = "D" THEN                  #D = cancellare la
                                                       preferenza
                  PRINT X$;                            #ripetere D
                  FT(FX) = 0
                  break
                  }
                else{                                  #il numero è una prefe-
                                                       renza
                  NN = VAL(X$)
                  IF 1 <= NN <= MC THEN
                    PRINT X$;                          #ripetere la scelta
                    FT(FX) = NN
                    break
                    }
                  }}
                }
              IF X$ = "Q" or "N" THEN                  #Q = l'editing è stato
                                                       effettuato
                break                                  #N = domanda se-
                                                       guente
              NEXT FX
            RETURN
```

La routine "askwish" mostra il set attuale di preferenze dall'array FT, e accetta dei cambiamenti o delle aggiunte. Q, D, e RETURN vengono interpretati come in "changechoice"; N significa passare alla prossima domanda.

Figura 7.30 — Domandare le preferenze al giocatore

vantaggio che un linguaggio "autodocumentante" ha rispetto a strumenti descrittivi come gli schemi delle operazioni (flowcharsts): ogni cambiamento deve essere prima effettuato sulla documentazione e poi sul programma.

Progettazione di sistemi di interazione Dalla meta degli anni 60, quando l'uso di sistemi di computer interattivi cominciò a aumentare, la progettazione dell'interazione che ha luogo tra l'utente e il programma è stata trascurata. Molti programmi che sono per altro ben progettati, hanno interazioni che sono state "buttate là" dal programmatore come un sotto-prodotto delle porzioni di elaborazione dei programmi. Altri programmi hanno interazioni che sono modellate sul tipo di input e output più appropriati per un sistema non interattivo. Altri ancora sono progettati per l'uso con telescriventi: ignorando così completamente le proprietà tipiche dei sistemi con display video che invece verranno usati.

Un altro problema comune che si trova in interazioni progettate da programmatori pieni di buone intenzioni è il troppo zelo. Suggerimenti troppo lunghi, offerte di aiuto, liste, e altre distrazioni similari sono certamente utili per un giocatore alle prime armi, ma tutti questi giochetti si trasformano in ostacoli quando l'azione necessaria deve essere veloce e scorrevole. Il loro effetto è come quello dell'addetto teatrale che ha lasciato le luci di platea accese durante la rappresentazione così gli spettatori potevano consultare il programma.

Nello studio del programma del gioco degli accoppiamenti, dovreste notare l'uso frequente di suggerimenti brevi, della pulizia dello schermo e dell'imput di caratteri singoli. Inoltre notate la routine a stringa (Figura 7.28) e il modo in cui viene usata

```
770*  FORFX=1TONF:PRINT:PRINT"VALUE";FV(FX);" CHOICE";
780   IFFT(FX) < >0THENPRINTFT(FX);
790   PRINT": ";
800   GOSUB2180:IFX$=CHR$(13)ORX$ = "Q"ORX$ = "N"THEN850
810   IFX$="D"THENPRINTX$;:FT(FX) = 0:GOTO850
820   NN=VAL(X$)
830   IF1 < = NNANDNN < = MCTHENPRINTX$;:FT(FX) = NN:GOTO850
840   GOTO800
850   IFX$="Q"ORX$="N"THEN870
860   NEXTFX
870   RETURN
```

* La versione per l'Apple di questa linea differisce soltanto nella spaziatura nella costante di stringa.

Figura 7.30a — BASIC per domandare le preferenze

nelle interazioni del programma. Lo scopo principale di questa routine è di sostituire un comando di LINE INPUT che manca in numerosi sistemi in BASIC usati dai piccoli computers. Ha, però, due caratteristiche particolari:

- La rimozione completa del carattere singolo "Q" dallo schermo.
- L'omissione completa di tutti gli input a ripetizione o con stringa azzerata (per esempio, un RETURN che sia stato scritto senza essere preceduto da nessun carattere non verrà ripetuto).

Nella interazione video, le informazioni non devono essere cancellate prima che il giocatore abbia finito di leggerle. Questo può succedere quando più di quella quantità di informazioni che trova posto sul video deve essere emessa dal programma (vedi Figura 7.44 il programma dell'Identità), o quando l'output di information viene cancellato perchè lo schermo viene pulito prima di inserire un nuovo comando.

Le Figure 7.18 e 7.22, le routine "newplayer" e "storenew", e la Figura 7.33, la routine "pairup", sono ottimi esempi di questo problema.

```
#Routine per la redazione delle preferenze di un membro del gruppo GP; l'altro gruppo è GC

changewish  FOR QX = 1 TO NQ(GC)
              GOSUB clearscreen
              GQ = GC: GOSUB showquestion        #stampare la domanda
                                                 e le scelte
              WC = WW(QX): GOSUB unpackwish      #desideri presenti in FT
              GOSUB askwish                      #aggiungere/cambiare
              GOSUB packwish: WW(QX) = WC        #reinserire in WC
              IF X$ = "Q" THEN                   #Q = niente più editing
                break
              NEXT QX
            RETURN

880   FORQX=1TONQ(GC):GOSUB2200
890   GQ=GC:GOSUB1240:WC=WW(QX):GOSUB920:GOSUB770:
      GOSUB950:WW(QX)=WC
900   IFX$ <> "Q" THEN NEXT QX
910   RETURN
```

La routine "changewish" passa attraverso tutte le domande del gruppo GC, mostrando le preferenze del giocatore, se ce ne sono, prendendole dall'array WW e accettando cambiamenti ed aggiunte. Le subroutines "packwish" e "unpackwish" vengono chiamate per tradurre il formato delle preferenze usate nell'array WW in quello delle preferenze manipolate nell'array FT da "askwish".

Figura 7.31 — Redigere le preferenze di un giocatore

```
#Routines per tradurre tra un desiderio in gruppo WC e l'array FT

unpackwish  FOR FX = 1 TO NF
              FT(FX) = WC mod (MC + 1)
              WC = (WC - FT(FX))/(MC + 1)
              NEXT FX
            RETURN
packwish    WC = 0: FM = 1
            FOR FX = 1 TO NF
              WC = WC + FT(FX) * FM
              FM = FM * (MC + 1)
              NEXT FX
            RETURN

920  FOR FX = 1 TO NF:FT (FX) = WC - (MC + 1)* INT(WC/(MC + 1))
930  WC=(WC-FT(FX))/(MC+1):NEXTFX:RETURN
950  WC=0:FM=1:FORFX=1TONF:WC=WC+FT(FX) *FM:FM=FM*
     (MC + 1):NEXT FX
960  RETURN
```

Le routines "packwish" e "unpackwish" passano tra i formati sciolti e in gruppo di diverse preferenze per una domanda. Nel formato sciolto, ciascuna preferenza è un numero tra 0 e MC. Uno zero significa nessuna preferenza, un valore di n(1 < = n < = MC) significa scelta n. Ci sono diverse preferenze e ciascuna ha un valore, questi valori vengono conservati nell'array FV. Quando si calcola il valore numerico di un accoppiamento, ciascuna risposta riceve un punteggio uguale al peso numerico della domanda preso dall'array WT moltiplicato per il valore preso dall'array FV che corrisponde alla prima posizione alla quale compare la risposta nell'array delle preferenze. Se la risposta non è una delle preferenze, riceve un punteggio zero.

Nel formato in gruppo, le preferenze, che sono dei numeri tra 0 e MC, vengono trattate come cifre in un numero a base MC + 1. Cioè ogni preferenza viene moltiplicata per una potenza di MC + 1 e i risultati vengono sommati.

Figura 7.32 — Preferenze sciolte ed in gruppo

```
#Trovare la giusta "metà" della coppia

pairup      GOSUB identity          #il giocatore è (PP,GP)
            GOSUB othergroup        #l'altro gruppo = GC
            GOSUB clearscreen
            GOSUB clearscore        #inizializzare "i punteg-
                                    gi più alti"
```

Figura 7.33 — L'Accoppiamento dei giocatori *(continua)*

```
PW = PP: GW = GP: GA = GC                    #i desideri del giocatore, le altre risposte
FOR PX = 1 TO NP(GC)
  PA = PX: GOSUB score                       #accoppiare (PW,GW) e (PA,GA)
  GOSUB savescore                            #tenere i punteggi più alti
  NEXT PX
PRINT GN$(GC); "BEST FOR"; NM$(PP,GP)
GS = GC: GOSUB printscores                   #stampare i punteggi più alti
GOSUB clearscore                             #inizializzare "i punteggi più alti"
PA = PP: GA = GP: GW = GC                    #le risposte del giocatore, gli altri desideri
FOR PX = 1 TO NP(GC)
  PW = PX: GOSUB score                       #accoppiare (PW,GW) e (PA,GA)
  GOSUB savescore                            #tenere i punteggi più alti
  NEXT PX
PRINT: PRINT GN$(GC);"THAT";NM$(PP,GP); "IS BEST FOR"
GS = GC: GOSUB printscores                   #stampare i punteggi più alti
GOSUB onech: RETURN                          #lasciare sullo schermo finchè non viene schiacciato un tasto
```

```
 970  GOSUB1310:GOSUB1290:GOSUB2200:GOSUB1110
 980  PW=PP:GW=GP:GA=GC
 990  FORPX = 1TONP(GC):PA = PX:GOSUB1050:GOSUB1140:NEXT PX
1000  PRINT GN$(GC);" BEST FOR ";NM$(PP,GP):GS = GC:GOSUB1120
1010  GOSUB1110:PA = PP:GA = GP:GW = GC
1020  FORPX=1TONP(GC):PW=PX:GOSUB1050:GOSUB1140:NEXT PX
1030  PRINT:PRINTGN$(GC);" THAT' ";NM$(PP,GP);" IS BEST FOR":GS =
      GC:GOSUB1120
1040  GOSUB2180:RETURN
```

La routine "pairup" trova la giusta "metà" della coppia nei membri dell'altro gruppo, controllando ed accoppiando le preferenze e le scelte. La routine "score" calcola il punteggio per ogni accoppiamento, la routine "savescore" mantiene un array dei punteggi più alti e la routine "printscores" stampa i nomi dei giocatori nell'altro gruppo i cui punteggi si trovano nell'array dei punteggi più alti.

Figura 7.33 — L'Acccoppiamento dei giocatori *(Fine)*

Un'altra caratteristica importante dei programmi interattivi, usata quando si stanno aggiornando gli archivi e anche durante la fase di editing, è la possibilità di ritirarsi da una azione, sia perchè è stata iniziata erroneamente, o perchè si sono verificati errori durante il processo, o semplicemente perchè il giocatore ha deciso di non completarla. Per esempio, notate che la routine "qnum" (Figura 7.56) può dare il valore QQ = 0, per indicare al programma chiamante che il giocatore vuole terminare l'azione. Per capire l'importanza di questa caratteristica, bisogna considerare un progetto alternativo per "qnum" che chiedesse costantemente informazioni relative al numero della domanda, finchè ne riceva uno valido. Questo si rivelerebbe estremamente frustrante per il giocatore che avesse per errore premuto il tasto "D" che sta per "delete a question" (cancellare una domanda) e poi si trovasse davanti alla richiesta "QUESTION NUMBER:". Poichè esiste "qnum", il giocatore deve solo schiacciare RETURN e il processo di cancellatura verrà fermato.

Il giocatore, nel programma del gioco degli Accoppiamenti, mentre risponde alle domande e richiede un accoppiamento non ha accesso ai comandi che regolano l'editing delle domande, e un giocatore che stia occupandosi dell'editing delle domande per un gruppo, non ha accesso ai comandi che regolano l'editing per l'altro gruppo.

Questo sistema è studiato per proteggere il giocatore nel caso egli inavvertitamente azionasse un comando errato che non si trovi in un gruppo speciale. Tuttavia non esiste alcuna protezione contro la chiamata intenzionale di un comando. Qualsiasi giocatore può muoversi da un suggerimento all'altro quando vuole, poichè i comandi necessari non sono segreti, e non sono protetti in nessun modo. In un caso (Figura

```
#Calcolare il punteggio SC delle risposte di un giocatore (PA,GA) contro i desideri di
(PW,GW)
score       SC = 0                                  #inizializzare il punteggio
            FOR QX = 1 TO NQ(GA)                    #controllare le domande
                                                    del gruppo GA
              WC = W(QX,PW,GW)                      #i desideri di (PW,GW)
              AV = A(QX,PA,GA)                      #le risposte di (PA,GA)
              GOSUB fit
              SC = SC + FT * WT(QX,GA)              #aggiornare il punteggio
              NEXT QX
            RETURN

1050 SC=0:FORQX=1TONQ(GA)
1060 WC=W(QX,PW,GW):AV=A(QX,PA,GA):GOSUB1080
1070 SC=SC+FT*WT(QX,GA):NEXTQX:RETURN
```

La routine "score" chiama più volte "fit" per accoppiare le risposte e le preferenze, e accumula un punteggio SC sommando i diversi punti di contatto.

Figura 7.34 — Il punteggio delle risposte contro le preferenze

7.16) il programma chiede "REALLY?" ed attende che gli sia data una risposta "Y" prima di rispondere ad un comando potenzialmente distruttivo.

Correzione degli errori. La correzione degli errori effettuata nel programma del gioco degli Accoppiamenti ha luogo senza troppo dsiturbare il flusso del gioco. La maggior parte degli input errati che consistono di un solo carattere vengono ignorati e non ripetuti. Nella Figura 7.27 (changename) vediamo un esempio di come viene trattato un input nullo errato (e spiega come "stringin" non ripete il RETURN su un input nullo). Nella Figura 7.22 (storenew) vediamo invece un esempio del modo errato di trattare un errore comune.

Tecniche di programmazione specifiche

Inoltre alle tecniche di programmazione generali discusse nella parte precedente, nel programma del gioco degli Accoppiamenti si parla anche di diverse tecniche di

```
#Misurare i punti di contatto FT tra le risposte AV e i desideri codificati WC

fit         FT = 0                          #inizializzare i punti di
                                            contatto
            GOSUB unpackwish                #tradurre WC nell'array
                                            FT
            FOR FX = 1 TO NF                #controllare gli arrays
                                            FT e FV
              IF FT(FX) = AV THEN {         #se ad AV è stato asse-
                                            gnato un valore
                FT = FV(FX)                 #allora FT = quel valore
                break
                }                           #altrimenti FT viene ini-
                                            zializzato come sopra
              NEXT FX
            RETURN

1080  FT=0:GOSUB920
1090  FOR FX=1TONF:IFFT(FX)=AVTHENFT = FV(FX):RETURN
1100  NEXTFX:RETURN
```

La routine "fit" trova la risposta AV quando si presenta per la prima volta nella lista di preferenze codificate in WC. Queste vengono poi tradotte da "unpackwish" nell'array FT e poi AV viene paragonata agli elementi dell'array.

Figura 7.35 — Il punteggio di una risposta

programmazione specifiche. Queste tecniche possono essere applicate in diversi programmi.

Progettazione di funzioni di editing Nel gioco degli accoppiamenti abbiamo usato due diversi tipi di editing. Il primo è spiegato dall'uso dei comandi illustrati nella Figura 7.4. Questo un approccio tipico alla necessità di provvedere un editing per un gruppo di informazioni che consiste di parti numerate (per esempio, una "pagina" di un testo che consiste di un numero n di linee, numerate da 1 a n per facilitare l'editing). Le funzioni necessarie sono quelle di aggiungere, modificare o eliminare una parte. L'aggiunta di una parte viene compiuta da una funzione facente parte di un gruppo di due ben distinte: e cioè inserire una parte o aggiungerne una alla fine. Questa separazione in due funzioni ha luogo perchè ci si trova davanti al problema di come specificare dove la parte inserita deve essere piazzata. Se il comando d'inserzione necessita della specifica delle voci che vanno inserite prima della loro inserzione, bisognerà trovare un metodo diverso che consenta di specificare "inserire alla fine". (Ve n'è chiaro il motivo)? D'altra parte, se il comando d'inserimento necessita di specifica delle voci dopo le quali la voce da inserire va piazzata, ci si troverebbe davanti ad un problema non appena si cercasse di inserire una voce prima di quella considerata numero uno. La soluzione più semplice rimane dunque quella che abbiamo usato nel programma del gioco degli Accoppiamenti.

L'altro tipo di editing che è stato usato nel gioco degli Accoppiamenti viene spiegato dal dialogo illustrato nella Figura 7.15. Se preferite redigere le vostre scelte e preferenze, il programma ve le mostrerà una alla volta, e mostrando via via anche che cosa è contenuto nell'archivio, dandovi così la possibilità di apportare modifiche se desiderate. Se la possibilità di avere una risposta del tipo "no answer" (nessuna risposta) è ammessa, allora il cambiamento apportato ad una risposta può anche comprendere il cancellarne una completamente e non sostituirla affatto. Questa cancellatura viene considerata come la funzione del input "D" che mostriamo nella Figura 7.15.

```
# Inizializzare l'array dei punteggi più alti

clearscores  SX = 0: RETURN                    #azzerare l'indice dell'array

1110  SX=0:RETURN

   La routine "clearscore" inizializza la coppia di array TS, TS azzerando l'indice SX
dell'"ultimo" elemento.
```

Figura 7.36 — Cancellare i punteggi più alti

Questo tipo di cancellatura si distingue da quella associata con il comando "D" che viene usato quando si redigono le domande. Infatti, in quel tipo di operazione si ha la possibilità di modificare la struttura dell'archivio delle domande oltre al suo contenuto. Mentre quando redigete le scelte e le preferenze, non avete nessun controllo sulla struttura; potete solo modificare il contenuto. Questo tipo di editing viene usato spesso con sistemi di ricerca degli errori interattivi — cioè con programmi che vi danno la possibilità di esaminare e di cambiare il contenuto della memoria del computer, poichè non vi è possibile alterare l'importo o l'ubicazione delle memorie.

Questo tipo di editing può essere usato soltanto per un archivio di proporzioni limitate, poichè bisogna esaminare le voci una per una prima di trovare quella che volete cambiare. Si sa bene che se il giocatore non può procedere velocemente, l'intero processo diventa tedioso. Questo avanzamento veloce viene reso possibile da un input ad un solo carattere nel gioco degli Accoppiamenti. È necessario soltanto premere il tasto "RETURN" per isolare una scelta o preferenza e poi continuare con quella seguente. Per cambiarla poi basta soltanto premere il numero della scelta (quindi un solo carattere perchè non è possibile avere più di nove scelte per domanda). E poi schiaccerete il tasto "Q" quando avete finito.

Algoritmi di inserimento e di cancellatura Nelle Figure 7.52 e 7.55 vediamo diversi esempi di cancellature. Nelle Figure 7.40 e 7.53 invece vediamo esempi di inserimenti. Algoritmi di inserimento e di cancellatura simili si possono usare ogni qualvolta ci si trovi davanti ad un array nel quale sia importante preservare l'ordine degli elementi. Quando si aggiunge una voce all'array, essa deve essere posta nell'indice cor-

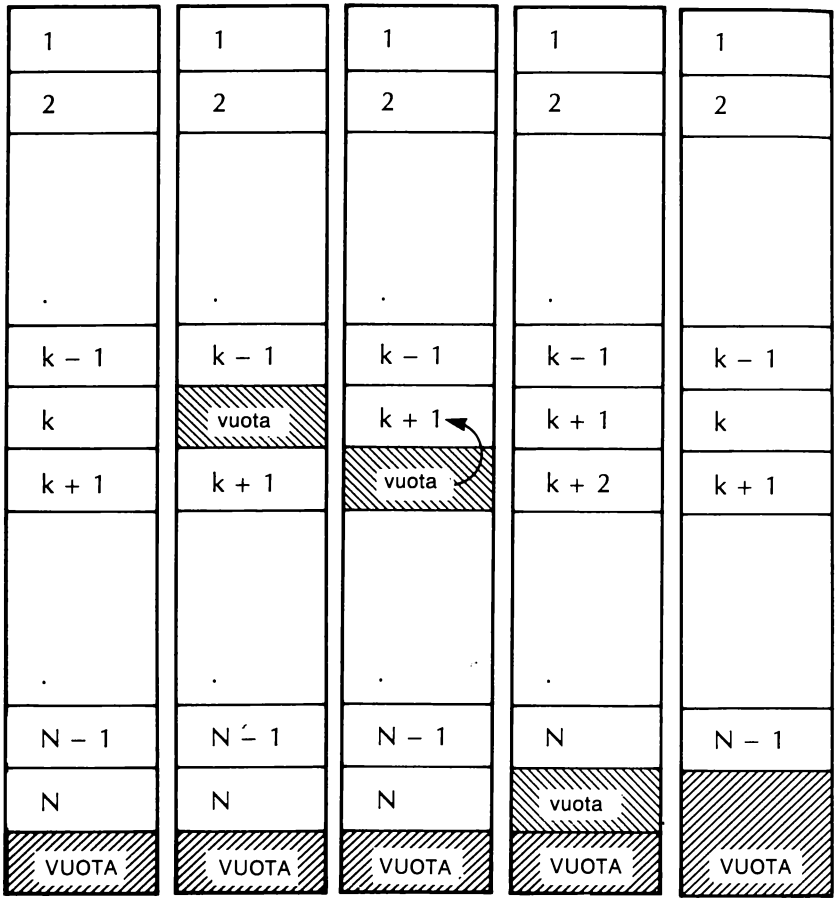
```
#Mostrare i punteggi più alti — gruppo GS — dagli arrays TS,TI

printscores   IF SX <= 0 THEN PRINT "NONE"          #se l'array è vuoto
                                                    # non stamparne
              else
                FOR SZ = 1 TO SX                    #controllare gli arrays
                  PRINT TS(SZ),NM$(TI(SZ),GS)       #stampare il punteggio
                                                    e il nome del giocatore
                  NEXT SZ
              RETURN

1120  IFSX<=0THENPRINT"NONE":RETURN
1130  FORSZ=1TOSX:PRINTTS(SZ),NM$(TI(SZ),GS):NEXTSZ:RETURN
```

La routine 'printscores" passa attraverso la coppia di arrays (TS,TI) stampando il punteggio da TS e il nome del giocatore usando l'indice da TI e il numero di gruppo GS.

Figura 7.37 — Mostrare sul video i nomi dei giocatori col punteggio più alto



Array prima della cancellatura

La voce k è cancellata, niente viene spostato

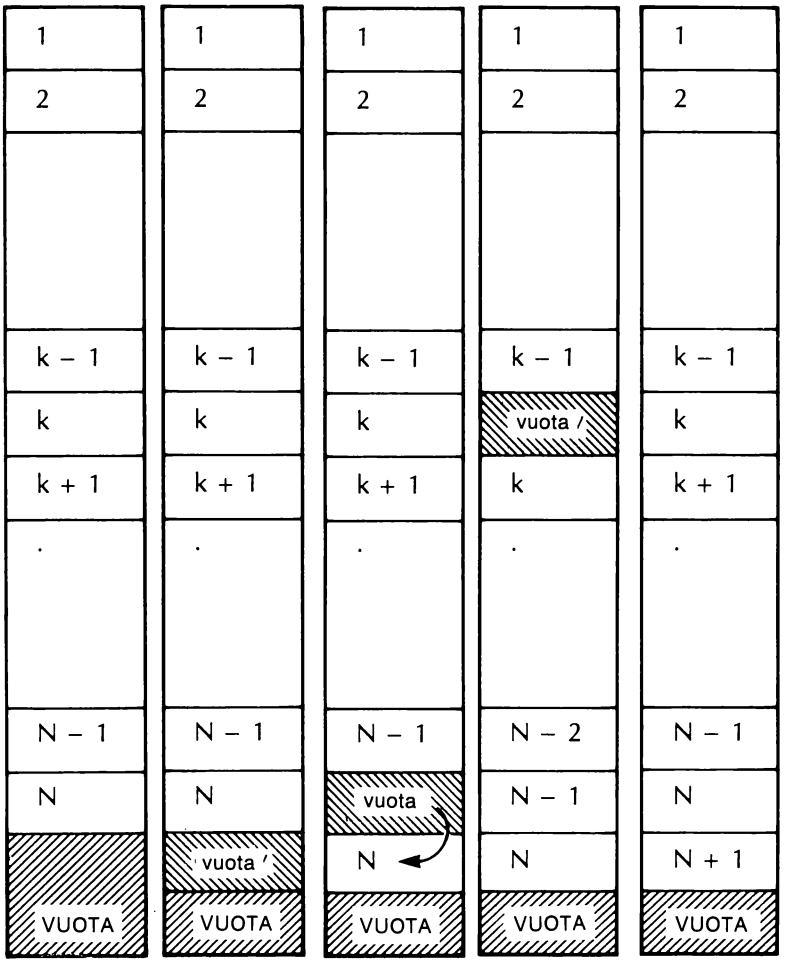
La voce K + 1 viene spostata verso l'alto la sua casella rimane vuota

Ogni voce è stata spostata verso l'alto, la casella della voce N è vuota

Le voci vengono rinumerate

La casella k è la prima a diventare vuota. La voce più vicina alla casella vuota (cioè la voce k + 1) viene spostata per prima.

Figura 7.38 — Cancellare la voce K



La casella N + 1 è la prima a diventare vuota. La voce più vicina alla casella vuota (cioè la voce N) viene mossa per prima.

Figura 7.39 — Inserimento prima della voce k

```
#Inserire il punteggio SC e l'indice del giocatore PX nella coppia di array (TS,TI)

savescore   IF SX = 0 THEN                          #se l'array è vuoto
              GOSUB addon: RETURN                   # inserire "alla fine"
            FOR SZ = 1 TO SZ
              IF SC > TS(SZ) THEN {                 #inserire prima di SZ
                IF SX < MT THEN
                  SX = SX + 1                       #c'è spazio per aggiun-
                                                    gerne un altro
                SE = SX - 1                         #l'indice più alto va
                                                    mosso
                IF SE >= SZ THEN
                  FOR SY = SE TO SZ STEP - 1        #muovere l'indice più
                                                    alto per primo
                    TS(SY + 1) = TS(SY)             #tutti si muovono verso
                                                    l'alto di uno
                    TI(SY + 1) = TI(SY)
                    NEXT SY
                TS(SZ) = SC                         #inserire la nuova voce
                TI(SZ) = PX
                RETURN
                }
              NEXT SZ                               #non inserire prima di
                                                    SZ
            IF SX < MT THEN                         #se c'è spazio
              GOSUB addon                           # inserire alla fine
            RETURN

1140  IFSX=0THENGOSUB1230:RETURN
1150  FORSZ=1TOSX:IFSC<=TS(SZ)THEN1210
1160  IFSX<MTTHENSX=SX+1
1170  SE=SX-1:IFSE<SZTHEN1200
1180  FORSY=SETOSZSTEP-1
1190  TS(SY + 1) = TS(SY):TI(SY + 1) = TI(SY):NEXTSY
1200  TS(SZ)=SC:TI(SZ)=PX:RETURN
1210  NEXTSZ:IFSX<MTTHENGOSUB1230
1220  RETURN
```

La routine "savescore" controlla l'array TS, paragonando il punteggio SC con gli elementi di TS. Poichè TS è in ordine decrescente, la prima posizione alla quale SC è maggiore dell'inserimento TS è l'indice al quale SC e PX saranno inseriti in TS e TI.

Figura 7.40 — Conservare i diversi punteggi

retto, e tutte le voci che seguono in quell'indice devono essere spostate. Quando si rimuove una voce, tutte quelle che la seguono devono essere spostate anch'esse così da non lasciare nessun spazio vuoto.

Esaminiamo cosa succede quando inseriamo o cancelliamo una voce dell'array. Per l'inserzione, ciascuna voce (se ovviamente ve ne sono) che segue quella inserita riceve l'indice con il numero più alto seguente; invece per la cancellatura, ciascuna voce che segue quella inserita riceve l'indice con il numero più basso seguente.

Quando si cancella una voce entro un certo array, si crea un'apertura nel mezzo, e la prima voce che si muoverà sarà quella la cui destinazione è questa apertura, (vedi Figura 7.38). Questo movimento crea di nuovo un'apertura, e la prossima voce che sarà mossa sarà quella che è immediatamente dopo l'apertura, e così via, finchè lo spazio vuoto non arrivi alla fine e sia così incorporato nello spazio vuoto finale.

```
addon       SX = SX + 1                          #aggiungere un altro alla lista
            TS(SX) = SC                          #aggiungerlo alla fine
            TI(SX) = PX
            RETURN

1230  SX=SX+1:TS(SX)=SC:TI(SX)=PX:RETURN
```

La routine "addon" è una subroutine breve che viene chiamata soltanto dalla routine "savescore".

Figura 7.40a — Una subroutine di "savescore"

```
#Mostrare la domanda (QX,GQ) e le sue scelte

showquestion PRINT Q$(QX,GQ)
             FOR CX = 1 TO NC(QX,GQ)
               PRINT CX; "";C$(CX,QX,GQ)
               NEXT CX
             RETURN

1240  PRINTQ$(QX,GQ)
1250  FORCX=1TONC(QX,GQ):PRINTCX; ""; C$(CX,QX,GQ):NEXTCX:RETURN
```

La routine "showquestion" mostra una domanda e le sue scelte. L'indice della domanda è QX, il numero del gruppo è GQ.

Figura 7.41 — Mostrare una domanda sul video

```
#Domandare il gruppo del giocatore

askgroup    PRINT                              #nuova linea
            FOR GX = 1 TO 2                    #stampare i nomi dei
                                               gruppi
              PRINT GX; " "; GN$(GX)
              NEXT GX
            PRINT "GROUP:";                    #domandare il gruppo
                                               del giocatore
            repeat
              GOSUB onech
              GP = VAL(X$)
                until (GP = 1 OR GP = 2)
            PRINT X$;                          #ripetere la risposta del
                                               giocatore
            RETURN

1260  PRINT:FORGX=1TO2:PRINTGX;" ";GN$(GX):NEXTGX:
      PRINT"GROUP: ";
1270  GOSUB2180:GP=VAL(X$):IFNOT(GP = 1ORGP = 2)THEN1270
1280  PRINTX$;:RETURN
```

La routine "askgroup" mostra i nomi dei due gruppi, domanda il gruppo del giocatore e accetta una risposta che sia 1 o 2.

Figura 7.42 — A che gruppo appartieni?

```
#Calcolare il numero GC dell'"altro" gruppo rispetto a GP

othergroup  IF GP = 1 THEN                     #se sei 1,
              GC = 2                           #l'altro è 2
            else                               #e viceversa
              GC = 1
            RETURN

1290  IFGP=1THENGC=2:RETURN
1300  GC=1:RETURN
```

La routine "othergroup" prende il numero del gruppo GP e calcola il numero dell'altro gruppo GC. Notate che questa relazione abbastanza semplice non può essere rappresentata facilmente da una funzione BASIC. (Potreste "imbrogliare": DEF FNOT(X) = 3 − X).

Figura 7.43 — A che gruppo non appartieni?

```
#Controllare l'identità del giocatore

identity    GOSUB askgroup                          #ottenere il gruppo del
                                                    giocatore
            GOSUB clearscreen
            IF NP(GP) = 0 THEN{
              PRINT "NO ";GN$(GP);" ON FILE."
              XX = 0: RETURN                        #segnale "fail" (errato)
              }                                     ritornare
            XX = 1                                  #segnale "success"
                                                    (corretto) ritornare
            FOR PX 1 TO NP(GP) STEP screenload
              GOSUB clearscreen

              PT=min (PX + screenload               #indice più alto per que-
              - 1,NP(GP)                            sto carico
              FOR PY = PX TO PT                     #mostrare dei
                PRINT PY; " "; NM$(PY,GP)           # numeri e nomi
                NEXT PY
              repeat{
                PRINT "NUMBER: ";:GOSUB stringin    #domandare il numero
                                                    del giocatore
                PP = VAL(SS$)
                IF 1 < = PP < = NP(GP) THEN         #numero valido per que-
                                                    sto gruppo
                  RETURN
                else IF PP = 0 THEN                 #non è un numero
                  break
                }
              IF X$ = "Q" THEN                      #Q abortisce
                {XX = 0: RETURN}                    #segnale "fail" ritornare
              NEXT PX
            PRINT "THAT'S ALL THE";GN$(GP)          #tutto il carico mostrato
                                                    sullo schermo
            XX = 0: RETURN                          #segnale "fail" ritornare
```

La routine "identity" domanda il gruppo del giocatore, poi mostra i nomi ed i numeri di tutti i membri di quel gruppo, uno alla volta. Dopo ciascuno il giocatore deve inserire un numero. Una risposta Q abortisce l'intero processo. Se si dovesse dare una risposta non numerica, si vedrà il nome seguente.

Figura 7.44 — Qual'è il tuo numero?

Quando bisogna inserire una voce nuova (vedi Figura 7.39) non esiste uno spazio vuoto dove questa possa essere inserita, a meno che non la si aggiunga alla fine. Verrà così identificata la prima apertura alla fine dell'array (se ovviamente ce n'è una) e poi la voce più vicina a questa apertura (per esempio la parte finale) vi viene inserita. Questo causa il movimento dell'apertura nello spazio dove prima c'era la voce finale, e la penultima si sposta in questo spazio vuoto. In questo modo l'apertura viene spostata dove vogliamo inserire la nuova voce.

Una volta che questa sia stata inserita, entra in funzione la rinumerazione automatica illustrata nelle Figure 7.38 e 7.39, poichè gli indici degli array appartengono alle caselle e non alle voci.

Per ricapitolare, la regola da ricordare durante l'inserimento e la cancellatura è: la voce vicina alla casella vuota si muove per prima.

Progettazione e funzioni di controllo dell'archivio Questo è un argomento ovviamente molto vasto, e il programma del gioco degli Accoppiamenti ne illustra soltanto una parte. In generale, un archivio non è altro che una raccolta di voci chiamate documenti, ognuna delle quali contiene un carattere particolare di ciascun argomento tenuto in archivio. Per esempio, un archivio di nomi di clienti, numeri di telefono e numeri di scarpa conterrà

```
PAUL BUNYAN          (509) BLUE 0X7-0048  17EEEE
```

Questi tipo di archivi vengono di solito conservati su dischi o nastri.

Nel programma del gioco degli Accoppiamenti, gli "archivi" sono gli arrays di do-

```
1310  GOSUB1260:GOSUB2200:IFNP(GP) >0THEN1330
1320  PRINT"NO ";GN$(GP);" ON FILE.":XX=0:RETURN
1330  XX=1:FORPX=1TONP(GP)STEP10:GOSUB2200
1340  PT=PX+9:IFPT >NP(GP)THENPT=NP(GP)
1350  FORPY=PXTOPT:PRINTPY;" ";NM$(PY,GP):NEXTPY
1360  PRINT"NUMBER: ";:GOSUB610:PP=VAL(SS$)
1370  IF1 <=PPANDPP <=NP(GP)THENRETURN
1380  IFPP< >0THEN1360
1390  IFX$="Q"THENXX=0:RETURN
1400  NEXTPX
1410  PRINT"THAT'S ALL THE ";GN$(GP):XX=0:RETURN
```

Figura 7.44a — BASIC per l'identità del giocatore

mande e di informazioni riguardanti il giocatore. Per esempio, l'"archivio" riguardante il giocatore 1 del gruppo 1 sarà composto da:

```
NM$ (1 , 1)
A (QX, 1 , 1)          QX = 1,...,NQ (1)
W (QX, 1 , 1)          QX = 1,...,NQ (2)
```

```
#Preparare la domanda ed i pesi numerici

create      repeat{
              GOSUB clearscreen                          #iniziare con lo schermo vuoto
              PRINT "::";:GOSUB onech: PRINT X$          #ottenere e ripetere il comando
              IF case
                X$ = "I" THEN GOSUB newgame              #iniziare da zero
                X$ = "N" THEN GOSUB editnames            #redigere i nomi dei gruppi
                X$ = "1" THEN GOSUB editone              #redigere le domande per il gruppo 1
                X$ = "2" THEN GOSUB edittwo              #redigere le domande per il gruppo 2
                X$ = "L" THEN GOSUB loadin               #prendere i dati da cassette o dischi
                X$ = "S" THEN GOSUB writeout             #registrare in cassette o dischi
                X$ = "Q" THEN break                      #lasciare che la partita inizi
              }
            RETURN

1420  GOSUB2200:PRINT"::";:GOSUB2180:PRINTX$
1430  IFX$="I"THENGOSUB1510:GOTO1420
1440  IFX$="N"THENGOSUB1700:GOTO1420
1450  IFX$="1"THENGOSUB1740:GOTO1420
1460  IFX$="2"THENGOSUB1750:GOTO1420
1470  IFX$="L"THENGOSUB2210:GOTO1420
1480  IFX$="S"THENGOSUB2220:GOTO1420
1490  IFX$ <> "Q"THEN1420
1500  RETURN
```

La routine "create" contiene la scelta del comando per la creazione e la redazione delle domande.

Figura 7.45 — Scegliere il comando "Setup" (preparare)

```
#Accettare i dati per la partita

newgame    FOR GX = 1 TO 2
             PRINT: PRINT "GROUP"; GX; ":"
             repeat                                  #ottenere il nome del
                                                     gruppo
               GOSUB stringin
               until (SS$ < > "")
             GN$(GX) = SS$
             NQ(GX) = 0                              #inizializzare il numero
                                                     della domanda
             while (NQ(GX) < MQ) {                   #chiedere MQ domande
               GOSUB clearscreen
               QX = NQ(GX) + 1
               GOSUB inquestion                      #ottenere una doman-
                                                     da, scrivere le scelte
               IF X$ = "Q" THEN                      #non più domande
                 break
               else
                 NQ(GX) = QX                         #aggiornare il numero
                                                     delle domande
                 }
             NEXT GX
           RETURN


1510  FORGX=1TO2
1520* PRINT:PRINT"GROUP";GX": ";
1530  GOSUB610:IFSS$=""THEN1530
1540  GN$(GX)=SS$:NQ(GX)=0
1550  IFNQ(GX)>=MQTHEN1590
1560  GOSUB2200:QX=NQ(GX)+1:GOSUB1600
1570  IFX$="Q"THEN1590
1580  NQ(GX)=QX:GOTO1550
1590  NEXTGX:RETURN
```

La routine "newgame" passa attraverso le voci necessarie per preparare la partita e domanda al giocatore di fornirgliele.

---

* La versione per l'Apple di questa linea differisce solamente nella spaziatura nelle costanti di stringa.

Figura 7.46 — Preparare una nuova partita

```
#Ottenere una domanda QX per il gruppo GX

inquestion   PRINT: PRINT "Q"; QX; "FOR"; GN$(GX)      #richiedere la domanda
             repeat{
               GOSUB stringin                          #accettare la stringa
               IF SS$ = "Q" THEN{
                 X$ = "Q"                              #Q = niente più domande

                 RETURN
                 }
               else IF SS$ < > "" THEN{                #richiedere una stringa non nulla

                 Q$(QX,GX) = SS$
                 break
                 }
               }
             NC(QX,GX) = 0                             #inizializzare il numero delle scelte
             while (NC(QX,GX) < MC){                   #permettere fino a MC scelte

               CX = NC(QX,GX) + 1
               PRINT CX; "";                           #domandare la scelta seguente

               repeat{
                 GOSUB stringin                        #accettare la stringa
                 IF SS$ = "Q" THEN
                     break: break                      #Q = niente più scelte
                 else IF SS$ < > "" THEN{              #richiedere una stringa non nulla

                   C$(CX,QX,GX) = SS$
                   break
                 } }
               NC(QX,GX) = CX                          #aggiornare il numero delle scelte
               }

             PRINT: PRINT "WEIGHT:";
             GOSUB stringin
             WT(QX,GX) = VAL(SS$)
             RETURN
```

La routine "inquestion" domanda e accetta una domanda e le sue scelte. Le funzioni di editing non vengono compiute in questa routine.

Figura 7.47 — Ottenere una domanda

La stringa NM$ (1 , 1) corrisponde al nome del giocatore, il numero A (QX, 1, 1) corrisponde alla risposta del giocatore alla domanda numero QX del gruppo 1, e il numero W (QX, 1 , 1) rappresenta la forma codificata delle preferenze del giocatore per le risposte possibili alla domanda numero QX del gruppo 2. Le espressioni QX = 1,...,NQ (1) e QX = 1,...,NQ (2) indicano che esiste una risposta per ciascuna domanda facente parte del gruppo 1, e una preferenza codificata per ciascuna domanda del gruppo 2. NQ (1) rappresenta il numero di domande facenti parte del gruppo 1, NQ (2) rappresenta il numero di domande per il gruppo 2.

Uno degli obbiettivi principali della programmazione di routines di controllo degli archivi è il mantenere l'integrità delle informazioni negli archivi stessi. Cioè le routine di controllo degli archivi devono far sì che l'informazione giusta vada inserita e mantenuta al posto giusto, e che non vengano inserite informazioni errate. Naturalmente, le routines non hanno modo di essere certe che il numero di scarpa di PAUL BUNYAN sia effettivamente 17EEEE, ma possono almeno far sì che la persona che inserisce questi dati ne sia certa prima di inserirli. Questo è lo scopo dello spazio usato nella redazione delle informazioni nel programma del gioco degli Accoppiamenti.

La routine "editplayer" illustrata nella Figura 7.17 mostra come questo spazio viene usato: la subroutine "loadwork" porta la "documentazione" relativa ad un certo giocatore (per esempio, le voci summenzionate prese dagli arrays NM$, A e W) e la mette in questo spazio, la subroutine "change" dà la possibilità a; giocatore di cambiare parte della documentazione che si trova in questo spazio (il contenuto originale degli arrays non può essere cambiata durante questa fase di editing) ed infine, la subroutine "update" inserisce la documentazione già redatta al posto del contenuto

```
1600* PRINT:PRINT"Q";QX;"FOR ";GN$(GX)
1610  GOSUB610:IFSS$="Q"THENX$="Q":RETURN
1620  IFSS$=""THEN1610
1630  Q$(QX,GX)=SS$:NC(QX,GX)=0
1640  IFNC(QX,GX)>=MCTHEN1690
1650  CX=NC(QX,GX)+1:PRINTCX;"";
1660  GOSUB610:IFSS$="Q"THEN1690
1670  IFSS$=""THEN1660
1680  C$(CX,QX,GX)=SS$:NC(QX,GX)=CX:GOTO1640
1690  PRINT:PRINT"WEIGHT: ";:GOSUB610:WT(QX,GX) = VAL(SS$): RETURN
```

* La versione per l'Apple di questa linea differisce soltanto nella spaziatura delle costanti di stringa.

Figura 7.47a — BASIC per "inquestion"

dell'array. La routine "update" viene chiamata soltanto quando il giocatore ha risposto con un "N" alla domanda "MORE?". Un "Q" come risposta mette fine e cancella l'intero editing, mentre qualsiasi altra risposta consente al giocatore di passare oltre la fase dell'editing.

Siate ben certi di aver capito fino in fondo il processo di editing che abbiamo appena discusso prima di tentare di applicarlo. Una buona routine di editing deve permettervi di potervi fermare ed abbandonare un processo di redazione già iniziato senza però danneggiare le informazioni esistenti. Effettuare dei cambiamenti su di una copia già funzionante è la tecnica principale usata per questo scopo. (Siete in grado di pensare ad un altro modo per raggiungere lo stesso risultato?)

Via via che andate avanti a studiare il programma del gioco degli Accoppiamenti, dovreste fare attenzione a come viene effettuata la redazione delle domande. Chiedetevi in che cosa differisce dalle tecniche di cui abbiamo appena parlato, e perchè.

Risparmio di spazio necessario Le tecniche di programmazione di cui parleremo adesso sono quelle che riguardano il risparmio di spazio necessario. Questo è un problema universale; non c'è mai memoria a sufficienza. Sia che si tratti di un piccolissi-

```
#Redigere i nomi dei gruppi

editnames   FOR GX = 1 TO 2
              PRINT GX; ""; GN$(GX)          #mostrare il nome del
                                             gruppo sullo schermo
              GOSUB stringin                 #accettare cambiamen-
                                             to
              IF SS$ < > "" THEN             #RETURN = "nessun
                                             cambiamento"
                GN$(GX) = SS$
              NEXT GX
            RETURN

1700  FORGX=1TO2:PRINTGX;"";GN$(GX)
1710  GOSUB610:IFSS$ < > ""THENGN$(GX)=SS$
1720  NEXTGX:RETURN
```

La routine "editnames" mostra il nome del gruppo per ciascuno dei due gruppi, e permette al giocatore di inserire un cambiamento. Un segnale "RETURN" significa "nessun cambiamento" altrimenti il nome viene sostituito dalla stringa scritta.

Figura 7.48 — Editing dei nomi dei gruppi

mo microprocessore, o di un computer ad elaboratore centrale, prima o poi il vostro programma finirà la memoria. Per parafrasare un principio famoso diremo che:

Il programma si espande per riempire la memoria disponibile.

```
#Routine di editing delle domande

editone     GX = 1: GOSUB edit
            RETURN

edittwo     GX = 2: GOSUB edit
            RETURN

edit        repeat{
              GOSUB clearscreen                    #iniziare con lo schermo vuoto
              PRINT " *";                          #suggerire per l'input
              GOSUB onech: PRINT X$                #ottenere il comando
              IF case
                X$ = "E" THEN GOSUB modques        #redigere una domanda
                X$ = "D" THEN GOSUB delques        #cancellare una domanda
                X$ = "A" THEN GOSUB addques        #aggiungere una domanda (alla fine)
                X$ = "B" THEN GOSUB insertques     #inserire una domanda
                X$ = "Q" THEN break                #non più editing
              }
            RETURN

1740  GX=1:GOSUB1760:RETURN
1750  GX=2:GOSUB1760:RETURN
1760  GOSUB2200:PRINT"*";:GOSUB2180:PRINTX$:IFX$=
      "Q"THENRETURN
1770  IFX$="E"THENGOSUB1990:GOTO1760
1780  IFX$="D"THENGOSUB1850:GOTO1760
1790  IFX$="A"THENGOSUB1820:GOTO1760
1800  IFX$="B"THENGOSUB1910:GOTO1760
1810  GOTO1760
```

Questo è il codice comune per maneggiare i due comandi di editing "1" e "2". In entrambi i casi, un * viene mostrato sullo schermo come suggerimento. Poi i comandi vengono accettati in modo da poter modificare, cancellare, aggiungere o inserire una domanda. Queste sono le funzioni di editing fondamentali che sono necessarie in ogni situazione di redazione di testo.

Figura 7.49 — Scegliere la funzione di editing delle domande

Inoltre, (guardando il problema sotto una luce più positiva) se state scrivendo un programma che diventerà un prodotto, o parte di un prodotto, un programma più piccolo risulterà in un prodotto meno costoso con dunque un potenziale di vendita maggiore. Per esempio un programma in BASIC che usa 8K sul computer XYZ sarà acquistato da un numero maggiore di persone che già posseggono un computer XYZ di un programma con le stesse caratteristiche che però usa 48K (a meno che, naturalmente, tutta la gamma dei computer XYZ ha almeno 48K di memoria).

Esistono due modi per risparmiare spazio necessario in piccoli computers:

- "Compressione" del programma BASIC.
- Programmazione che riduca lo spazio necessario e algoritmi per l'archiviazione dei dati.

Le possibilità di "comprimere" il vostro programma BASIC dipendono da come funziona il vostro sistema. Per esempio, sul TRS-80, tutti gli spazi che inserite tra il

```
#Aggiungere una domanda al gruppo GX

addques     IF NQ(GX) >= MQ THEN {
              PRINT "NO ROOM"
              RETURN
              }
            QX = NQ(GX) + 1                    #indice della nuova domanda

            GOSUB inquestion
            IF X$ <> "Q" THEN                  #Q abortisce il processo
              NQ(GX) = QX                      #aggiornare il numero delle domande

            RETURN

1820  IFNQ(GX) >=MQTHENPRINT"NO ROOM":RETURN
1830  QX=NQ(GX) +1:GOSUB1600:IFX$ <>"Q"THENNQ(GX) =QX
1840  RETURN
```

Questa routine aggiunge una domanda alla fine della lista per il gruppo GX. L'errore fatto nella routine "storenew" (Figura 7.22) non viene ripetuto. Quindi il processo viene terminato immediatamente se non c'è più spazio.

Notare l'uso della variabile QX e dell'inserimento dell'array NQ(GX). QX viene fissato immediatamente al nuovo numero, ma se il processo di inserimento della domanda viene abortito ("inquestion" ritorna con X$ = "Q"), allora NQ(GX) non viene alterato.

Figura 7.50 — Aggiungere una domanda

numero della linea e la fine di una dichiarazione in BASIC diventano parte del programma; ciascuno spazio cioè (a parte il primo) vi costerà un byte di memoria. (N.d.T. Un byte corrisponde ad una sequenza di cifre binarie considerate come un'unità e di solito più breve di una parola).

Sul Pet, gli spazi tra il numero della linea e il primo carattere scritto della dichiarazione vengono ignorati (Il Pet fornisce solo uno spazio negli elenchi), ma tutti gli altri spazi diventano parte integrante del programma. Sull'Apple, tutti gli spazi che voi scrivete vengono ignorati: infatti l'Apple fornisce delle spaziature e degli elenchi propri. (In tutti questi sistemi, tutti gli spazi che esistono entro una costante a stringa — per esempio "HOW ARE YOU" — vi costa un byte, anche sul TRS-80 che ha codici a singoli caratteri (ASCII 192 a 255) fino a 63 caratteri vuoti consecutivi).

Il programma del gioco degli Accoppiamenti è stato sviluppato su un Pet a 8K, e i comandi in BASIC che compaiono nella Figura 7.16 e nelle successive fino alla 7.59 non hanno che pochissimi spazi non essenziali. Naturalmente questo diminuisce la loro chiarezza alla lettura, ma certamente questo non è importante, perchè i comandi in BASIC non sono stati concepiti per essere letti da occhi umani. Soltanto le descrizioni in Free BASIC sono state ideate con questo scopo.

Per la stessa ragione, la mancanza di dichiarazioni REM nei comandi in BASIC non costituisce un problema, perchè le forme BASIC e quella Free BASIC vengono appaiate in moduli piccoli nelle figure che fanno parte del programma del gioco degli Accoppiamenti. In realtà, poichè dovete tradurre dal Free BASIC in BASIC a mano e non avrete un programma a farlo per voi, sarebbe utile iniziare ciascuna subroutine con una dichiarazione REM che contenga il nome della subroutine. Così, se potete mostrare sullo schermo tutte le linee che contengono una parte specifica del programma, potrete anche generare una "tavola di simboli" che metta in relazione i numeri di linee ai nomi delle subroutines. Per esempio, su di un Pet con il "Programmer Toolkit" di Nestar, potrete inserire il comando:

```
#Cancellare una domanda dal gruppo GX

delques     GOSUB qnum                      #ottenere un numero di
                                            domanda QQ
            IF QQ < > 0 THEN                #QQ = 0 abortisce
              GOSUB qout
            RETURN

1850  GOSUB2160:IFQQ < >0THENGOSUB1870
1860  RETURN
```

Questa routine cancella una domanda dal gruppo GX. La meccanica della cancellazione viene effettuata nella routine "qout".

Figura 7.51 — Cancellare una domanda

FIND REM

per ottenere sullo schermo

```
210 REM EDITPLAYER
260 REM NEWPLAYER
300 REM CLEARWORK
```

```
#Eliminare la domanda QQ dal gruppo GX

qout        NQ(GX) = NQ(GX) - 1                    #una domanda in meno
            IF QQ > NQ(GX) THEN                    #niente altro da fare se
              RETURN                               # si cancella la domanda finale

            FOR QX = QQ TO NQ(GX)                  #altrimenti spostare ogni
              Q$(QX,GX) = Q$(QX + 1,GX)            # domanda verso l'alto di una casella
              IF NC(QX + 1, GX) > 0 THEN           #spostare anche le scelte, se ci sono
                FOR CX = 1 TO NC(QX + 1,GX)
                  C$(CX,QX,GX) = C$(CX,QX + 1,GX)
                  NEXT CX
              NC(QX,GX) = NC(QX + 1,GX)            #spostare il numero delle scelte
              WT(QX,GX) = WT(QX + 1,GX)            #spostare il peso numerico
              NEXT QX
            RETURN

1870  NQ(GX)=NQ(GX)-1:IFQQ>NQ(GX)THENRETURN
1880  FORQX=QQTONQ(GX):Q$(QX,GX)=Q$(QX+1,GX):IFNC(QX+
      1,GX) < = 0THEN1900
1890  FORCX=1TONC(QX + 1,GX):C$(CX,QX,GX)=
      C$(CX,QX+1,GX):NEXTCX
1900  NC(QX,GX)=NC(QX+1,GX):WT(QX,GX)=WT(QX+1,GX):
      NEXTQX:RETURN
```

La routine "qout" implementa la meccanica di eliminazione della domanda QQ dal gruppo GX. La stringa della domanda, le scelte, il numero delle scelte e il peso numerico devono essere spostati, perchè tutte le domande vengono spostate verso l'alto di una casella.

Figura 7.52 — Meccanica di eliminazione di una domanda

```
#Inserire una domanda nel gruppo GX
insertques   IF NQ(GX) >=MQ THEN {                  #se non c'è spazio
               PRINT "NO ROOM"                      # informare il giocatore
                                                    e uscire
               RETURN
             }
             PRINT "AHEAD OF";                      #trovare dove si può in-
                                                    serire la domanda
             GOSUB qnum
             IF QQ = 0 THEN RETURN                  #QQ = 0 se il giocatore
                                                    abortisce il processo
             FOR QX = NQ(GX) TO QQ STEP -1          #muovere per prima
                                                    l'ultima domanda
               Q$(QX + 1,GX) = Q$(QX,GX)
               IF NC(QX,GX) >0 THEN                 #muovere le scelte, se
                                                    ci sono
                 FOR CX = 1 TO NC(QX,GX)
                   C$(CX,QX + 1,GX) = C$(CX,QX,GX)
                   NEXT CX
               NC(QX + 1,GX) = NC(QX,GX)            #muovere il numero del-
                                                    le scelte
               WT(QX + 1,GX) = WT(QX,GX)            #muovere il peso
               NEXT QX
             NQ(GX) = NQ(GX) + 1                    #aggiornare il numero
                                                    delle domande
             QX = QQ: GOSUB inquestion              #ottenere la domanda
             IF X$ = "Q" THEN                       #Q significa abortire
               GOSUB qout                           #ridare lo spazio
             RETURN
```

```
1910 IFNQ(GX) >= MQTHENPRINT"NO ROOM":RETURN
1920 PRINT"AHEAD OF";:GOSUB2160:IFQQ=0THENRETURN
1930 FORQX=NQ(GX)TOQQSTEP-1:Q$(QX + 1,GX)=Q$(QX,GX)
1940 IFNC(QX,GX) <= 0THEN1960
1950 FORCX=1TONC(QX,GX):C$(CX,QX + 1,GX)=
     C$(CX,QX,GX):NEXTCX
1960 NC(QX + 1,GX)=NC(QX,GX):WT(QX + 1,GX)=WT(QX,GX):NEXTQX
1970 NQ(GX)=NQ(GX) + 1:QX=QQ:GOSUB1600:IFX$=
     "Q"THENGOSUB1870
1980 RETURN
```

La routine "insertiques" inserisce una domanda nel gruppo GX. Al giocatore viene chiesto di specificare il numero della domanda prima della quale verrà inserita quella nuova. Spostando tutte le domande verso l'alto, partendo dal numero scelto, si libera così una casella per la domanda da inserire. Per l'inserimento è essenziale che tutte le domande vengano spostate nella nuova casella prima che lo spazio sia libero per inserire la domanda proveniente dal basso.

Figura 7.53 — Inserire una domanda

Se voleste inserire questo output su di una cassetta, potreste poi farlo passare su di un programma che è diventato:

```
EDITPLAYER    210
NEWPLAYER     260
CLEARWORK     300
```

```
#Redigere una domanda nel gruppo GX
modques     GOSUB qnum                          #numero della domanda
            IF QQ = 0 THEN
              RETURN
            PRINT Q$(QQ,GX)                     #mostrare la domanda
                                                sullo schermo
            GOSUB stringin                      #accettare il cambia-
                                                mento
            IF SS$ = "Q" THEN                   #Q = nessun cambia-
                                                mento, smettere
              RETURN
            else IF SS$ < > "" THEN             #RETURN = nessun
                                                cambiamento
              Q$(QQ,GX) = SS$                   #conservare la nuova
                                                stringa
              GOSUB choices                     #redigere le scelte
            PRINT: PRINT "WEIGHT:"; WT(QQ,GX);":"; #mostrare il peso nume-
                                                rico sul video
            GOSUB stringin                      #accettare il cambia-
                                                mento
            IF VAL(SS$) < > 0 THEN              #non è numerico, o è
                                                zero
              WT(QQ,GX) = VAL(SS$)              #significa nessun cam-
                                                biamento
            RETURN
1990  GOSUB2160:IFQQ=0THENRETURN
2000  PRINTQ$(QQ,GX):GOSUB610:IFSS$="Q"THENRETURN
2010  IFSS$ < > ""THENQ$(QQ,GX)=SS$
2020  GOSUB2050:PRINT:PRINT"WEIGHT:";WT(QQ,GX)":";:GOSUB610
2030  IFVAL(SS$) < >0THENWT(QQ,GX)=VAL(SS$)
2040  RETURN
```

La routine "modques" fa sì che il giocatore possa redigere una domanda nel gruppo GX. L'editing della stringa della domanda viene effettuato in questa fase. L'editing delle scelte, invece, avviene nella routine "choices".

Figura 7.54 — La redazione di una domanda

```
#Redigere le scelte per la domanda (QQ,GX)

choices     CX = 1                                  #inizializzare l'indice
                                                    delle scelte
            while (CX < = NC(QQ,GX) ) {             #per ogni scelta,
              PRINT CX; " "; C$(CX,QQ,GX)           # mostrare il numero
                                                    ed il testo
              GOSUB stringin                        # e accettare cambia-
                                                    menti
              IF SS$ = "Q" THEN RETURN              # Q = smettere
              else IF SS$ = "D" THEN {              #D = cancellare la
                                                    scelta
                NC(QQ,GX) = NC(QQ,GX) -1            #una scelta in meno
                IF CX < = NC(QQ,GX) THEN            #se l'ultima scelta non
                                                    viene eliminata
                  FOR CY = CX TO NC(QQ,GX)          # muovere le scelte
                                                    che seguono verso l'alto
                    C$(CY,QQ,GX) = C$(CY + 1,QQ,GX)
                  NEXT CY
                }
              else {                                #non Q o D
                IF SS$ < >"" THEN                   #se non è una stringa
                                                    nulla
                  C$(CX,QQ,GX) = SS$                # sostituire la scelta
                                                    con SS$
                CX = CX + 1
                }
              }
            while (NC(QQ,GX) < MC){                 #mentre c'è ancora
                                                    spazio
              CX = NC(QQ,GX) +                      # fissare l'indice della
                                                    scelta e
              PRINT CX; "";:GOSUB stringin          # chiederla
              IF SS$ = "" or "Q" THEN RETURN        #RETURN o Q fa termi-
                                                    nare l'intero processo
              else {
                C$(CX,QQ,GX) = SS$                  #fissare la scelta a SS$
                NC(QQ,GX) = CX                      #contarla
              }
            }
            RETURN
```

La routine "choices" permette al giocatore di redigere il gruppo di scelte per la domanda i cui indici sono (QQ,GX). Prima di tutto le scelte esistenti vengono esaminate, e il giocatore può redigerne o elimerne. (D = cancellare, RETURN = nessun cambiamento). Poi le nuove scelte possono essere aggiunte alla fine. Nella fase di aggiunta, sia un Q che un RETURN possono far terminare l'intero processo.

Nella prima parte del programma, CX viene controllato in modo evidente. Un loop FOR..NEXT non potrebbe essere usato, perchè la cancellatura delle scelte potrebbe alterare il limite superiore NC(QQ,GX).

Figura 7.55 — La redazione delle scelte

Infine, se inseriste questo output su di una cassetta, poi usandolo come input per un programma di classificazione, potreste ottenere una lista in ordine alfabetico abbastanza utile:

```
ASKGROUP    1260
ASKOK        510
ASKWISH      770
```

Vi sono diverse tecniche che possono essere usate per "comprimere" il vostro programma in BASIC. Certamente il vostro manuale in BASIC del sistema che usate conterrà dei suggerimenti al riguardo. Un altro approccio di cui parleremo qui è l'uso della programmazione che riduca lo spazio necessario e gli algoritmi per l'archiviazione dei dati.

Spesso non esiste un modo "ottimale" per compiere determinate operazioni o per archiviare delle informazioni. Normalmente vi sono situazioni in cui un certo approccio fa sì che si abbia un programma più corto che però impiega più tempo per eseguire un'operazione, mentre un altro può usare un programma più vasto che però sia più veloce nel compiere la stessa operazione. Non esiste una formula che renda più facile prendere una decisione di questo tipo. Infatti, se non si porta a termine il progetto in tutti e due i diversi modi, non si può calcolare esattamente quali saranno le necessità di spazio e di tempo.

Nel programma del gioco degli Accoppiamenti, questo si verifica nella codifica delle preferenze del giocatore. Per ciascuna domanda dell'"altro" gruppo, un gioca-

```
2050  CX=1
2060  IFCX>NC(QQ,GX)THEN2130
2070  PRINTCX;" ";C$(CX,QQ,GX):GOSUB610:IFSS$=
      "Q"THENRETURN
2080  IFSS$ <: "D"THEN2110
2090  NC(QQ,GX)=NC(QQ,GX)-1:IFCX > NC(QQ,GX)THEN2060
2100  FORCY=CXTONC(QQ,GX):C$(CY,QQ,GX)=
      C$(CY + 1,QQ,GX):NEXTCY:GOTO2060
2110  IFSS$ <> ""THENC$(CX,QQ,GX)=SS$
2120  CX=CX + 1:GOTO2060
2130  IFNC(QQ,GX)=>MCTHENRETURN
2140  CX=NC(QQ,GX) + 1:PRINTCX;" ";:GOSUB610:IFSS$=
      ""ORSS$="Q"THENRETURN
2150  C$(CX,QQ,GX)=SS$:NC(QQ,GX)=CX:GOTO2130
```

Figura 7.55a — BASIC per "Choices"

tore può assegnare una scelta a ciascuno dei valori delle preferenze NF. (NF è una variabile il cui valore viene determinato da una dichiarazione DATA nella routine "init" nelle Figure 7.11, 7.13 e 7.15 questo valore è 4). Il modo più facile per archiviare queste preferenze sarebbe di tenerle in un array le cui dimensioni sono NF, MQ, MP, GM. Cioè, la scelta sarebbe indicizzata per gruppo, per numero del giocatore, per numero della domanda e per numero di valore della preferenza. Assumendo un valore per NF=4, per MQ=6, per MP=12, per GM=2, questo array conterrebbe 4 x 6 x 12 x 2 = 576 voci.

Queste voci occuperebbero 6.838 bytes di spazio di archivio in un Apple o in un Pet, e 5.474 bytes in un TRS-80. L'approccio usato nel programma del giorno degli Accoppiamenti è quello di codificare tutte le preferenze per una certa domanda in un numero (vedremo in seguito come) cosicchè le dimensioni dell'array sono MQ, MP, GM. Usando i valori di cui sopra, si produce un array di 144 voci, che occupa 1.376 bytes su di un Apple o di un Pet, e 1.104 bytes su di un TRS—80 — dunque un risparmio di circa l'80% in entrambi i casi.

Le preferenze vengono raggruppate in una sola parola trattandole come numeri con la base MC + 1, dove MC è la variabile che contiene il numero massimo delle scelte che sono permesse. Questo valore è fissato in una dichiarazione DATA e non può superare il 9.

Probabilmente saprete già che se b è un numero intero, allora qualsiasi altro numero intero N può essere scritto in un solo modo

$$N = a_0 + a_1 \times b + a_2 \times b^2 + \ldots + a_k \times b^k$$

```
#Ottenere un numero di domanda QQ per il gruppo GX; QQ = 0 se non è valido

qnum        PRINT "QUESTION NUMBER:";
            GOSUB stringin
            QQ = VAL(SS$)
            IF QQ < 0 OR QQ > NQ(GX) THEN
              QQ = 0
            RETURN

2160  PRINT"QUESTION NUMBER:";:GOSUB610:QQ=VAL(SS$):
      IFQQ<0ORQQ>NQ(GX)THENQQ=0
2170  RETURN
```

La routine "qnum" chiede un numero di domande e ne controlla la validità.

Figura 7.56 — Ottenere un numero di domanda

dove $a_0, a_1, \ldots, a_k$ sono numeri interi nella gamma da 0 a b − 1. Per esempio, usando b = 10,

$$1980 = 0 + 8 \times 10 + 9 \times 100 + 1 \times 1000;$$

e cioè

$$1980 = 0 + 8 \times 10 + 9 \times 10^2 + 1 \times 10^3$$

Non esiste altro modo per spezzettare 1980 in unità, decine, centinaia e migliaia con ognuno dei coefficienti (cioè 1, 9, 8, 0) nella gamma da 0 a 9.

Per illustrare il raggruppamento delle preferenze, assumiamo che ogni domanda può avere fino a 9 scelte, cioè, MC ha valore 9. Poi le preferenze raggruppate sono rappresentate da un numero con base 10 (MC + 1). Ciascuna cifra è il numero della scelta compresa tra lo zero e il nove (lo zero significa nessuna scelta) che corrisponde a uno dei valori delle preferenze. Nella Figura 7.13 le preferenze di SUSAN per la domanda uno sarebbero codificate come 1432; le preferenze di SUSAN per la domanda due sarebbero codificate come 1203.

Con numeri piccoli di valori di preferenze e di scelte di domande, questo schema fa sì che sia possibile codificare in un solo numero le preferenze di un giocatore per una certa domanda. Questo va bene finchè il numero diventa così grande che viene arrotondato. Il limite al quale questo arrotondamento avviene varia da sistema a sistema. Su di un Apple o un Pet, è nove cifre; su di un TRS—80 è sei cifre, a meno che venga usata una variabile a "doppia precisione". In questo caso l'array 6 x 12 x 2 che

```
#Alcune routines ben conosciute — versioni per il Pet

onech        repeat
               GET X$
               until (X$ < > "")
             RETURN

clearscreen  PRINT "clr";
             RETURN

2180  GETX$:IFX$=""THEN2180
2190  RETURN
2200  PRINTCHR$(147);:RETURN
```

Le versioni per l'Apple e per il TRS-80 di queste routines sono come abbiamo mostrato nelle Figure 2.11 e 2.12.

Figura 7.57 — Un input ad un solo carattere e la pulizia dello schermo

ha usato 1.104 bytes ne prenderebbe invece 2.196. La grandezza massima che una preferenza codificata può raggiungere può essere derivata dalla formula che segue

$$M = (MC + 1)^{NF} - 1$$

Per esempio, se MC ha valore 9 e NF (il numero di valori delle preferenze) ha valore 4, allora M = 9999.

Abbiamo visto come la codifica delle preferenze può evitare l'uso di larghe parti della memoria. Il prezzo da pagare però per questo risparmio di spazio della memoria si vede nel programma "pairup" (Figura 7.33), dove le chiamate 2* NP (GC) a "score" (Figura 7.34) che a sua volta chiama "fit" (Figura 7.35) risultano in

NP(GC)*NQ(GC) + NP(GC)*NQ(GP)

chiamate per "unpackwish". Tutte queste chiamate potrebbero essere eliminate se gli array più grandi che abbiamo considerato prima fossero stati usati, ma diversi sistemi di computer sarebbero incapaci di assegnare così tanta memoria a quell'array. Tuttavia c'è un accorgimento che si può applicare. Nella prima metà del programma "pairup" le preferenze di un certo giocatore vengono controllate con le risposte di ogni altro giocatore nell'altro gruppo così da smembrare il gruppo delle preferenze codificate del giocatore NP(GC) volte. Se, invece, queste preferenze fossero smembrate tutte una sola volta e conservate in una versione bidimensionale dell'array FT, allora il primo termine della somma sopra illustrata potrebbe essere ridotto da NP(GC)*NQ(GC) a NQ(GC). L'array bidimensionale FT avrebbe dimensioni NF e MQ. Usando gli stessi numeri che abbiamo usato prima, questo significherebbe che l'array FT sarebbe allargato da 4 voci a 4 x 6 = 24 voci e il numero delle chiamate a "unpackwish" sarebbe ridotto da 12 x 6 + 12 x 6 = 144 a 6 + 12 x 6 = 78. Questo sembra soltanto un aumento minimo dell'uso della memoria che si ottiene in cambio di una piccola riduzione del tempo necessario per eseguire l'operazione, ma se il numero dei giocatori in ciascun gruppo aumentasse da 12 a 24, allora non ci sarebbe nessun altro aumento dell'uso della memoria, e il numero delle chiamate sarebbe ridotto da 24 x 6 + 24 x 6 = 288 a 6 + 24 x 6 = 150.

Questo conclude la nostra discussione sul risparmio di spazio necessario. (Vi potrebbe interessare di portare avanti il cambiamento che è stato accennato qui sopra).

Ci si pongono davanti diversi problemi che nascono da come organizzare il sistema di comunicazione tra "pairup" "score" "fit" e "unpackwish" per la nuova struttura. Per esempio bisognerebbe riscrivere "unpackwish" così da poter spezzettare tutti i gruppi dei diversi desideri del giocatore in una volta sola? In caso contrario, esisterebbero due versioni diverse di FT — una con una dimensione e una con due? Come misurereste l'esatto risparmio di tempo che risulta da questo cambiamento?

Questo conclude la nostra discussione sul programma del gioco degli Accoppiamenti.

Dovreste leggere tutti gli elenchi del programma, tenendo a mente quello di cui abbiamo parlato, e cercando di vedere che altro potreste imparare.

CAMBIAMENTI E MIGLIORIE

Dopo (o durante) il vostro programma di studi del programma del gioco degli Accoppiamenti, potreste voler introdurre alcuni dei cambiamenti descritti qui di seguito:

- Incrementare le caratteristiche di archivi esterni.
- In questo caso, se si rientra la fase di costruzione del gioco dopo aver preparato gli archivi, il giocatore e gli archivi delle domande possono diventare incompatibili.

```
#Routines di registrazione con mezzi esterni al sistema

loadin        GOSUB copout
              RETURN

writeout      GOSUB copout
              RETURN

loadplayers   GOSUB copout
              RETURN

saveplayers   GOSUB copout
              RETURN

copout        PRINT "EXTERNAL STORAGE IS NOT IMPLEMENTED"
              RETURN

2210  GOSUB2250:RETURN
2220  GOSUB2250:RETURN
2230  GOSUB2250:RETURN
2240  GOSUB2250:RETURN
2250  PRINT "NOT IMPLEMENTED":RETURN
```

Queste routines sono "matrici" che indicano il posto nel programma al quale verrà inserito il codice per l'uso dei sistemi di registrazione esterni al sistema dei diversi dati.

Figura 7.58 — Possibilità di registrazione esterna al sistema

```
#Inizializzazione

init        GM = 2                                  #due gruppi soltanto
            READ MQ,MC,MP                           #numero massimo delle
                                                    domande, delle scelte e
                                                    dei giocatori
            DATA maxquestions, maxchoice, maxplayers
            DIM NQ(GM), NP(GM),                     #numero delle domande
                                                    e dei giocatori
            GN$(GM), NM$(MP,GM),                    #nomi dei gruppi, nomi
                                                    dei giocatori
            Q$(MQ,GM), WT(MQ,GM)                    #domande, pesi numeri-
                                                    ci
            NC(MQ,GM), C$(MC,MQ,GM),                #numero delle scelte,
                                                    scelte
            A(MQ,MP,GM), W(MQ,MP,GM)                #risposte dei giocatori,
                                                    desideri
            READ MT: DATA maxtop                    #grandezza dell'array
                                                    dei punteggi più alti
            DIM TS(MT),TI(MT)                       #punteggi più alti e nu-
                                                    meri dei giocatori
            READ NF: DATA numfit                    #grandezza dell'array
                                                    dei desideri
            DIM FV(NF),FT(NF)                       #valore dei desideri e
                                                    arrays di espansione
            FOR FX = 1 TO NF                        #leggere i valori dei de-
                                                    sideri
              READ FV(FX)
              NEXT FX
            DATA highest wish value,...,lowest wish value
            RETURN


2260* GM=2:READMQ,MC,MP:DATA2,5,2
2270  DIMNQ(GM),NP(GM),GN$(GM),NM$(MP,GM),Q$(MQ,GM)
2280  DIMWT(MQ,GM),NC(MQ,GM),C$(MC,MQ,GM)
2290  DIMA(MQ,MP,GM),W(MQ,MP,GM)
2300  READMT:DATA3
2310  DIMTS(MT),TI(MT)
2320  READNF:DATA4
2330  DIMFV(NF),FT(NF)
2340  FORFX=1TONF:READFV(FX):NEXTFX
2350  DATA 2,1,-1,-2
2360  RETURN
```

Figura 7.59 — Inizializzazione *(continua)*

Le dichiarazioni dei dati permettono di fissare le grandezze degli arrays. Per un piccolo computer (8K di RAM) le grandezze degli arrays devono essere fissate a valori minimi — come 2 domande, 5 scelte, e 2 giocatori per gruppo. Valori più grandi possono essere usati con sistemi più grandi.

---

* I valori di "maxquestion", "maxchoices" e "maxplayers" che compaiono in questa linea sono abbastanza piccoli da permettere al programma di essere usato su di un 8K Pet. Valori più grandi si possono usare con sistemi più grandi. Su di un 16K TRS-80, per esempio, valori di 10, 5, 30 possono essere usati. Ma ricordatevi che "maxchoice" non può superare 9.

Figura 7.59 — Inizializzazione *(Fine)*

Considerate le caratteristiche seguenti:

— Apportare dei cambiamenti automatici agli archivi relativi al giocatore quando si aggiungono delle domande o quando se ne cancellano dagli archivi delle domande.

— Ricodificare automaticamente le preferenze del giocatore se i valori di NF o di MC cambiano.

— Svuotare gli archivi relativi al giocatore automaticamente, o dietro comando, in determinate circostanze.

Incrementate una qualsiasi di queste caratteristiche secondo un vostro ordine logico di importanza. Che problemi non vengono però risolti da questi cambiamenti?

- Far sì che esistano dei comandi di editing per gli archivi del giocatore che vi diano la possibilità di specificare a che punto del ciclo dell'editing volete iniziare.
- Far sì che vi siano dei comandi ad un solo carattere che vi diano la possibilità di saltare direttamente fino all'inizio dell'editing delle preferenze, o che vi consentano di andare avanti fino alla domanda seguente senza dover continuamente usare RETURN per saltare tutti i valori delle preferenze relative alla domanda di cui vi state occupando.
- Correggete il problema inerente in "storenew" (Figura 7.22).
- Far sì che sia possibile avere un accoppiamento di massa per mezzo di una funzione che generi una lista di coppie così da massimizzare il totale di tutte le preferenze dei giocatori appaiati. Se un gruppo è più grande dell'altro, assegnate più di un membro del gruppo più grande a certi membri del gruppo più piccolo.

SOMMARIO

Il gioco degli Accoppiamenti usa gruppi di domande a diverse scelte per accoppiare membri di due gruppi. Il programma che implementa questo gioco illustra concetti

di programmazione generale e tecniche di programmazione sia generali che specifiche.

In particolare, i concetti di programmazione di cui abbiamo parlato sono la modularità, la struttura "sottosopra", l'isolamento di una funzione, la possibilità di alterazioni e l'uso delle matrici per definire la struttura. Le tecniche di programmazione di cui abbiamo discusso comprendono la progettazione di documentazione, e dell'interazione. Abbiamo inoltre parlato di editing per quanto riguarda le funzioni, di algoritmi di inserzione e di cancellatura, di progettazione di sistemi e di funzioni di controllo degli archivi e di tecniche per risparmiare spazio necessario.

CAPITOLO 8

CRAPS

Craps è la versione da piccolo computer del popolare gioco da casinò. Abbiamo ricreato l'azione rapida ed eccitante del gioco usando delle tecniche di programmazione abbastanza semplici.

ISTRUZIONI PER IL GIOCO

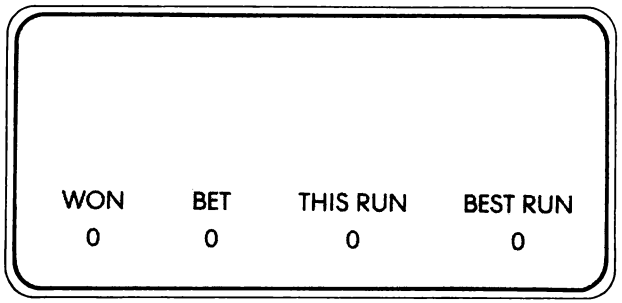
Nella sua forma originale, Craps è un semplice gioco che si fa coi dadi. Ciascun giocatore riceve a turno i dadi e li tira. Quando viene il vostro turno cominciate col fare una scommessa. Se la serie dei vostri tiri risulta in una vincita, ritirate la somma che avete scommesso, ne piazzate un'altra, e ricominciate con un'altra serie di tiri. Se, invece, i vostri tiri non sono fortunati e vi portano a perdere, pagate la scommessa, e passate i dadi ad un altro giocatore.

Se il vostro primo tiro in una serie è un 7 o un 11 (i numeri si riferiscono alla somma dei puntini sulle due facce dei dadi), allora vincete immediatamente. Se il vostro primo tiro è un 2, un 3 o un 12, allora perdete immediatamente. Se invece tirate una delle altre sei possibilità (4, 5, 6, 8, 9, 10) allora il numero diventa il vostro "punteggio". Dovete continuare a tirare i dadi finchè non avete un 7, nel cui caso perdete, o tirare il vostro "punteggio", nel cui caso vincete. Dunque, se il vostro punteggio è 8, o tirate un altro 8 e vincete, o tirate un 7 e perdete. Qualsiasi altro numero che esca durante questi tiri, compresi il 2, il 3, l'11 e il 12 non viene contato. Per esempio la seguenza di tiri qui illustrata risulterà in una vincita:

8, 5, 11, 6, 9, 3, 8

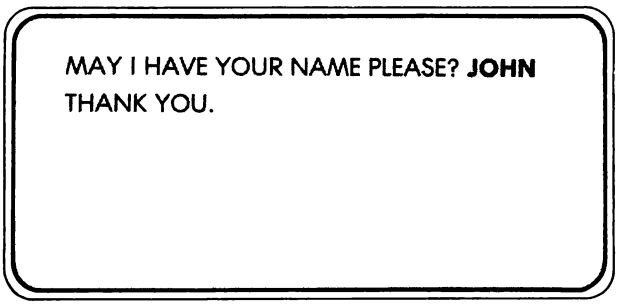
Qui abbiamo parlato soltanto di questo tipo più originale di Craps, ad un tavolo di casinò, ovviamente, le combinazioni sono più variate. In effetti, poichè si gioca su di un tavolo, voi potete scommettere che esca per esempio un 3, piazzando un gettone su un certo punto del tavolo. Se il 3 non esce, il banco si prende il gettone; se invece

il 3 esce, il banco pagherà 12 altri gettoni e li piazzerà accanto al gettone originale della scommessa. Se non volete scommettere tutte e 13 i gettoni dovrete molto velocemente ritirarli dal tavolo. L'azione ad un tavolo di craps è estremamente veloce, e ci sono così tante scommesse piazzate sul tavolo che ci si dimentica persino quali erano i propri gettoni.

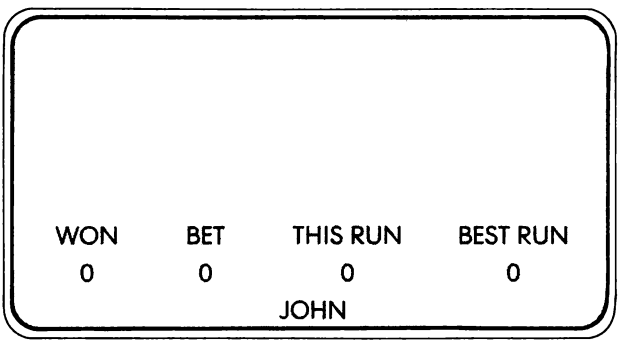


Display iniziale

Il giocatore schiaccia N



Il programma richiede il nome del giocatore e questi lo inserisce



Display iniziale con il nome

Il giocatore ha risposto al display iniziale schiacciando N (il comando per il "nuovo giocatore"). Il programma richiede il nome del giocatore e questi inserisce JOHN. Il programma risponde "THANK YOU". Poi, dopo una breve pausa, ritorna a mostrare sullo schermo il display iniziale che adesso però include anche il nome del giocatore.

Figura 8-1 — JOHN è il primo giocatore

Non abbiamo però inserito questo tipo di scommessa perchè non sarebbe stato molto facile rappresentare il tipo di schema di tavolo sullo schermo, e anche perchè non esiste la possibilità di input / output separato per ciascun giocatore. Tuttavia si potranno introdurre altre scommesse come spiegheremo poi nel capitolo riguardante i miglioramenti.

Il programma per la nostra versione di Craps inizia con il display di "inizio del gioco" (vedi Figura 8.1). Vi sono diversi comandi che possono essere inseriti a questo punto. Il gioco può essere giocato da una sola persona, o da più di una. Se vi è più di un giocatore, si potrà usare il comando N per inserire il nome di ciascun giocatore. Questo viene spiegato nella Figura 8.1. Nella Figura 8.2, si vede come il programma annuncia la sequenza dei giocatori. (Nota: siccome l'input dei nomi usa il comando INPUT, i nomi composti di due parole (per es. JOHN SMITH) devono essere inseriti tra virgolette).

Supponiamo che anche MARY e SUSAN giochino con JOHN, ci sono dunque ora tre giocatori. È ancora il turno di JOHN. Ora JOHN dovrà piazzare una scommessa. (Figura 8.3).

Dopo aver piazzato una scommessa di 25 JOHN comincia a tirare i dadi schiacciando la sbarretta dello spazio una volta.

Nella Figura 8.4 vediamo che cosa succede poi. Per prima cosa appaiono un paio di dadi sullo schermo. Poi il risultato del tiro viene scritto sullo schermo stesso (per es. SORRY, YOU LOSE — MI SPIACE, HAI PERSO — oppure POINT IS 5 — IL PUNTEGGIO È 5 —). Dopo una breve pausa, i dadi scompaiono, ma il risultato rimane. Poi, se il punteggio è una vincita, il display iniziale corrispondente a JOHN riappare. Se invece si tratta di una perdita (come illustrato nella Figura 8.4) sullo schermo compare il display che si riferisce al prossimo giocatore, (in questo caso MARY). Se il risultato non è nè una vincita nè una perdita, compare una linea che dà sia il punteggio che il numero di tiri effettuati.

MAY I HAVE YOUR NAME PLEASE? **MARY**
THANK YOU. YOU WILL FOLLOW JOHN.

In risposta ad un altro comando N, il programma ha chiesto il nome del giocatore. Questa volta è stato inserito il nome MARY, e il programma ha piazzato MARY dopo JOHN nel turno.

Figura 8.2 — MARY segue JOHN

Questo è proprio quello che succede a MARY (Figura 8.5) MARY piazza una scommessa di 50, poi schiaccia la sbarretta dello spazio per tirare i dadi. Compaiono un 6 e un 2, poi il risultato (POINT IS 8), ed infine, la "linea finale" (POINT:8 THROWS:1). I dadi scompaiono, e il programma aspetta che il giocatore tiri ancora. MARY schiaccia la sbarretta e appaiono due 4 — MARY vince. Il display iniziale che corrisponde a MARY compare un'altra volta. I punteggi di quanto ha vinto (WON), di questo tiro (THIS RUN) e del tiro migliore (BEST RUN) vengono aggiornati, mentre quello della scommessa (BET) rimane uguale.

Ora che avete un'idea di come si gioca, diamo un'occhiata ai comandi che potete inserire quando il display iniziale compare. Questa lista è illustrata nella Figura 8.6.

Ora discuteremo questi comandi uno per uno.

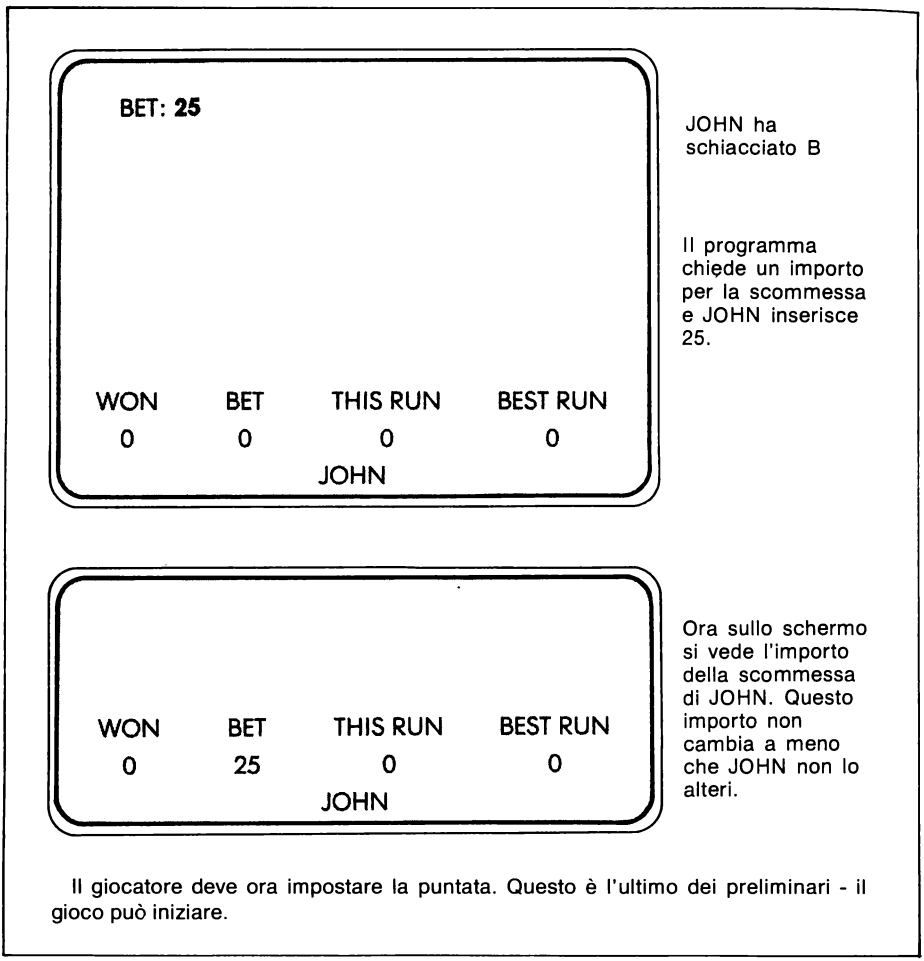
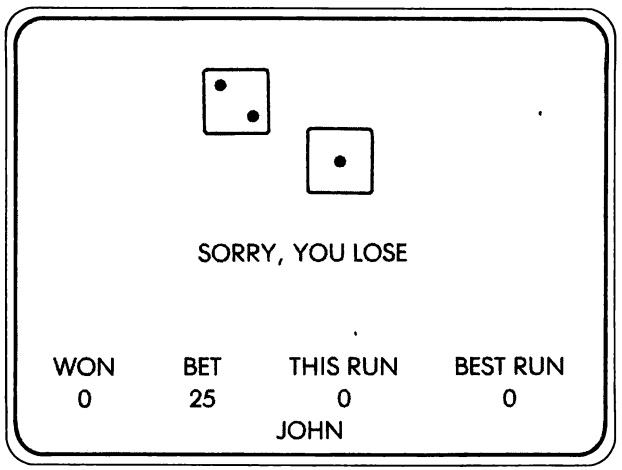


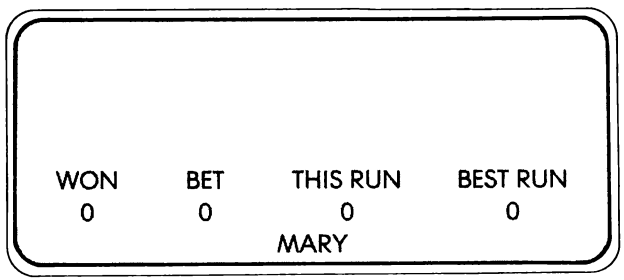
Figura 8.3 — La scommessa di JOHN

Il comando N (vedi Figura 8.1) viene usato per inserire il nome di un nuovo giocatore. N fa sì che un'inserimento corrispondente vada nell'array delle statistiche, che viene usato per ricordare l'informazione (WON, BET, ecc.) dalla linea in basso al vostro display iniziale mentre altri giocatori stanno giocando. Così, quando arriva di nuovo il vostro turno, il programma non ha dimenticato i dati che vi riguardano.

Il comando P fa sì che possiate passare i dadi al giocatore che viene dopo di voi, senza aspettare che perdiate. (Normalmente tenete i dadi finchè continuate a vincere).



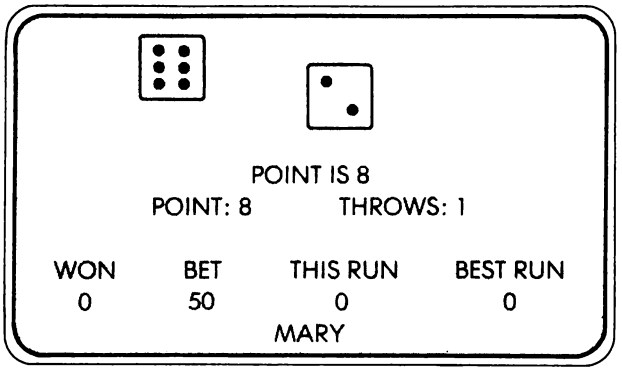
JOHN schiaccia la sbarra dello spazio. Prima di tutto compaiono i due dadi.
Poi il messaggio: SORRY YOU LOSE (Mi spiace, hai perso)



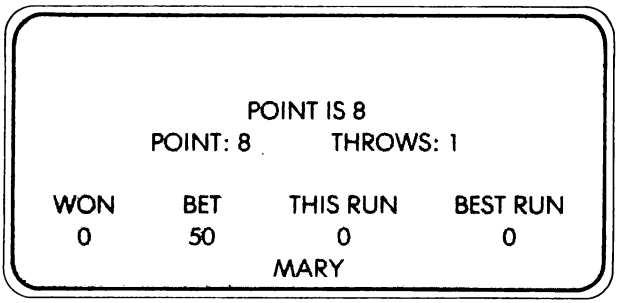
Dopo una breve pausa, sullo schermo compare il display relativo a MARY.

Il turno di JOHN consiste in un solo tiro dei dadi. Un tiro di 3 fa perdere JOHN immediatamente, e i dadi passano al giocatore seguente.

Figura 8.4 — JOHN tira i dadi

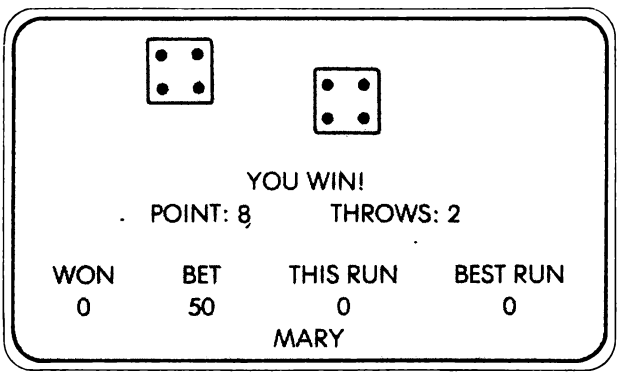


Prima di tutto compaiono i dadi sullo schermo.
Poi compare: "POINT IS 8" (Il punteggio è 8)
Poi compare: "POINT: 8 THROWS: 1" (Punteggio: 8 Tiri: 1)



I dadi scompaiono

Il programma aspetta che MARY tiri un'altra volta, (cioè che schiacci la sbarretta dello spazio)



MARY tira un altro 8

Il display iniziale compare con anche la vincita di MARY.

Figura 8.5 — MARY tira e fa un buon punteggio

Così i dati che vi riguardano rimangono intatti, e vengono richiamati quelli del giocatore che viene dopo di voi.

Il comando Q permette ad un giocatore (qualsiasi — non solo quello il cui display iniziale compare sullo schermo) di smettere di giocare. Nella Figura 8.7 vediamo il dialogo necessario per questa operazione. Una lista di nomi e numeri relativi ai giocatori compare sullo schermo. Qualsiasi giocatore può così abbandonare la partita semplicemente inserendo il proprio numero.

Il comando S non compie nessuna operazione. (Quando discuteremo il programma in seguito spiegheremo perchè è stato incluso).

La sbarretta dello spazio è usata per iniziare il gioco. Il primo tiro del giocatore compare sullo schermo, e il gioco procede come sopra descritto.

I comandi per le scommesse consentono di poter cambiare l'importo della scom-

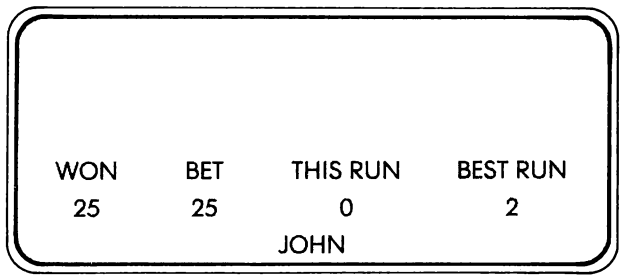
COMANDO	SIGNIFICATO
N	Nuovo giocatore. Il programma chiede il nome del giocatore e lo inserisce nel turno di gioco.
P	Passare i dadi. Appare il display iniziale relativo al giocatore seguente.
Q	Smettere. Un giocatore può smettere di giocare. Compare sullo schermo una lista di giocatori e dei numeri relativi. Si inserisce il numero del giocatore che vuole smettere di giocare, e il suo numero viene eliminato (se si inserisce uno zero il processo viene abortito e nessun numero viene cancellato). Nella lista sono comprese anche le vincite o le perdite di ciascun giocatore.
S	Scommesse di altri giocatori, non del giocatore che sta tirando i dadi. Questo comando non è stato reso funzionante.
spazio	Primo tiro. Il primo tiro del giocatore viene effettuato, iniziando la serie corrente.
B	Scommessa. Il giocatore può inserire una scommessa usando un importo in numero.
D	Raddoppio. L'importo della scommessa viene raddoppiato.
L o W	Perdite/Vincite. L'importo totale perso o vinto viene riscommesso.
O	Importo originale. L'ultimo importo che è stato inserito con un comando B viene considerato l'importo della scommessa.
R	Somma. L'importo della scommessa è la somma dell'importo della scommessa che è stata piazzata per ultima più l'importo della vincita. Questo comando ha lo stesso risultato del comando D.

Figura 8.6 — Comandi che si possono usare nel display iniziale

messa. Nella Figura 8.3 vediamo come il comando B viene usato per inserire una scommessa. Gli altri comandi che regolano le scommesse (D, L, O, R, W) servono per cambiarne l'importo sulla linea delle statistiche, ma non cambiano il resto del display.

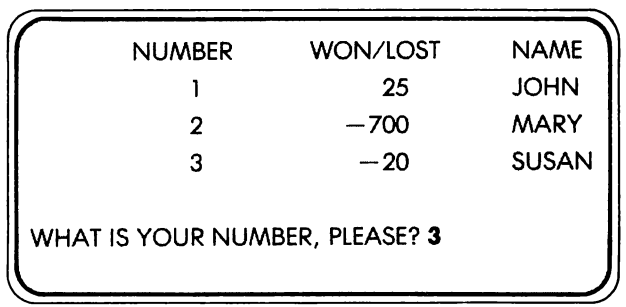
IL PROGRAMMA

Il programma di questo gioco viene illustrato dalla Figura 8.8 fino alla 8.39. Abbia-

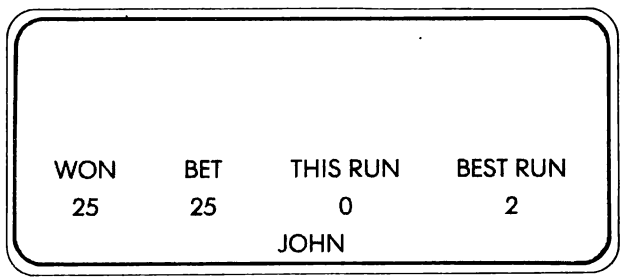


Display iniziale relativo a JOHN

SUSAN vuole smettere di giocare. Il giocatore inserisce Q



SUSAN viene eliminata



Il display iniziale relativo a JOHN riappare

Dopo un certo tempo durante il gioco siamo ritornati al turno di JOHN, e SUSAN si vuole ritirare. Il comando Q fa sì che sullo schermo compaia un display in cui il numero relativo a SUSAN è 3. Dopo che SUSAN si è ritirata è sempre il turno di JOHN.

Figura 8.7 — SUSAN si ritira

```
#Craps
            GOSUB initialize                    #fissare le costanti e i
                                                formati
            repeat {
              GOSUB start                       #cancellare la partita
                                                precedente
              GOSUB onech                       #ottenere un carattere
                                                come input
              GOSUB incode                      #decodificarlo (ritorna-
                                                re IC)
              ON IC GOSUB
                askbet,                         #stabilire l'importo della
                                                scommessa
                game,                           #giocare una partita
                new,                            #un nuovo giocatore
                                                vuole giocare
                quit,                           #un giocatore si ritira
                sidebet,                        #scommesse da altri
                                                giocatori, non quello che
                                                sta tirando i dadi
                pass                            #passare i dadi
            }
100  GOSUB 1950
130* GOSUB 960:GOSUB2340:GOSUB 160
140  ON IC GOSUB 560,230,1400,1690,1820,1560
150  GOTO 130
```

Questo è il loop principale di Craps. Ogni passaggio nel loop è per una partita. Il giocatore inizia con l'inserire un input di un solo carattere che determina che cosa seguirà. La sbarretta dello spazio, se schiacciata, fa tirare i dadi. Altri inputs vengono usati per far sì che un giocatore possa ritirarsi dal gioco o possa entrare a far parte del gioco, o per piazzare altre scommesse che non siano quelle che il giocatore in possesso dei dadi piazza. Le scommesse possono essere inserite numericamente, possono essere raddoppiate, aggiunte all'intero importo vinto o perso, o riportate all'importo originale. Anche le vincite possono essere riscommesse.

La routine "start" mostra il nome del giocatore in possesso dei dadi sullo schermo e i suoi risultati. Non appena il giocatore perde, i dadi passano al giocatore seguente.

---

* A causa della lunghezza della routine di inizializzazione nell'Apple e nel TRS-80, le routine illustrate nelle figure 8.37, 8.38 e 8.39 hanno dei numeri di linee diversi nelle versioni per l'Apple e per il TRS-80. Questi sono:

Routine	*Linea nel Pet*	*Linea nell'Apple e TRS-80*	*Figura*
draw	2240	2500	8.38
cleardice	2300	2580	8.39
onech	2340	2660	8.37

Quindi il "GOSUB 2340" nella linea 130 diventa "GOSUB 2660" nelle versioni per l'Apple e per il TRS-80.

Figura 8.8 — Craps

mo già parlato, nei capitoli precedenti, di diversi aspetti di questo programma, per cui non ne discuteremo in dettaglio. Questi aspetti sono:

- Struttura modulare e "sottosopra".
- Isolamento delle funzioni (per esempio "throw" e "draw").
- Selezione di una lista per identificare un giocatore (Figure 8.31 e 8.32).
- Sistemi di cancellatura da un array (Figura 8.32).
- Funzioni di controllo degli archivi (Figura 8.34).

Ci occuperemo dunque soltanto di questi nuovi aspetti:

- Uso dello schermo per ottenere un insieme di informazioni costanti e variabili.
- Come far sì che il gioco risulti veloce e scorrevole.
- Scambio di contesto.
- Scelte e risultati affidati al caso.

```
#Analizzare i codici dell'input

Incode      IF case
              X$ = " " THEN IC = 2          #sbarretta dello spazio
                                            per giocare
              X$ = "N" THEN IC = 3          #nuovo giocatore
              X$ = "Q" THEN IC = 4          #il giocatore si ritira
              X$ = "S" THEN IC = 5          #scommesse seconda-
                                            rie
              X$ = "P" THEN IC = 6          #passare i dadi
            else
              IC = 1                        #errore: trattare come
                                            un codice di scommessa
            RETURN

160 REM INCODE
170 IF X$=" " THEN IC=2:RETURN
180 IF X$="N" THEN IC=3:RETURN
190 IF X$="Q" THEN IC=4:RETURN
200 IF X$="S" THEN IC=5:RETURN
210 IF X$="P" THEN IC=6:RETURN
220 IC = 1:RETURN
```

La routine "incode" analizza il primo input del gioco. Se il carattere viene riconosciuto dalla routine, questa determina il valore in IC, che viene usato nella costruzione ON...GOSUB nella routine principale. Se il carattere non viene riconosciuto, ad IC viene dato il valore d'errore, che fa sì che il comando venga trattato come un comando per fissare le scommesse dalla routine "askbet".

Figura 8.9 — Decodificare il primo input ad un solo carattere

Uso dello schermo

Il programma qui presentato si regge sull'uso di uno schermo. Infatti il gioco sarebbe rovinato se si usasse un sistema di scrittura, per esempio un teleprinter. All'inizio di ciascun tiro, lo schermo viene pulito. Compaiono i dadi in un punto dello schermo, in un altro le statistiche, e in un altro ancora il risultato del tiro e poi ancora la li-

```
#Giocare una partita

game    GOSUB throw                          #tiro iniziale
        IF D = 2, 3 or 12 THEN               #un 2, un 3 o un 12 perdono
          {GOSUB lose: RETURN}
        else IF D = 7 or 11 THEN             #un 7 o un 11 vincono
          {GOSUB win: RETURN}
        else{
          PT = D: TH = 1                     #fissare il punto, il numero dei tiri
          RS$ = "POINT IS" + STR$(PT)        #annunciare il "risultato" del tiro
          GOSUB result
          }
        GOSUB gamestats                      #mostrare il punto e i tiri
        repeat{                              #cercare di fare il punto
          GOSUB onech                        #aspettare che il giocatore schiacci il tasto
          GOSUB throw: TH = TH + 1           #generare un tiro
          GOSUB gamestats                    #mostrare il punto e i tiri
          IF D = PT THEN
            {GOSUB win: break}               #lasciare che il punto vinca
          else IF D = 7 THEN
            {GOSUB lose: break}              #il 7 perde
          else{
            RS$ = STR$(D) + "— THROW AGAIN"
            GOSUB result
          }}
        RETURN
```

La routine "game" passa in rivista un intero gruppo di tiri. Un 7 o un 11 iniziale fa vincere immediatamente; un 2, un 3 o un 12 fa perdere subito. Qualsiasi altro tiro diventa il "punto" e i tiri successivi continueranno finchè non si tira il "punto" o un 7.

Figura 8.10 — Un tiro di dadi

nea con il punteggio e i tiri effettuati. Ciascuno di questi dati ha un suo preciso spazio e cancella qualsiasi informazione non aggiornata.

L'uso separato di diversi spazi dello schermo si basa su di una routine che posiziona il cursore (vedi Figura 8.27) e fa sì che sia possibile fermarlo su di una linea o su di una colonna particolari. Questa scelta e il suo posizionamento vengono effettuati durante la routine d'inizializzazione (Figura 8.35).

Il display dei dati viene compiuto dalle routines "draw" e "cleardice" (Figure 8.38 e 8.39). La routine "draw", illustrata nella Figura 8.38, si basa sui caratteri che muovono il cursore nel Pet. Dopo aver posizionato il cursore per ciascuna figura, la routine "draw" mostra la stringa BX$, che fa comparire un quadrato sullo schermo.

Questa "scatola" conterrà poi i puntini, e il cursore viene così posizionato all'angolo sinistro in alto di questo quadrato. La versione del Pet della stringa BX$ contiene dei caratteri per "cursore a sinistra", "cursore in basso" e "cursore in alto", che sono mischiati con i caratteri di linea orizzontale e verticale (shift $, shift # shift% e shift ') che costruiscono la "scatola". Dopo che questa si trova in posizione, viene stampata una delle stringhe a sei facce proveniente dall'array D$. Queste stringhe combinano spazi, puntini circolari (shift Q) e comandi di movimento del cursore così da riempire le nove posizioni all'interno della "scatola".

Questo display si può ottenere anche su di altri sistemi. La versione per l'Apple (che però non viene illustrata qui) usa un array bidimensionale D$(6,5) invece di quello monodimensionale D$(6) usato per la versione del Pet. Nell'Apple non c'è la stringa BX$ per la "scatola". Invece, ciascuna figura consiste di cinque linee, com-

```
230  REM GAME
240  GOSUB 370
250  IF D=2 OR D=3 OR D=12 THEN GOSUB 490:RETURN
260  IF D=7 OR D=11 THEN GOSUB 420:RETURN
270  PT=D:TH=1
280  RS$="POINT IS"+STR$(PT):GOSUB 1020
290  GOSUB 980
300* GOSUB 2340:GOSUB 370
310  TH=TH+1:GOSUB 980
320  IF D=PT THEN GOSUB 420:GOTO 360
330  IF D=7 THEN GOSUB 490:GOTO 360
340  RS$=STR$(D)+"-THROW AGAIN":GOSUB 1020
350  GOTO 300
360  RETURN
```

* Nelle versioni per l'Apple e per il TRS-80, 2340 viene sostituito da 2660 (Vedi Figura 8.8).

Figura 8.10a — BASIC per "Game"

presi i contorni della "scatola" e i puntini. Quindi, nell'Apple, il disegno della prima figura che servirà poi da stampo, viene eseguito dai seguenti comandi:

```
FOR LX = L1 TO L1 + 4
    LL = LX: CC = C1: GOSUB cursor
    PRINT D$(D1, LX = L1 + 1);
    NEXT LX
```

Una seguenza di comandi simile viene usata per cancellare lo stampo nella versione dell'Apple di "cleardice". I lati della "scatola" sono composti di trattini e due punti nell'Apple, mentre i puntini sono asterischi. La versione del TRS–80 (alla quale si riferiscono le note alla fine delle figure) è simile a quella dell'Apple, ma è possibile ottenere una versione per il TRS–80 che sia simile a quella del Pet.

Il display e la seguente cancellazione delle scommesse, del risultato, e delle informazioni riguardanti i tiri, il punteggio e il risultato delle statistiche sono facili da capire. La cancellazione viene fatta con delle stringhe di spazi vuoti.

Vedere le routines "clearbet" (Figura 8.23), "clresult" (Figura 8.24) e "clstat" (Figura 8.25).

Non è necessario cancellare la linea che dà il punteggio e il numero di tiri. La linea

```
#Tirare i dadi

throw       D1 = INT(6*RND(1) + 1)        #primo dado
            D2 = INT(6*RND(1) + 1)        #secondo dado
            GOSUB draw                    #mostrare i dadi
            D = D1 + D2                   #valore totale del tiro
            RETURN

370   REM THROW
380*  D1=INT(6*RND(1)+1)
390*  D2=INT(6*RND(1)+1)
400** GOSUB 2240
410   D=D1+D2:RETURN
```

La routine "throw" genera due numeri a caso compresi tra l'uno e il sei – i valori sulle facce dei due dadi. La subroutine "draw", che mostra i dadi sullo schermo, appare alla fine del programma, con "cleardice", "inizialize" e "wait". Queste quattro routines hanno significati diversi in diversi computers. Il resto del programma è praticamente identico nel Pet, nell'Apple e nel TRS-80.

* Nella versione per il TRS-80, RND(1) viene sostituito da RND(0).

**Nelle versioni per l'Apple e per il TRS-80, 2240 viene sostituito da 2500 (Vedi Figura 8.8).

Figura 8.11 – Un tiro

viene infatti riscritta, e la nuova linea copre sempre completamente la vecchia. (Figura 8.20).

Gioco veloce e scorrevole

Far si che il gioco risulti veloce e scorrevole è un aspetto importantissimo della programmazione del gioco. Il programma di Craps raggiunge questo scopo usando:

- Un input con un solo carattere, senza suggerimenti importuni.
- Scomparsa automatica dei dadi e dei messaggi dallo schermo dopo un certo intervallo.
- Zone diverse di display che possono cambiare toccando un tasto.

```
#Conteggio di una vincita

win     WN = WN + BT                    #aumentare le vincite
        RD = 2*BT                       #importo da riscommettere
        PS = PS + 1                     #un'altra vincita consecutiva
        BR = max(BR,PS)                 #BR = punteggio migliore
        ZP = CP: GOSUB savestats        #aggiornare gli arrays
        RS$ = "YOU WIN!": GOSUB result  #annunciare il risultato
        GOSUB delay                     #dare al giocatore il tempo per leggerlo
        RETURN

420 REM WIN
430 WN=WN+BT
440 RD=2*BT:PS=PS+1
450 IF PS>BR THEN BR=PS
460 ZP=CP:GOSUB 1890
470 RS$="YOU WIN!":GOSUB 1020
480 GOSUB 1380:RETURN
```

Quando il giocatore vince, viene ciamata la routine "win" per aggiornare le statistiche e per annunciare il risultato.

Figura 8.12 — Conteggi dopo una vincita

Scambio di contesto

La nostra versione di Craps fornisce un ottimo esempio di scambio di contesto, che è certamente uno degli aspetti più importanti dei programmi più sofisticati come quelli a multiprogrammazione. Per applicare il linguaggio del computer al tavolo da gioco, diciamo che ciascun giocatore rappresenta un "processo". Il vostro processo è formato dalle azioni che compite durante il vostro turno: cioè scommettere e tirare i dadi. Il processo degli altri giocatori è formato dalle loro azioni durante i turni che gli spettano.

In un casinò, il tavolo dei dadi, i dadi e la posizione della scommessa sul tavolo sono tutte "risorse comunitarie". Soltanto un "processo" per volta può usarle (per esempio, soltanto un giocatore per volta può tirare i dadi); gli altri "processi" vengono dunque "sospesi" mentre questo ha luogo.

Nel nostro programma di Craps, la tastiera, lo schermo, e i programmi di controllo sono tutte risorse comunitarie e i processi sono le diverse interazioni dei giocatori con queste risorse. Le uniche voci che variano da programma a programma sono il

```
#Conteggio di una perdita

lose        WN = WN - BT                          #diminuire le vincite
            RD = OB                               #importo da riscommettere = "vecchia scommessa"
            PS = 0                                #tiro è finito
            ZP = CP: GOSUB savestats              #aggiornare gli arrays
            RS$ = "SORRY, YOU LOSE": GOSUB result #annunciare il risultato
            GOSUB delay                           #dare al giocatore il tempo per leggerlo
            GOSUB pass                            #passare i dadi al giocatore seguente
            RETURN

490   REM LOSE
500   WN=WN - BT
510   RD=OB:PS=0
520   ZP=CP:GOSUB 1890
530   RS$="SORRY, YOU LOSE":GOSUB 1020
540   GOSUB 1380:GOSUB 1560
550   RETURN
```

Quando il giocatore perde, la routine "lose" viene chiamata per aggiornare le statistiche, annunciare il risultato e passare i dadi al giocatore seguente.

Figura 8.13 — Conteggi dopo una perdita

```
#Chiedere la scommessa
askbet      repeat {
              GOSUB betcode                    #valutare il carattere
                                               dell'input
              ON B GOSUB
                double,                        #raddoppiare la scom-
                                               messa
                ride,                          #riscommettere le vinci-
                                               te
                won,                           #importo della scom-
                                               messa vinto/perso
                old,                           #ritornare all'importo
                                               precedente
                number                         #inserire una scommes-
                                               sa in numeri
              IF BT <= 0 THEN {                #rendere nullo il codice
                                               della scommessa
                LL = BL
                CC = BC
                GOSUB cursor
                PRINT "SORRY, PLEASE..."
                GOSUB onech                    #ottenere un altro input
                }
              else IF BT > MX THEN {           #la scommessa eccede
                                               il limite
                LL = BL
                CC = BC
                GOSUB cursor
                PRINT "HOUSE LIMIT IS"; MX     #dichiarare il limite per
                                               le scommesse
                BT = MX                        #fissare le scommesse
                                               al massimo
                IF OB > MX THEN
                  OB = MX
                GOSUB delay                    #dare al giocatore il
                                               tempo di leggere
                }
            } until (0 < BT <= MX)
          RETURN
```

La routine "askbet" esamina il carattere singolo che rappresenta il comando per la scommessa che è stato passato in X$ dalla routine principale. Chiama una delle routine "double", "ride", "won", "old" o "number" e poi valuta l'importo della scommessa BT che ritorna.

Figura 8.14 — Il comando per la scommessa

nome del giocatore e le sue statistiche. Queste costituiscono il contesto in cui i programmi vengono eseguiti quando un processo relativo ad un qualsiasi giocatore è attivo. In particolare, il contesto è formato dalle variabili WN (vincite / perdite) BT (importo della scommessa), BR (miglior punteggio), O8 (scommessa originale) e CP (numero del giocatore che sta giocando — questo viene anche usato come indice per l'array dei nomi, e come uno degli indici dell'array delle statistiche).

Lo scambio di contesto avviene durante la routine "pass" nella Figura 8.30. In questo esempio, la maggior parte del macchinario e del programma vengono usati da più di un giocatore — le uniche risorse separate sono infatti solo le loro caselle negli arrays delle statistiche e dei nomi. Questo cambiamento così semplice è dunque la base di qualsiasi programma a multiprogrammazione. In un programma più complesso, come un sistema a partizione del tempo, ciascun processo può avere sia il proprio schermo che la propria tastiera. In questo caso, lo scambio di contesto avverrà molto più frequentemente — e cioè ogni volta che il programma sia in attesa di input dalla tastiera di uno dei processi, il processo seguente potrà funzionare. Non notereste mai che il programma stia occupandosi di una scommessa e di un tiro di un altro giocatore, infatti avreste l'illusione di avere il completo controllo del macchinario. Mentre siete in attesa della risposta del programma, questo sarà a vostra completa disposizione, ma mentre vi sembra che il programma sia in attesa di una vostra mossa, in effetti esso non vi presterà la benchè minima attenzione. (Questo è simile a quello che faceva il grande cantante Chaliapin, che sosteneva che, respirando quando nessuno se lo aspettava e non respirando quando sarebbe stato ovvio, dava l'impressione di non respirare mai).

```
560  REM ASK FOR BET
570  GOSUB 700
580  ON B GOSUB 860,880,900,920,770
590  IF 0<BT AND BT<=MX THEN 690
600  IF BT<=MX THEN 650
610  LL=BL:CC=BC:GOSUB 1270
620* PRINT "HOUSE LIMIT IS";MX;"";
630  BT=MX:IF OB > MX THEN OB=MX
640  GOSUB 1380:GOTO 690
650  LL=BL:CC=BC:GOSUB 1270
660  PRINT "SORRY, PLEASE ENTER A NEW BET CODE";
670** GOSUB 2340
680  GOTO 570
690  RETURN
```

* La versione per l'Apple di questa linea differisce soltanto nella spaziatura della costante di stringa.

** Vedere la nota nella Figura 8.8.

Figura 8.14a — BASIC per "askbet"

Vi è una buona parte che riguarda i programmi a multiprogrammazione che non abbiamo neanche toccato, ma il programma di Craps vi darà un'idea sufficiente di come funzionino questi sistemi.

Scelte e risultati affidati al caso

È importante far sì che i risultati di ogni tiro non siano predicibili. Sfortunatamente, poichè sia un programma che un computer si possono comportare allo stesso modo ogni qualvolta vengano accesi, la sequenza dei numeri generati dalla funzione affidata al caso, RND, si può predirre. Il processo con il quale il programmatore nasconde questa predicibilità al giocatore viene chiamata "casualità". La tecnica usata nella

```
#Interpretare il codice per la scommessa rappresentato da un solo carattere

betcode     IF case
              X$ = "D"          THEN B = 1        #raddoppiare l'importo
                                                  della scommessa
              X$ = "R"          THEN B = 2        #riscommettere l'ultima
                                                  vincita
              X$ = "W" or "L"   THEN B = 3        #scommessa = totale
                                                  vinto/perso
              X$ = "O"          THEN B = 4        #scommessa = importo
                                                  originale
              X$ = "B"          THEN B = 5        #il giocatore inserirà un
                                                  numero
            else                       B = 4      #errore = originale
            RETURN

700  REM BETCODE
710  IF X$="D" THEN B=1:RETURN
720  IF X$="R" THEN B=2:RETURN
730  IF X$="W" OR X$="L" THEN B=3:RETURN
740  IF X$="O" THEN B=4:RETURN
750  IF X$="B" THEN B=5:RETURN
760  B=4:RETURN
```

La routine "betcode" prende i risultati passatigli da "incode" e li esamina. Se questa routine non è in grado di riconoscere il comando, questo viene trattato come il comando per l'importo della "scommessa precedente". Questa è un'azione di errore senza risultato. Il controllo ritorna al programma principale, cosicchè il giocatore che si inserisce erroneamente in "askbet" non può alterare il codice della scommessa.

Figura 8.15 — Decodificare il comando per la scommessa

```
#Accettare un numero

number      AA = 0                          #inizializzare il numero
            GOSUB betprompt                 #chiedere l'input
            repeat {
              GOSUB onech                   #il carattere seguente
              NN = ASC(X$) - ASC("0")       #va trasformato in una
                                            cifra
              IF NN < 0 OR NN > 9 THEN      #se non lo è già
                 break
              PRINT X$;                     #ripetere la cifra
              AA = 10*AA + NN               #accumulare il numero
              }
            BT = AA                         #fissare l'importo della
                                            scommessa
            OB = BT                         #fissare la scommessa
                                            "originale"
            RETURN

#Suggerire un input

betprompt   GOSUB clearbet
            PRINT "BET:";
            RETURN

770  REM NUMBER
780  AA=0:GOSUB 940
790* GOSUB 2340
800  NN=ASC(X$)-ASC("0")
810  IF NN<0 OR NN>9 THEN 840
820  PRINT X$;:AA=10*AA+NN
830  GOTO 790
840  BT=AA:OB=BT
850  RETURN

940  REM BETPROMPT
950  GOSUB 1180:PRINT "BET:";:RETURN
```

La routine "number" e la sua subroutine "betprompt" forniscono un esempio di pessima programmazione (Spiegate il perchè). Una routine alternativa viene illustrata nella Figura 8.17.

---

* Nella versione per l'Apple e per il TRS-80, 2340 viene sostituito da 2660 (vedi Figura 8.8).

Figura 8.16 — Accettare un input numerico di scommessa

routine "onech" nella Figura 8.37 per ottenere questa casualità è quella di chiamare la funzione RND (numero a caso) per un numero non predicibile di volte. Il numero di chiamate non è predicibile perchè la chiamata fa parte di un loop che attende che il giocatore schiacci un tasto. Più tempo il giocatore aspetta a schiacciarlo, più volte viene chiamata la funzione RND. La durata del loop è così breve (al massimo qualche millesimo di secondo) che il giocatore non è assolutamente in grado di controllare il numero delle chiamate.

Questa tecnica funziona perfettamente sul Pet e sul TRS-80, ma non per l'Apple, poichè quest'ultimo ha un input a carattere singolo che aspetta finchè un carattere non è stato inserito prima di ritornare. Dunque è possibile usare il loop solo una volta.

In modo da far sì che ci sia questa casualità che non possa essere controllata dal dovrete usare le "palette". Dovrete dunque:

1. Far sì che il giocatore schiacci il tasto della "paletta".
2. Attendere che abbia finito di schiacciare il tasto.
3. Chiamare continuamente RND e controllare il bottone, finchè viene schiacciato.

L'azione numero 2 fa sì che il giocatore non possa controllare le chiamate per RND schiacciando il bottone prima che il programma abbia iniziato il controllo.

```
#Versione alternativa di un input numerico di scommessa
number      repeat{
              GOSUB clearbet                    #pulire la linea della
                                                scommessa
              PRINT "BET:";: GOSUB stringin     #ottenere un input fino a
                                                RETURN
              IF XX$ = "" THEN                  #valvola di scarico
                RETURN
              else
                NN = VAL(XX$)                   #convertire l'input in un
                                                numero
              }until (NN < > 0)                 #non accettare 0 o un
                                                input non numerico
            BT = NN: OB = BT                    #fissare l'importo della
                                                scommessa e la scom-
                                                messa "originale"
            RETURN
```

Questa routine usa la routine "stringin" del programma degli Accoppiamenti, e la funzione VAL che è insita nel BASIC.

Figura 8.17 — Input numerico migliore per la scommessa

SUGGERIMENTI E MIGLIORIE

Vi sono certamente diversi modi per migliorare questo gioco. Eccovene alcuni:

- Evitare che un giocatore un po' troppo entusiasta tiri due volte di seguito schiacciando per caso la sbarretta dello spazio. Questo si può fare inserendo

  repeat GET X$ until (X$ = " ")

  in uno spazio appropriato nel programma.

- Sviluppare un modo per introdurre l'intero standard di scommesse che si possono fare su un vero tavolo da casinò di Craps. Cioè non solo il giocatore a cui tocca tirare può scommettere o ritirare le proprie vincite durante la partita.

- Considerare un diverso sistema di conteggio, dando a ciascuno dei giocatori dei gettoni, e il numero dei gettoni che ogni giocatore ha in mano viene conteggiato e

```
#Subroutines di specifica della scommessa

double      BT = 3*BT: RETURN          #raddoppiare la scommessa
ride        BT = RD: RETURN            #riscommettere le vincite
won         BT = ABS(WN):RETURN        #scommettere l'intero importo vinto/perso
old         BT = OB: RETURN            #ritornare alla scommessa "originale"

860 REM DOUBLE
870 BT=2*BT:RETURN

880 REM RIDE
890 BT=RD:RETURN

900 REM WON
910 BT=ABS(WN):RETURN

920 REM OLD
930 BT=OB:RETURN
```

Queste sono le routines brevi chiamate da "askbet" per implementare le diverse possibilità di scommessa.

Figura 8.18 — Fissare la scommessa

```
#Iniziare il gioco

start       GOSUB clearscreen          #pulire lo schermo
            GOSUB stats                #mostrare la linea delle
                                       statistiche e il nome del
                                       giocatore
            RETURN


960  REM START
970  GOSUB 1250:GOSUB 1050:RETURN
```

La routine "start" pulisce lo schermo e mostra le informazioni basilari sul giocatore — il nome, le vincite, l'importo della scommessa, e così via. Non compare nessun dato riguardante il turno di gioco.

Figura 8.19 — Prepararsi a giocare

```
#Stampare la linea delle statistiche del gioco

gamestats   LL = GL: CC = GC: GOSUB cursor
            PRINT "POINT:"; PT; "THROWS:"; TH;
            RETURN


 980  REM GAMESTATS
 990  LL=GL:CC=GC:GOSUB 1270
1000* PRINT "POINT:";PT;"THROWS:";TH;
1010  RETURN
```

La routine "gamestats" mostra la linea che contiene il punto del giocatore e il numero di tiri già effettuati. "Gamestats" viene chiamata da "game".

Prima di tutto "gamestats" posiziona il cursore, poi scrive i dati. Poichè il valore di PT (il punto) non varia durante una partita, l'unica parte del display che varia è il valore di TH, cioè il numero di tiri.

---

* La versione per l'Apple di questa linea differisce soltanto nella spaziatura nelle costanti di stringa.

Figura 8.20 — Mostrare il punto e i tiri

```
#Mostrare il risultato del tiro — RS$

result      GOSUB clresult          #svuotare la linea del risultato
            PRINT RS$               #mostrare il risultato
            GOSUB delay             #dare al giocatore il tempo di guardare
            GOSUB cleardice         #ritirare i dadi
            RETURN


1020  REM RESULT
1030  GOSUB 1150
1040* PRINT RS$;:GOSUB 1380:GOSUB 2300:RETURN
```

La routine "result" mostra il risultato del tiro. La stringa RS$ viene passata a questa routine dal programma che chiama ("game", "win" o "lose") RS$ dice frasi come: "YOU WIN" (vinci) o "POINT IS 8" (Il punto è 8) o "6 THROW AGAIN" (6 tira un'altra volta). Lo svuotamento della linea del risultato, oltre ad assicurare la completa eliminazione del risultato precedente, fa sì che la linea lampeggi, anche se il risultato attuale è uguale a quello precedente. Il lampeggiare indica la fine del processo di controllo del tiro.

Dopo che il risultato viene scritto sullo schermo, il programma ritarda un momentino (la lunghezza di questo ritardo è determinata nella routine d'inizializzazione specificando un valore per la variabile DC); poi i dadi scompaiono dallo schermo.

Notate come questa routine breve risulta in tre diversi cambiamenti di schermo:

— lo svuotamento della linea del risultato
— il nuovo risultato appare
— la scomparsa dei dadi

Questo gruppo di cambiamenti in sequenza cattura l'attenzione del giocatore e rende il gioco più interessante.

---

* Nelle versioni per l'Apple e per il TRS-80, 2300 viene sostituito da 2580 (vedi Figura 8.8).

Figura 8.21 — Mostrare il risultato del tiro

```
#Mostrare le statistiche del giocatore

stats       GOSUB clstat                          #svuotare la parte delle
                                                  statistiche
            IF WN < 0 THEN                        #dire WON (vinto) o
                                                  LOST (perso)
              PRINT "LOST",
            else
                PRINT "WON",
            PRINT "BET", "THIS RUN", "BEST RUN"   #mostrare il resto
            PRINT ABS(WN), BT, PS, BR; "   ";     #mostrare i valori
            IF CP > 0 THEN{                       #se c'è mostrare il no-
                                                  me del giocatore
              CC = NC: LL = NL: GOSUB cursor
              PRINT NM$(CP);
              }
            RETURN
```

```
1050  REM STATS
1060  GOSUB 1210
1070* IF WN < 0 THEN PRINT "LOST",:GOTO 1090
1080* PRINT "WON",
1090* PRINT "BET", "THIS RUN", "BEST RUN"
1100* PRINT ABS(WN),BT,PS,BR;"   ";
1110  IF CP=0 THEN 1140
1120  CC=NC:LL=NL:GOSUB 1270
1130  PRINT NM$(CP);
1140  RETURN
```

La routine "stats" mostra le statistiche relative al giocatore — l'importo che ha vinto o perso, la scommessa, il numero di vincite consecutive e il numero massimo di vincite consecutive ottenuto da questo giocatore. I valori stampati provengono dalle variabili globali WN, BT, PS, BR: queste variabili vengono riempite con dati provenienti dalle caselle relative al giocatore negli array delle statistiche dalla routine "pass" non appena arriva di nuovo il turno di questo giocatore.

Il fatto che il programma stampi WIN o LOST, a secondo se WN è positiva o negativa, è sembrato migliore di "WINNINGS: — 100" quando il giocatore sta perdendo.

---

* Nella versione di questa routine per l'Apple, la spaziatura dei quattro dati statistici e le loro definizioni viene effettuata per mezzo di TAB(10), TAB(20) e TAB(30) nella lista PRINT, invece che con delle virgole. La ragione di questa diversità è che l'uso delle virgole in dichiarazioni nell'Apple BASIC PRINT dà luogo a soltanto tre voci per linea e non quattro.

Figura 8.22 — Mostrare le statistiche del giocatore

```
#Svuotare per ricevere l'input della scommessa

clearbet    GOSUB start                          #display appare sullo
                                                 schermo
            CC = BC: LL = BL: GOSUB cursor       #posizionare il cursore
            RETURN

1180  REM CLEARBEAT
1190  GOSUB 960
1200  CC = BC : LL = BL : GOBUS 1270 : RETURN
```

La routine "clearbet" si prepara per l'input della scommessa. Poichè sullo schermo non c'è niente d'importante, "clearbet" lo svuota e rimostra i dati statistici relativi al giocatore, e poi riposiziona il cursore al posto giusto.

Figura 8.23 — Prepararsi per l'input della scommessa

```
#Svuotare la linea del risultato

clresult    CC = RC: LL = RL: GOSUB cursor       #posizionare il cursore
            PRINT RC$;                           #svuotare e riposiziona-
                                                 re
            RETURN

1150  REM CLRESULT
1160  CC=RC:LL=RL:GOSUB 1270
1170  PRINT RC$;:RETURN
```

La routine "clresult" svuota la linea del risultato e riposiziona il cursore per mostrare il nuovo risultato. Il riposizionamento del cursore viene effettuato con l'aggiunta di caratteri che muovono il cursore nella stringa RC$. Questo rende la routine specifica per il Pet, ma la versione per l'Apple e per il TRS-80 è soltanto leggermente diversa. In questi sistemi, RC$ non ha caratteri che possano muovere il cursore, e la prima linea viene ripetuta dopo il PRINT RC$. Poichè la differenza è solo minima, questa routine non viene inclusa con le altre routines specifiche alla fine del programma. Infatti la versione per l'Apple e per il TRS-80 funzionerebbe bene anche sul Pet, senza perdere di efficienza. I comandi in BASIC per l'Apple e per il TRS-80 sono:

```
1150  REM CLRESULT
1160  CC=RC: LL=RL: GOSUB 1270: PRINT RC$;
1170  CC=RC: LL=RL: GOSUB 1270: RETURN
```

Figura 8.24 — Svuotare la linea del risultato

```
#Svuotare le linee dei dati statistici sullo schermo

clstat      CC = 0: LL = SL: GOSUB cursor      #prima colonna della
                                               prima linea dei dati
            PRINT BL$: PRINT BL$;              #svuotare due linee
            CC = 0: LL = SL: GOSUB cursor      #riposizionare
            RETURN

1210 REM CLSTAT
1220 CC=0:LL=SL:GOSUB 1270
1230 PRINT BL$:PRINT BL$;
1240 CC=0:LL=SL:GOSUB 1270:RETURN
```

La routine "clstat" svuota due linee dello schermo (linee SL, SL + 1). Il cursore viene poi posizionato all'inizio della zona vuota, per prepararsi a mostrare i dati statistici riguardanti il giocatore per mezzo della routine "stats".

Figura 8.25 — Svuotare le linee dei dati statistici

```
#Pulire lo schermo

clearscreen  PRINT "clr";
             RETURN

1250 REM CLEARSCREEN
1260 PRINT CHR$(147);:RETURN
```

La routine "clearscreen" pulisce lo schermo. Questa è la versione per il Pet, quelle per l'Apple e per il TRS-80 sono leggermente diverse — la prima linea viene sostituita da HOME nell'Apple o da CLS nel TRS-80. In tutti i tre casi, lo schermo viene pulito, e il cursore si muove e va alla posizione in alto a sinistra (Vedi Figura 2.12).

Figura 8.26 — Pulire lo schermo

```
#Posizionare il cursore alla linea LL, colonna CC — versione per il Pet
cursor      PRINT "home";
            CC = CC mod columns: LL = LL mod lines
            IF CC < > 0 THEN                          #muovere il cursore
                                                      verso destra CC volte
              FOR ZZ = 1 TO CC
                PRINT "right";
                NEXT ZZ
            IF LL < > 0 THEN                          #muovere il cursore
                                                      verso il basso LL volte
              FOR ZZ = 1 TO LL
                PRINT "down";
                NEXT ZZ
            RETURN
1270  REM CURSOR
1280  PRINT CHR$(19);
1290  IF CC<0 THEN CC=CC+40:GOTO 1290
1300  IF CC >39 THEN CC=CC-40:GOTO 1300
1310  IF LL<0 THEN LL=LL+24:GOTO 1310
1320  IF LL>23 THEN LL=LL-24:GOTO 1320
1330  IF CC=0 THEN 1350
1340  FOR ZZ=1 TO CC:PRINT CHR$(29);:NEXT ZZ
1350  IF LL=0 THEN RETURN
1360  FOR ZZ=1 TO LL:PRINT CHR$(17);:NEXT ZZ: RETURN
```

Le versioni per l'Apple e per il TRS-80 differiscono come spiegato nella Figura 4.16.

Figura 8.27 — Posizionare il cursore

```
#Dare al giocatore il tempo di leggere quello che appare sullo schermo
delay       FOR ZZ = 1 TO DC
              NEXT ZZ
            RETURN
1380  REM DELAY
1390  FOR ZZ=1 TO DC:NEXT ZZ:RETURN
```

La routine "delay" passa attraverso un loop di "far niente" tante volte quante sono state specificate in DC, che è stato determinato nella dichiarazione DATA nella routine d'inizializzazione "inizialize". La lunghezza di questo ritardo è determinante per la velocità del gioco. DC va determinato usando esperienza. Un'alternativa più precisa, che però non esiste nell'Apple o nel TRS-80, sarebbe l'uso del TI nel Pet. (Figura 9.36)

Figura 8.28 — Far niente per un po' di tempo

```
#Aggiungere un nuovo giocatore

new         IF NP >= MP THEN {                          #se non c'è spazio
              CC = BC: LL = BL: GOSUB cursor            #  informare il giocatore
              PRINT "NO ROOM...";
              GOSUB delay: RETURN
            }
            NP = NP + 1: GOSUB clearscreen              #un giocatore in più
            INPUT "... NAME, PLEASE"; NM$(NP)           #ottenere il suo nome
            PRINT: PRINT "THANK YOU.";
            IF NP > 1 THEN {                            #se non è il primo gio-
                                                        catore
              PRINT "...FOLLOW";NM$(NP - 1)
              ZP = CP: GOSUB savestats                  #conservare i dati statis-
                                                        tici
              GOSUB zap: ZP = NP: GOSUB savestats       #dare al giocatore dei
                                                        nuovi dati statistici
              ZP = CP: GOSUB getstats                   #ristorare i dati attuali
            }
            else                                        #se è il primo giocatore,
                                                        i dati attuali
              {ZP = 1: GOSUB savestats: CP = 1}         #diventano quelli del
                                                        giocatore 1
            GOSUB delay: GOSUB delay: RETURN            #lasciare che il giocato-
                                                        re legga quello che ap-
                                                        pare sullo schermo

1400  IF NP<MP THEN 1450
1420  CC=BC:LL=BL:GOSUB 1270
1430  PRINT "NO MORE ROOM AT THE TABLE";
1440  GOSUB 1380:RETURN
1450  NP=NP + 1:GOSUB 1250
1470* INPUT "MAY I HAVE YOUR NAME PLEASE";NM$(NP)
1480  PRINT:PRINT "THANK YOU.";
1490  IF NP < = 1 THEN 1540
1500  PRINT "YOU WILL FOLLOW";NM$(NP - 1)
1510  ZP=CP:GOSUB 1890
1520  GOSUB 1860:ZP=NP:GOSUB 1890
1530  ZP=CP:GOSUB 1920
1540  CP=1:ZP=1:GOSUB 1890
1550  GOSUB 1380:GOSUB 1380:RETURN
```

La routine "new" aggiunge un giocatore. Al nuovo giocatore vengono dati dei dati inizializzati e gli viene assegnato un posto nel turno di gioco.

Figura 8.29 — Aggiungere un giocatore

```
#Passare i dadi

pass        IF NP = 0 THEN RETURN                  #se non ci sono altri gio-
                                                   catori, non passare
            ZP = CP: GOSUB savestats               #conservare i dati stati-
                                                   stici del giocatore
            CP = (CP mod NP) + 1                   #preparare per il nuovo
                                                   giocatore
            ZP = CP: GOSUB getstats                #inserire i dati del nuovo
                                                   giocatore
            PS = 0: RETURN                         #iniziare senza vincite

1560  IF NP=0 THEN RETURN
1580  ZP=CP:GOSUB 1890
1590  CP=CP + 1:IF CP > NP THEN CP=1
1600  ZP=CP:GOSUB 1920:PS=0
1610  RETURN
```

La routine "pass" passa i dadi da un giocatore a quello che segue. Viene chiamata automaticamente da "lose", o da un comando esplicito proveniente dal giocatore.

Figura 8.30 — Passare i dadi

```
#Mostrare i nomi e le vincite per ogni giocatore

displayer   GOSUB clearscreen                      #pulire lo schermo
            PRINT "NUMBER", "WON/LOST", "NAME"     #mostrare il titolo
            PRINT
            FOR ZZ = 1 TO NP
              PRINT ZZ, ST(1, ZZ), NM$(ZZ)         #mostrare il numero
              NEXT ZZ                              #  le vincite, il nome
            PRINT: RETURN

1620  GOSUB 1250
1640* PRINT:PRINT "NUMBER","WON/LOST","NAME":PRINT
1650  FOR ZZ=1 TO NP
1660* PRINT ZZ,ST(1,ZZ),NM$(ZZ):NEXT ZZ
1670  PRINT
1680  RETURN
```

Lo scopo principale della routine "displayer" è quello di rivelare i numeri assegnati ai giocatori, così questi si potranno riferire al numero quando usano comandi come Q (ritirarsi).

Figura 8.31 — Assegnare i numeri ai giocatori

```
#Il giocatore si ritira

quit        GOSUB displayer                      #mostrare i nomi e i numeri
            repeat {
              INPUT "...NUMBER, PLEASE"; QP      #chiedere il numero
              IF QP = 0 THEN                     #0 = valvola di scarico
                RETURN
              else IF 0 < QP < = NP THEN         #se esiste,
                break                            #  cancellarlo
              else                               #altrimenti,
                PRINT "PLEASE REFER TO LIST"     #  informare il giocatore
              }
            IF QP = CP THEN                      #se il giocatore
              GOSUB pass                         #passa i dadi
            IF QP < NP THEN                      #se non si cancella l'ultimo
              FOR ZZ = QP TO NP - 1              #  spostare quelli in basso verso l'alto
                FOR YY = 1 TO nstats
                  ST(YY,ZZ) = ST(YY,Z + 1)
                  NEXT YY
                NM$(ZZ) = NM$(ZZ + 1)
                NEXT ZZ
            NP = NP - 1                          #un giocatore in meno
            IF CP > = QP THEN                    #rinumerare il giocatore che sta tirando
              CP = CP - 1
            IF NP = 0 THEN                       #se non ci sono più giocatori
              GOSUB zap                          #  cancellare i dati statistici
            RETURN
```

Questa routine permette che un giocatore venga cancellato dal turno. Vengono usati tutti i normali principi dell'eliminazione (Vedi Figura 7.38).

Figura 8.32 — Ritirarsi dalla partita

```
1690  REM QUIT
1700  GOSUB 1620
1710* INPUT "WHAT IS YOUR NUMBER, PLEASE";QP
1720  IF QP=0 THEN RETURN
1730  IF QP<=0 OR QP>NP THEN PRINT "PLEASE REFER TO THE LI-
      ST":GOTO 1710
1740  IF QP=CP THEN GOSUB 1560
1750  IF QP=NP THEN 1790
1760  FOR ZZ=QP TO NP-1:FOR YY=1 TO 4
1770  ST(YY,ZZ)=ST(YY,ZZ+1):NEXT YY
1780  NM$(ZZ)=NM$(ZZ+1):NEXT ZZ
1790  NP=NP-1:IF CP>=QP THEN CP=CP-1
1800  IF NP=0 THEN GOSUB 1860
1810  RETURN
```

* La versione per l'Apple di questa linea differisce soltanto nella comparsa di un punto interrogativo nella costante di stringa.

Figura 8.32a — BASIC per "Quit"

```
#Scommesse secondarie — matrice

sidebet     CC = BC:LL = BL: GOSUB cursor      #andare alla linea della
                                                scommessa
            PRINT "SIDEBETS NOT...";           #annunciare che non è
                                                in funzione
            GOSUB delay                        #lasciare che il giocato-
                                                re legga l'annuncio
            RETURN

1820  REM SIDEBET
1830  CC=BC:LL=BL:GOSUB 1270
1840  PRINT "SIDEBETS NOT YET IMPLEMENTED";
1850  GOSUB 1380:RETURN
```

Questa matrice informa il giocatore che il comando per le scommesse secondarie non funziona.

Figura 8.33 — Scommesse secondarie

```
#Manipolare l'array dei dati statistici

zap         WN = 0                          #inizializzare le vincite
            BT = 0                          #la scommessa
            BR = 0                          #il punteggio migliore
            OB = 0                          #e la scommessa originale
            RETURN

savestats   ST(1, ZP) = WN                  #conservare i dati statistici del giocatore ZP
            ST(2, ZP) = BT
            ST(3, ZP) = BR
            ST(4, ZP) = OB
            RETURN

getstats    WN = ST(1, ZP)                  #prendere i dati statistici del giocatore ZP
            BT = ST(2, ZP)
            BR = ST(3, ZP)
            OB = ST(4, ZP)
            RETURN

1860  REM ZAP
1870  WN=0:BT=0:BR=0:OB=0
1880  RETURN

1890  REM SAVESTATS
1900  ST(1,ZP)=WN:ST(2,ZP)=BT:ST(3,ZP)=BR:ST(4,ZP)=OB
1910  RETURN

1920  REM GETSTATS
1930  WN=ST(1,ZP):BT=ST(2,ZP):BR=ST(3,ZP):OB=ST(4,ZP)
1940  RETURN
```

Questo gruppo di routines fornisce le possibilità fondamentali necessarie per la manutenzione dell'array dei dati statistici. La routine "zap" inizializza le variabili "funzionanti" dei dati statistici; "savestat" conserva i valori nell'array e "getstats" riprende i detti valori dall'array dove erano stati inseriti.

Figura 8.34 — Manutenzione dell'archivio dei dati statistici

```
#Fissare le costanti e i formati — versione per il Pet
initialize    READ SL: DATA statline                    #posizioni sullo schermo
              READ GL, GC: DATA gameline, gamecol
              READ RL, RC: DATA resultline, resultcol
              READ L1, C1: DATA die 1 line, die 1 col
              READ L2, C2: DATA die 2 line, die 2 col
              READ BL, BC: DATA betline, betcol
              READ NL, NC: DATA nameline, namecol
              READ DC: DATA delayconstant               #limite del loop in "delay" (ritardo)
              READ MX: DATA houselimit                  #scommessa massima permessa

              FOR ZZ = 1 TO 6
                READ D$(ZZ)
                NEXT ZZ
                DATA "die face for 1"                   #tracciato di punti 3 per 3
                DATA "die face  for 2"                  # (compresi
                DATA "die face for 3"                   # i caratteri di movimento
                DATA "die face for 4"                   #  del cursore)
                DATA "die face for 5"
                DATA "die face for 6"
              READ BX$: DATA "box surrounding dots"     #cornice per i dadi
              READ BF$: DATA "blank out box & dots"     #sistema di cancellazione dei dadi
              READ RC$: DATA "blank out result line"    #sistema di cancellazione della linea del risultato
              READ BL$: DATA "blank line"               #sistema di cancellazione dell'intera linea
              WN = 0: PS = 0: BT = 0: BR = 0            #inizializzare le variabili
              NP = 0: CP = 0
              MP = maxplayers
              DIM NM$(maxplayers),                      #nome del giocatore
                ST(nstats, maxplayers)                  #dati statistici relativi al giocatore
              RETURN
```

La routine "inizialize" inizializza le costanti, le variabili e gli arrays usati in tutto il programma. Le stringhe del tracciato di punti nell'array DS, e le cornici dei dadi BX$ e BF$ e le stringhe per cancellare i dadi contengono tutte i caratteri di movimento del cursore e ne sono dipendenti nel Pet. Le routines che usano queste stringhe appaiono alla fine del programma. Le versioni per altri sistemi sono completamente diverse.

Figura 8.35 — Inizializzare

```
1950  REM INITIALIZE
1960  READ SL:DATA 20
1970  READ GL,GC:DATA 16,9
1980  READ RL,RC:DATA 11,10
1990  READ L1,C1:DATA 1,10
2000  READ L2,C2:DATA 3,17
2010  READ BL,BC:DATA 0,3
2020  READ NL,NC:DATA 23,14
2030  READ DC:DATA 200
2040  READ MX:DATA 500
2050  FOR XX=1 TO 6:READ D$(XX):NEXT XX
2060* DATA "(see note)"
2070* DATA "(see note)"
2080* DATA "(see note)"
2090* DATA "(see note)"
2100* DATA "(see note)"
2110* DATA "(see note)"
2120  READ BX$
2130* DATA "(see note)"
2140* READ BF$:DATA "(see note)"
2150  READ C1$:DATA "          "
2160* READ C2$:DATA "(see note)"
2170  RC$=C1$+C2$
2180  READ BL$
2190  DATA "                "
2200  GOSUB 1860: PS=0
2210  NP=0:MP=9:CP=0
2220  DIM NM$(9),ST(4,9)
2230  RETURN
```

Queste sono i comandi in BASIC per la routine d'inizializzazione della Figura 8,35, per la versione del Pet. Le versioni per l'Apple e per il TRS-80 sono illustrate nella Figura 8.36a.

* Le linee 2060-2110 contengono le figure dei due dadi. Ciascuna consiste di 3 linee di puntini e di spazi vuoti. La prima e la seconda linea, in ogni caso, sono seguite da 1 carattere "cursore verso il basso" e 3 "cursore indietro". Dunque ciascuna stringa è composta di 17 caratteri. La linea 2130 contiene la "cornice" dei dadi. La stringa contiene una "linea superiore" di spazi, 3 sottolinee, spazio, 3 linee laterali composte di una linea verticale a destra, 3 spazi, una linea verticale a sinistra e una linea in basso composta di 1 spazio, e 3 sopralinee. La linea superiore di ciascuno di questi lati è seguita da 5 caratteri "cursore all'indietro" e "cursore in basso". La linea inferiore è seguita da 3 caratteri "cursore all'indietro" e 3 "cursore verso l'alto" (lasciando il cursore nell'angolo in alto a sinistra dello schermo). Dunque, la stringa contiene 55 caratteri. La linea 2140 contiene la stringa per cancellare i dadi, che consiste di 5 righe di 5 spazi vuoti ciascuna. Le prime 4 linee sono seguite da 5 caratteri "cursore all'indietro" e 1 "cursore verso il basso". La stringa nella linea 2160 consiste interamente di caratteri "cursore all'indietro". La lunghezza della stringa è la stessa della lunghezza della stringa di spazi vuoti nella linea 2150 (18 caratteri).

Figura 8.36 — BASIC per "inizializing" (l'inizializzazione)

```
1950   REM INITIALIZE
1960*  READ SL: DATA 20
1970*  READ GL,GC: DATA 16,9
1980*  READ RL,RC: DATA 11,10
1990   READ L1,C1: DATA 1,10
2000*  READ L2,C2: DATA 3,17
2010   READ BL,BC: DATA 0,3
2020*  READ NL,NC: DATA 23,14
2030*  READ DC: DATA 400
2040   READ MX: DATA 500
2050   FOR ZZ=0 TO 6: FOR XX=1 TO 5
2060   READ D$(ZZ,XX)
2070   NEXT XX: NEXT ZZ
2080   DATA "    ","    ","    ","    ","    "
2130   DATA "---",":  :",": * :",":  :","---"
2180   DATA "---",":* :",":   :",": *:","---"
2230   DATA "---",":* :",": * :",": *:","---"
2280   DATA "---",":**:",":   :",": *:","---"
2330   DATA "---",":**:",": * :",":**:","---"
2380   DATA "---",":**:",":* *:",":**:","---"
2440   READ RC$: DATA "                "
2450** READ BL$ DATA "                "
2460   GOSUB 1860: PS=0
2470   NP=0:MP=9:CP=0
2480   DIM NM$(9),ST(4,9)
2490   RETURN
```

Questa è la lista in BASIC per l'Apple e per il TRS-80 per Craps. La ragione per il vuoto tra i numeri delle linee tra 2080 e 2440 è che in una versione precedente del programma, 2080, 2130, 2180, 2230, 2280, 2330 e 2380 sono state sostituite da cinque linee separate.

---

* Nelle versioni per il TRS-80 di queste linee, il valore SL è 13, il valore GL è 11, il valore RL è 10, il valore RC è 22, il valore C2 è 25, il valore NL è 15, il valore NC è 26 e il valore DC è 100.

** Nella versione per il TRS-80 di questa linea, compaiono 63 spazi vuoti, poichè lo schermo ha 64 colonne.

Figura 8.36a — Versioni di inizializzazione per l'Apple e il TRS-80

mostrato sullo schermo invece del numero delle vincite e delle perdite. Poi, quando un giocatore decide di uscire dal gioco, può incassare i gettoni e accreditarli verso il costo originale.

- Aggiungere degli altri dati statistici, per esempio il tiro migliore di uno qualsiasi dei giocatori, non solo di quello a cui tocca tirare, il numero maggiore dei tiri che è stato necessario per avere o una vincita o una perdita (per tutti i giocatori), le perdite o le vincite più alte, e qualsiasi altri dati che vi possano interessare.
- Aggiungere altre possibilità di scommessa, comprese delle strategie "automatiche" come il raddoppiamento.
- Mostrare sullo schermo il valore finale di tutti i dati statistici riguardanti un giocatore prima di passare alla persona seguente.

Se giocherete a Craps per un certo periodo di tempo, vi verranno in mente certamente delle idee per migliorarlo.

```
#Attendere l'inserimento di un solo carattere e intanto scegliere a caso

onech      repeat{                          #controllare se c'è un
                                            carattere
             GET X$                         #far funzionare il siste-
                                            ma di creazione dei nu-
                                            meri affidato al caso
           ZZ = RND (1)
           } until (X$ < > "")              #finchè un carattere
                                            viene stampato
           RETURN

2340  REM ONECH
2350* GET X$:IF X$< >"" THEN RETURN
2360* ZZ=RND(1):GOTO 2350
```

"Onech" è la routine d'input ad un solo carattere che compare in tutto il libro, ma alla quale vengono aggiunte diverse funzioni RND. Poichè il numero di volte che il loop viene eseguito sarà probabilmente grande ed impossibile da controllare per il giocatore, "onech" assicura un controllo casuale.

Poichè, nell'Apple non è possibile ottenere un completo affidamento al caso, a causa del controllo GET, la versione per l'Apple è illustrata nella Figura 2.11.

Le versioni per l'Apple e per il TRS-80 di questa routine sono alle linee 2660-2680.

* Nelle versioni di queste linee per il TRS-80, GET X$ viene sostituito da X$=INKEY$, e RND(1) da RND(0).

Figura 8.37 — Input ad un solo carattere

SOMMARIO

Craps è un gioco da computer che ha le caratteristiche di quello da casinò. Il suo programma illustra diversi punti che non avevamo toccato nelle discussioni precedenti. Diversi aspetti del gioco sono collegati all'uso indipendente di zone dello schermo.

Questo è possibile grazie al posizionamento del cursore.
Un gioco veloce e scorrevole si ottiene usando input ad un solo carattere, facendo

```
#Disegnare i dadi — versione per il Pet

draw        LL = L1                          #posizionare il cursore
            CC = C1
            GOSUB cursor
            PRINT BX$; D$(D1);               #disegnare il primo dado

            LL = L2                          #posizionare il cursore
            CC = C2
            GOSUB cursor
            PRINT BX$; D$(D2);               #disegnare il secondo dado

            RETURN

2240 REM DRAW
2250 LL=L1:CC=C1:GOSUB 1270
2260 PRINT BX$;D$(D1);
2270 LL=L2:CC=C2:GOSUB 1270
2280 PRINT BX$;D$(D2);
2290 RETURN
```

A causa dell'inserimento di caratteri per muovere il cursore nelle stringhe degli arrays BX$ e D$, la routine "draw" è estremamente semplice. Le versioni per l'Apple e per il TRS-80 sono completamente diverse.

Le versioni per l'Apple e per il TRS-80 usano una versione bi-dimensionale dell'array D$. I comandi in BASIC sono:

```
2500 REM DRAW
2510 FOR LX=L1 TO L1+4
2520 LL=LX:CC=C1:GOSUB 1270
2530 PRINT D$(D1,LX-L1+1);:NEXT LX
2540 FOR LX=L2 TO L2+4
2550 LL=LX:CC=C2:GOSUB 1270
2560 PRINT D$(D2,LX-L2+1);:NEXT LX
2570 RETURN
```

Figura 8.38 — Mostrare i dadi sullo schermo

scomparire automaticamente ed al momento giusto alcuni elementi del display, e cambiando il display ogni volta che si tocca un tasto.

Il passare dei dadi da un giocatore all'altro è un esempio molto semplice di scambio di contesto, un concetto che si rivela importante nei sistemi a scambiamento di tempo ed in quelli a multiprogrammazione.

Il tempo che il giocatore impiega per inserire un input, a causa della sua impredicibilità, viene usato dal programma per dare ai dadi valori impredicibili. Vengono effettuate diverse chiamate alla routine RND nel loop di attesa dell'input.

È ovviamente possibile apportare diversi cambiamenti e migliorie al gioco, specialmente l'aggiunta di altre possibilità di scommessa e un conteggio più dettagliato.

```
#Ritirare i dadi — Versione per il Pet

cleardice    CC = C1
             LL = L1
             GOSUB cursor
             PRINT BF$;
             CC = C2
             LL = L2
             GOSUB cursor
             PRINT BF$;
             RETURN

2300 REM CLEARDICE
2310 CC=C1:LL=L1:GOSUB 1270:PRINT BF$;
2320 CC=C2:LL=L2:GOSUB 1270:PRINT BF$;
2330 RETURN
```

La routine "cleardice" cancella i dadi dallo schermo che erano stati disegnati da "draw". Entrambe queste routines sono tipiche del Pet. Le versioni per l'Apple e per il TRS-80 sono completamente diverse. Ecco i comandi in BASIC:

```
2580 REM CLEARDICE
2590 FOR LX=L1 TO L1 + 4
2600 LL=LX:CC=C1:GOSUB 1270
2610 PRINT D$(0,LX-L1+1);:NEXT LX
2620 FOR LX=L2 TO L2 + 4
2630 LL=LX:CC=C2:GOSUB 1270
2640 PRINT D$(0,LX-L2+1);:NEXT LX
2650 RETURN
```

La funzione di "cleardice" è estremamente importante. I dadi vengono cancellati dallo schermo senza disturbare il resto del quadro. Questo contribuisce ad ottenere uno svolgimento rapido ed interessante della partita.

Figura 8.39 — Ritirare i dadi

CAPITOLO 9

ESSERI SPAZIALI

Il gioco degli Esseri Spaziali è un composto derivato da due giochi: Incontri Spaziali, che è un gioco sviluppato dall'Autore, e il Gioco della Vita, che è un gioco a orientamento grafico che è stato più volte sviluppato per uso su di un computer. Il gioco degli Esseri Spaziali nasce dunque dall'innesto del Gioco della Vita su quello degli Incontri Spaziali.

INCONTRI SPAZIALI

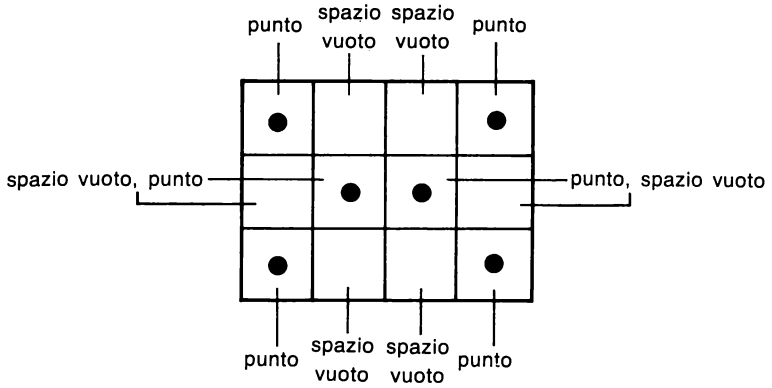
Lo scopo del gioco è quello di inviare dei messaggi ad esseri che vivono a diversi anni luce di distanza dalla Terra. I messaggi da voi creati saranno delle sequenze di zero e di uno che codificano delle figure. Il programma vi aiuta a creare delle figure e si occupa della codifica.

Nel libro "Intelligent Life in the Universe" di I.S. Shklovskii e Carl Sagan, si suggerisce che la nostra tecnologia radio è in grado di inviare delle sequenze di zero e di uno attraverso distanze interstellari. Dicendo zero e uno in termini di trasmissioni radio intendiamo punti e linee come quelli usati nell'Alfabeto Morse.

Nella Figura 9.1 vediamo come una semplice figura matrice di 3 puntini per 4 può essere codificata in una sequenza di 12 puntini e linee.

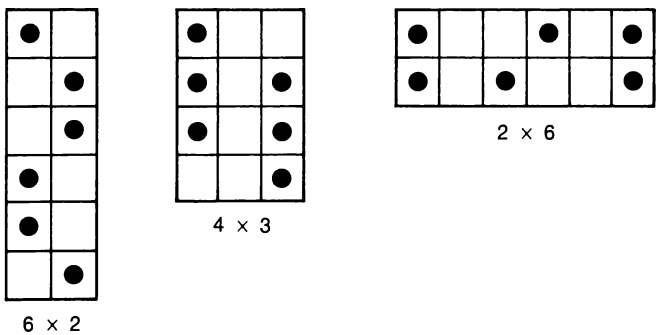
Sfortunatamente, questo tipo di codifica si basa sulla presunzione che il ricevente sappia che si vuole rappresentare una figura di 3 × 4 puntini. Nella Figura 9.2 vediamo tre interpretazioni alternative bidimensionali dello stesso gruppo di punti e linee.

E nella Figura 9.3 vediamo una delle tre possibili interpretazioni tridimensionali. Poichè gli esseri spaziali ai quali indirizziamo le nostre trasmissioni non conoscono certamente le nostre intenzioni, dobbiamo minimizzare il numero di interpretazioni alternative che si possono dare alle nostre trasmissioni. Possiamo farlo trasmettendo delle sequenze di N zero e uno (N bits, in breve), dove N sia il prodotto di due numeri



La sequenza di dodici "punti" e "spazi vuoti" corrispondono ai puntini e agli spazi vuoti in una figura con matrice di puntini 3 x 4. La sequenza rappresenta una trasmissione radio composta da due tipi diversi di segnali (come il Codice Morse). La trasmissione viene inviata ripetutamente per consentire un'identificazione dei dodici "punti" e "spazi vuoti".

Figura 9.1 — Trasmissione di una semplice figura



Queste sono tre interpretazioni alternative della figura illustrata nella Figura 9.1. Se colui che ricevesse la trasmissione non sapesse che rappresenta una matrice di punti 3 x 4, tre diverse interpretazioni.

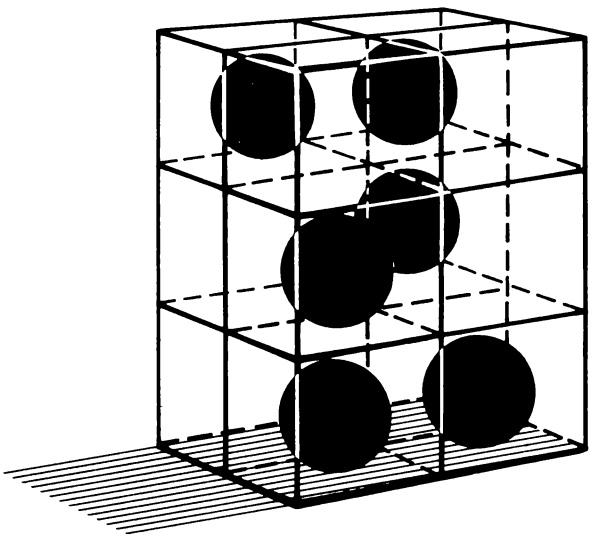
Figura 9.2 — Altre interpretazioni della semplice figura

primi*. Se N è il prodotto di due numeri primi, allora vi sono soltanto due interpretazioni possibili di una matrice di punti che derivi da una sequenza di N bits.

Una figura di una matrice di 29 × 19 puntini ideata nel 1961 da Frank Drake compare nel libro di Shklovskii e Sagan di cui abbiamo parlato in precedenza. Nella Figura 9.4 vediamo una matrice di 13 × 11 puntini che è un adattamento di quella di 29 × 19.

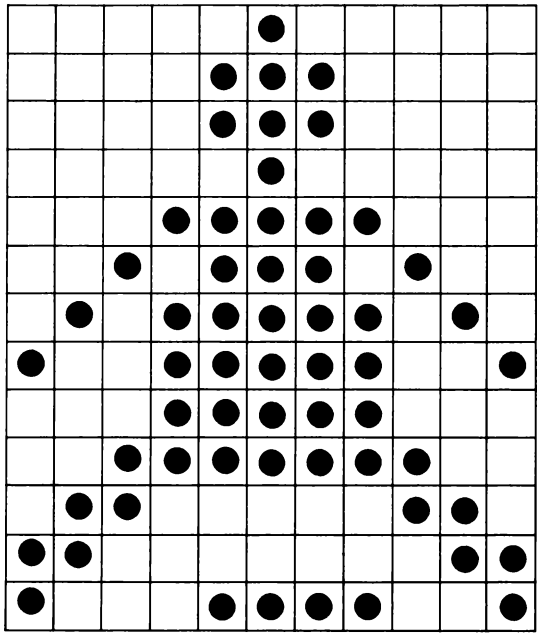
Rappresenta l'autoritratto di un essere spaziale, e comprende una sequenza di quattro puntini ai quali l'essere spaziale dovrà riferirsi in futuro quando parla di sè stesso in ogni comunicazione futura. La Figura 9.5 mostra l'unica altra interpretazione della sequenza di 143 bits illustrata nella Figura 9.4.

* *La definizione di numero primo è un numero intero positivo che non può essere il prodotto di due altri numeri positivi interi. Per esempio, 2, 3, 5 e 7 sono numeri primi, mentre 9 non lo è perchè 9 = 3 × 3. Inoltre, per definizione, 1 non è un numero primo.*



Questa è un'interpretazione tri-dimensionale della trasmissione come una matrice di puntini 3 × 2 × 2.

Figura 9.3 — Un'interpretazione solida della semplice figura



La figura mostra una matrice di puntini 13 x 11, l'autoritratto di un essere spaziale. La riga di quattro puntini alla fine è il "nome" della creatura. In qualsiasi comunicazione futura l'essere spaziale userà i quattro puntini per riferirsi a se stesso. La figura è stata adattata da una matrice complicata 29 x 19 ideata da Frank Drake nel 1961.

Inviato da solo il messaggio sarebbe composto da una sequenza di "uno" e di "zero":

00000100000000011100000001110000
00000100000000111111000010111010 0
01011111010100111111001000111110 00
00111111100011000001101100000001 1
10001111001.

Dunque questa sequenza di "punti" e "spazi vuoti" dove 0 rappresenta un punto e 1 uno spazio vuoto, sarebbe inviata ripetutamente per consentire una trasmissione accurata di tutti i 143 "bits". Poi lo scienziato che riceve deve solo supporre che si tratti di una figura 13 x 11.

Figura 9.4 — Il primo messaggio da un essere spaziale

Forse vi interesserà scambiare questo tipo di messaggi con degli altri giocatori, per vedere se è possibile alla mente umana di comprendere i messaggi e il loro significato. Il programma che presentiamo in questo capitolo non fornisce nessun aiuto per questo tipo di scambi, ma alla fine del capitolo vi diamo suggerimenti e migliorie che potrete apportare al programma stesso.

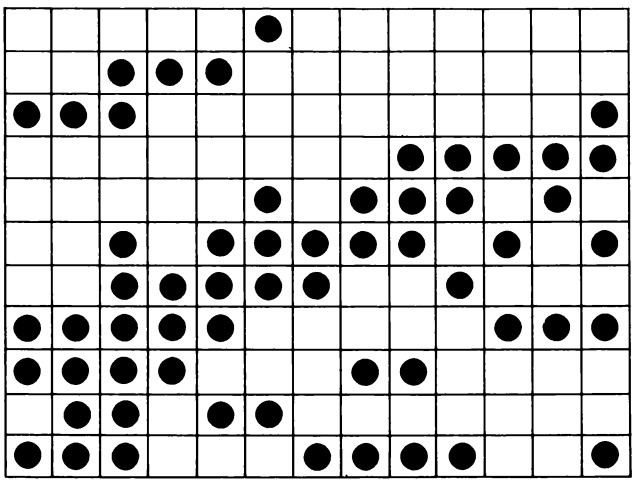
Per iniziare il gioco, il programma richiede le dimensioni del messaggio, questo dialogo si può vedere nella Figura 9.6. Il programma chiede:

LINES, COLUMNS:

e voi rispondete con un paio di numeri primi separati da una virgola. Se per caso inseriste i numeri in un formato non esatto, o se ne inseriste di non permessi, cioè di troppo grandi rispetto al valore massimo permesso, il programma vi ripeterà la domanda.

Se invece inserite due numeri della grandezza giusta ma non primi, il programma chiederà

NON-PRIMES OK?



Questa è la figura che risulta dalla misinterpretazione del messaggio come una matrice 11 × 13 invece che una 13 × 11.

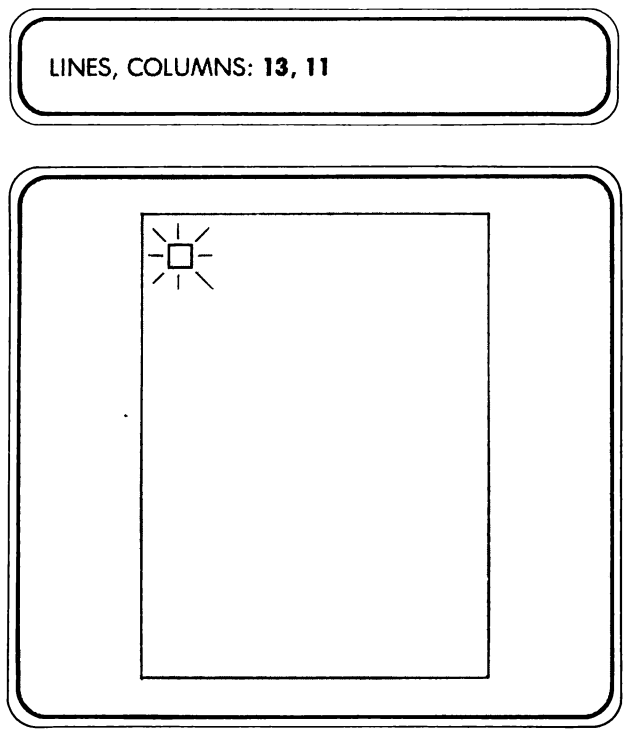
Figura 9.5 — Misinterpretazione del messaggio dell'essere spaziale

Se non volevate usare dei numeri non primi, allora dovrete schiacciare il tasto N, e il programma vi ridomanderà le dimensioni. Se schiacciate un altro tasto, il programma userà queste dimensioni formate da numeri non primi.

Come abbiamo visto nella Figura 9.6, una volta che si siano determinate le dimensioni, il programma disegna una "scatola" interno all'area prescelta per la figura, e inserisce un cursore luminoso intermittente nell'angolo a sinistra in alto. Poi il programma attende che voi inseriate un comando. A parte il comando "E" (uscita) e quello "L" (Gioco della Vita) — di cui parleremo in seguito — gli unici comandi che potrete inserire sono usati prima di tutto per creare e per cancellare i puntini, e poi per muovere il cursore.

Ciascun comando viene eseguito in due fasi. Nella prima una delle tre azioni seguenti ha luogo:

— Un puntino viene inserito alla posizione del cursore.



Questo è il dialogo iniziale degli Incontri Spaziali.

Figura 9.6 — Iniziare gli Incontri Spaziali

— Se c'è un puntino alla posizione del cursone, questo viene cancellato.

— Non si effettua nessun cambiamento al puntino o allo spazio vuoto che si trova alla posizione del cursore.

Nella seconda fase, il cursore viene mosso verso l'alto, il basso, a destra o a sinistra. Questo movimento si effettua verso la posizione seguente nella direzione data, ma con un sistema di "avvolgimento". (Il termine "avvolgimento" viene usato in situazioni in cui la "fine" e l'"inizio" sono limitrofi). Per esempio, se si volesse muovere il cursore verso destra dalla colonna che si trova già all'estrema destra, si verificherà il caso in cui il cursore si trova nella posizione all'estrema sinistra della stessa linea.

Nella Figura 9.7 vediamo i comandi e la relazione che intercorre tra i comandi per il disegno e le posizioni nella tastiera numerica. Se il vostro sistema non ha una tastiera numerica, allora userete degli altri caratteri, Vi spiegheremo come fare quando parleremo del programma degli Esseri Spaziali. In modo da far sì che i comandi siano facili da ricordare, abbiamo usato il metodo seguente:

— Prima di tutto abbiamo scelto un gruppo di quattro tasti per regolare il movimento del cursore per le quattro direzioni. Tenendo schiacciato il tasto SHIFT mentre si schiaccia uno di questi quattro tasti, un puntino apparirà al posto del cursore, prima che questo si muova. Se si schiaccia il tasto senza tenere quello di SHIFT schiacciato, il puntino (o lo spazio bianco) non cambierà posizione e il cursore si muoverà secondo la direzione indicatagli.

— I quattro tasti che comandano il movimento del cursore sono stati scelti perchè posizionati ai quattro angoli di un rombo. Il tasto "in alto" è quello che si trova al vertice superiore, il tasto "in basso" si trova al vertice inferiore, quello "a sinistra" e quello "a destra" si trovano ai rispettivi lati.

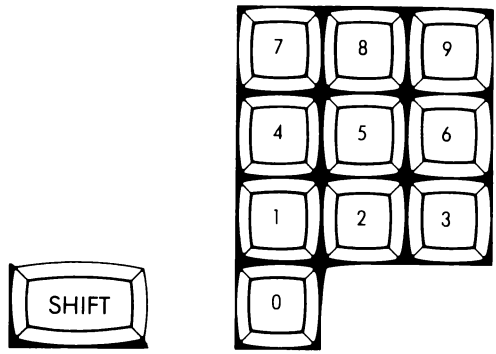
Il primo di questi punti illustra l'ortogonalità, un termine preso dalla teoria degli spazi vettori. In poche parole, le due funzioni di scelta dell'azione e scelta del movimento, sono state assegnate ad "assi coordinate" indipendenti, e qualsiasi azione può essere accoppiata a qualsiasi movimento. Sfortunatamente, l'"asse" che abbiamo scelto per l'azione ha soltanto due "punti": "tasto di SHIFT schiacciato" e "tasto di SHIFT non schiacciato". Dunque non c'è la possibilità di specificare la terza azione, e cioè cancellare un punto, schiacciando un altro tasto mentre si schiaccia uno di quelli che determinano la direzione. (In alcuni piccoli computers, c'è un'asse a tre valori data dai tasti SHIFT e CTRL. I tre valori sono "SHIT schiacciato", "CTRL schiacciato" e "nessuno dei due schiacciato". Se il vostro sistema è uno di questi, potrete cambiare i codici dei comandi per usare questa possibilità di lavoro.) Ora che abbiamo descritto i comandi del disegno, occupiamoci di vedere come funzionano.

Nella Figura 9.8 avete cominciato a disegnare la figura mostrata nella Figura 9.4. Iniziando dalla posizione illustrata nella Figura 9.6 avete usato la sequenza di co-

Comandi per disegnare

8* Cursore verso l'alto
2* Cursore verso il basso
4* Cursore verso sinistra
6* Cursore verso destra
0 Cancellare il puntino, poi muovere il cursore verso sinistra
7 Cancellare il puntino, poi muovere il cursore verso l'alto
5 Cancellare il puntino, poi muovere il cursore verso detra
1 Cancellare il puntino, poi muovere il cursore verso il basso

* Con il tasto SHIFT marcare, poi spostare il cursore come indicato. Senza il tasto SHIFT, muovere soltanto il cursore.



Parte numerica della tastiera

Comandi Vari

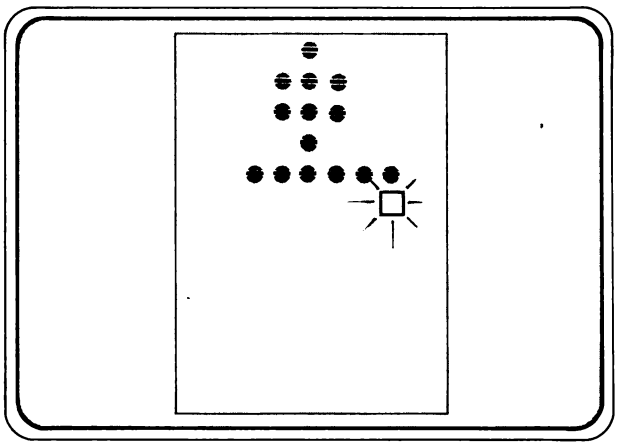
E Exit. Emette un tracciato come sequenza di zero e di uno. (Per uso con sistemi di registrazione esterna, ma nella versione attuale, la sequenza di zero e di uno viene mostrata sullo schermo).
L Life (verrà spiegata in seguito).

Questi sono i comandi che possono essere caricati dal programma Alien Encounter quando il cursore lampeggia. Ciascuno è un singolo carattere.

Figura 9.7 — I comandi degli Incontri Spaziali

mandi illustrata nella Figura 9.8 per eseguire un movimento di andata e ritorno sistematicamente avanti e indietro, completando ciascuna linea di puntini e proseguendo con quella seguente.

Per errore, avete inserito un puntino in più nella quinta fila. Per cancellarlo dovrete usare il comando che muove il cursore in alto di una fila, come illustrato nella Figura 9.9. Dopo aver posizionato il cursore dove c'è il puntino da cancellare, potrete usare uno qualsiasi dei quattro comandi per "cancellare il puntino e muovere il cursore". Il comando "cancellare e muoversi verso il basso" è uno di quelli che più si adatta ai movimenti che avete eseguito fino ad adesso.



6,6,6,6,6,6^S,2,4^S,4^S,2^S,6^S,6^S,
2^S,4,4^S,4,2,6^S,6^S,6^S,6^S,6^S,2^S

Iniziando dalla posizione illustrata nella Figura 9.6, il giocatore ha iniziato a disegnare la figura illustrata nella Figura 9.4. Sulla destra dello schermo compare la sequenza di comandi ad un solo carattere che il giocatore ha inserito. Una s significa "usando il tasto SHIFT".

La sequenza inizia con cinque 6 per spostare il cursore al centro della linea superiore. Poi un 6^S fa apparire il puntino e il cursore si sposta verso destra.

Un 2 muove il cursore alla posizione più a destra della seconda linea. 4^S due volte seguito da 2^S fa apparire la seconda linea di puntini e il cursore si sposta alla posizione più a sinistra della terza linea. I comandi seguenti continuano questo movimento sistematico, ma notate che è stato commesso un errore.

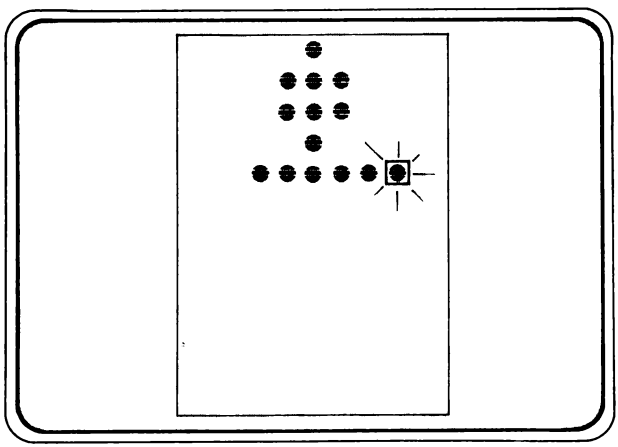
Figura 9.8 — Iniziare a disegnare l'essere spaziale

Nella Figura 9.10 vediamo come si ottiene una versione codificata della figura. Ogni qualvolta il cursore si illumini ad intermittenza, l'inserimento del comando "E" fa apparire sullo schermo la forma codificata della figura. Lo zero corrisponde agli spazi vuoti e l'uno ai puntini.

Nella Figura 9.10 vediamo poi un'altra caratteristica del programma degli Esseri spaziali. Dopo che avete finito di esaminare la versione codificata della figura e avete schiacciato un qualsiasi altro tasto (a parte "Q") il programma chiederà

LINES, COLUMNS:

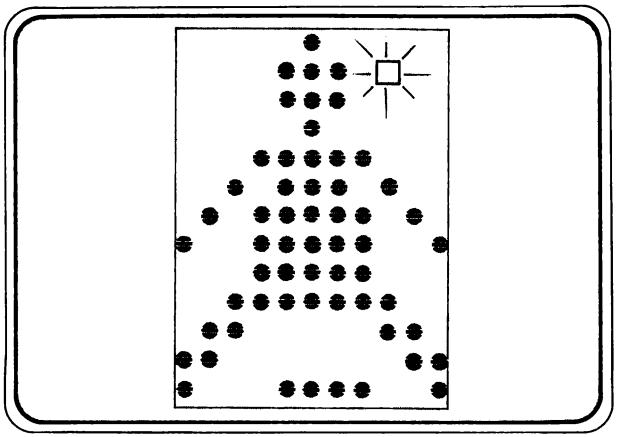
e voi, invece di inserire un nuovo gruppo di dimensioni, schiacciate il tasto RETURN, e la figura su cui stavate lavorando prima è riapparsa sullo schermo. A questo punto certamente vorrete conservare la versione codificata della vostra figura su di una cassetta, su di un disco o su di uno stampato, così da poterla trasmettere ad un altro giocatore. Vorrete certamente inserire delle immagini codificate ricevute da un altro giocatore nel vostro computer. Le funzioni che abbiamo illustrato nella Figura 9.10 sono le azioni delle "matrici" il cui scopo finale è quello di incrementare l'input/output delle figure codificate per le cassette, i dischi o per gli stampati.



Dalla posizione illustrata nella Figura 9.8 il giocatore ha inserito un 8 per riposizionare il cursore all'ultimo puntino che è stato disegnato. Il cursore appare completamente diverso dove copre un puntino.

Ora il giocatore può rimuovere il puntino dove è posizionato il cursore con uno qualsiasi dei comandi 0, 7, 5 o 1. Ogni comando lascierà il cursore in una posizione diversa.

Figura 9.9 — Cancellare un puntino



La figura dell'essere spaziale è completa e il cursore viene spostato ad una posizione vicina all'angolo destro in alto

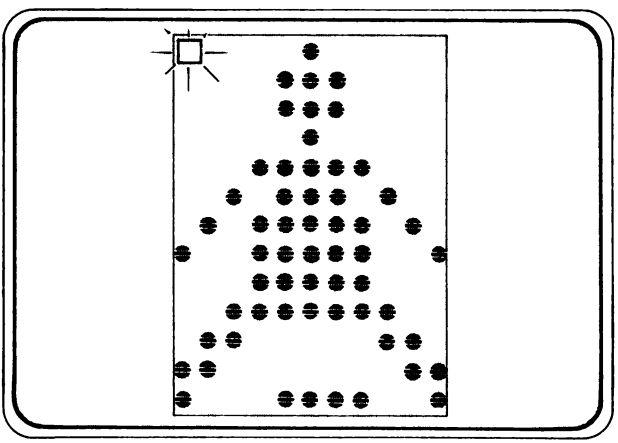
Il giocatore inserisce il comando "E"

```
0000010000000001110000000011100000000010
0000000111110000010111010001011111010100
1111100100011111100000111111100011000011
0110000000111000111100 1
```

Lo schermo si svuota, e la codifica della figura in una sequenza di zero e di uno compare. Il giocatore schiaccia RETURN

```
LINES, COLUMNS:
```

Il programma richiede un'altra dimensione di cornice. Il giocatore schiaccia RETURN per riottenere la figura precedente.



Il programma è pronto a ricevere altri comandi.

Figura 9.10 — Codificare la Figura

LE REGOLE DEL GIOCO DELLA VITA

Il Gioco della Vita è un gioco di cui parleremo soltanto come aggiuntivo a quello degli Incontri Spaziali.

Quando il cursore si illumina ad intermittenza, si può inserire un comando "L". Questo comando fa sì che il programma trasformi l'intera figura secondo le regole del Gioco della Vita.

La trasformazione della figura secondo queste regole necessita del computo dei "vicini" per ciascuna posizione. Nella Figura 9.11 vediamo cosa significa "posizioni vicine". Queste sono infatti le posizioni che si trovano lungo le linee diagonali a 45°, orizzontali o verticali immediatamente accanto ad una determinata posizione. In genere per ogni posizione ve ne sono otto, cinque a tre "vicine", a secondo di dove questa posizione si trova rispetto alle linee perimetrali della zona. Il numero di "vicini" di una posizione è dunque il numero delle posizioni che si trovano immediatamente vicine a quest'ultima. Per esempio, nella Figura 9.10 la posizione del puntino in cima alla testa dell'Essere Spaziale (posizione 0,5) ha tre vicini: i tre puntini in basso (posizioni (1,4), (1,5) e (1,6)). Le posizioni alla destra e alla sinistra del puntino in alto hanno anche loro tre "vicini". I vicini della posizione (0,4) sono (1,4), (0,5) e (1,5) e i vicini di (0,6) sono (0,5), (1,5) e (1,6). La posizione immediatamente sotto a quella in cima (1,5) nel mezzo della testa dell'Essere Spaziale, ha sei vicini (0,5), (1,4), (1,6), (2,4), (2,5) e (2,6). I puntini che si trovano alla fine delle due braccia dell'Essere Spaziale hanno solo un vicino ciascuno.

Nella Figura 9.12 vediamo le regole di trasformazione del Gioco della Vita. Nella Figura 9.13 vediamo come applicare queste regole ed alcuni esempi pratici. Non abbiamo bisogno di studiare a fondo tutte le situazioni che possono capitare quando si gioca, questo è stato fatto in diversi altri punti, ed è solo marginalmente importante per quello di cui veramente vogliamo parlare.

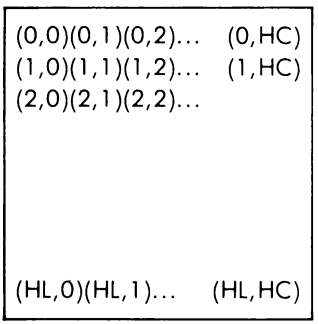
Ora consideriamo l'effetto che il comando "L" ha sull'autoritratto dell'Essere Spaziale. Nella Figura 9.14 vediamo i diversi stadi attraverso i quali il ritratto passa prima di raggiungere una configurazione costante. Forse l'Essere voleva inviare delle altre informazioni che lo riguardavano usando i diversi stadi di configurazione.

Le possibilità certamente mostrano quali dimensioni in più il Gioco della Vita ha apportato a quello degli Incontri Spaziali.

Se avete usato degli altri programmi del Gioco della Vita, certamente saprete che le capacità di disegno del gioco degli Incontri Spaziali e la possibilità di cambiare le sequenze tra due trasformazioni non si trovano normalmente in altri programmi. Inoltre lo sviluppo delle possibilità d'archivio e di ripresa delle figure come sequenze di bits su sistemi di archivio esterni che esistono nel Gioco degli Esseri Spaziali non si trovano in programmi del Gioco della Vita.

IL PROGRAMMA DEL GIOCO DEGLI ESSERI SPAZIALI

Ora che abbiamo descritto il gioco degli Esseri Spaziali, diamo un'occhiata al suo programma. Questo, illustrato nelle Figure 9.15 e fino alla 9.44, è simile in diversi punti ad altri che abbiamo già spiegato in questo libro. Ci occuperemo, durante la



(1,1) ha otto posizioni vicine:

(0,0), (0,1), (0,2), (1,0), (1,2), (2,0), (2,1), (2,2).

(1,0) ha cinque posizioni vicine:

(0,0), (0,1), (1,1), (2,0), (2,1).

(0,0) ha solo tre posizioni vicine:

(0,1), (1,0), (1,1).

Le posizioni della figura sono state etichettate usando i numeri di linea e di colonna in modo da rendere l'idea di "posizioni vicine". In genere, una posizione "lontana dai lati" ha otto posizioni vicine; una posizione "lungo i lati" ma "lontano dagli angoli" ha cinque posizioni vicine. Le posizioni "agli angoli" hanno tre posizioni vicine. Nei casi particolari in cui l'area della figura ha solo una linea o una colonna, i numeri delle posizioni vicine sono uno o due.

Figura 9.11 — Posizioni vicine

Numero dei vicini	Nuovo contenuto
Minore di 2	Nessun puntino
2	Il vecchio contenuto rimane inalterato
3	Puntino
Maggiore di 3	Nessun puntino

Queste sono le regole che determinano la trasformazione della figura quando viene inserito il comando "L" (Gioco della Vita). Il programma prima di tutto calcola il numero di posizioni vicine per ciascuna posizione dell'area della figura, poi determina il nuovo contenuto di ciascuna posizione sulla base del vecchio contenuto e del numero dei vicini, usando le regole di cui sopra.

Figura 9.12 — Regole del Gioco della Vita

Pagina mancante

Pagina mancante

spiegazione, in particolare dei comandi e delle figure. Discuteremo dunque la programmazione che è necessaria per ottenere il riquadro, parleremo dei puntini che compongono la figura e del cursore. Parlando dei comandi ne descriveremo la codifica, e la possibilità di inserimento di nuovi comandi.

```
#Incontri Spaziali e il Gioco della Vita

        GOSUB init                          #inizializzare gli arrays
                                            e le stringhe
        repeat{
          GOSUB setframe                    #determinare le dimen-
                                            sioni di una nuova corni-
                                            ce
          repeat{
            GOSUB drawcmd                   #convertire l'input in
            IF C1 <>0 THEN                  # C1, C2; C1 = 0 signi-
                                            fica fatto
              ON C1 GOSUB                   #eseguire il comando
                                            per disegnare
                space,                      # muovere il cursore
                mark,                       # marcare un punto (e
                                            muovere il cursore)
                unmark,                     # rimuovere il segno (e
                                            muovere il cursore)
                miscmd                      # altri comandi
            }until (C1 = 0)
          GOSUB outstream                   #mostrare la figura in
                                            zero e uno
          GOSUB onech                       #segnalare di iniziare
                                            una nuova cornice
        }until (X$ = "Q")                   #Q = exit
        END

100  GOSUB1190
110  GOSUB160
120  GOSUB700:IFC1=0THEN140
130  ONC1GOSUB720,730,740,750:GOTO120
140  GOSUB560:GOSUB990:IFX$<>"Q"THEN110
150  END
```

Questa è la routine principale del programma degli Esseri Spaziali, la struttura di questa routine è simile a quella di altre routines principali contenute in questo libro.

Figura 9.15 — Esseri Spaziali

```
#Preparare la cornice

setframe    GOSUB clearscreen              #iniziare con lo schermo vuoto
            GOSUB primes                   #domandare le dimensioni della figura
            IF PX = -1 THEN {              #PX = -1 per registrare il tracciato esistente
              GOSUB blockin                #ottenere il tracciato
              GOSUB choices                #ottenere P1, P2, HL, HC.
              }
            else {
              HL = P1 - 1                  #numero di linea più alto
              HC = P2 - 1                  #numero di colonna più alto
              GOSUB cleardot               #pulire le parti HL x HC di D
              }
            GOSUB center                   #calcolare LZ e CZ per l'angolo a sinistra più in basso
            GOSUB clearscreen              #pulire lo schermo
            GOSUB drawframe                #disegnare una cornice
            GOSUB drawdots                 #disegnare in puntini (dal tracciato "blockin")
            CC = 0: LL = 0                 #fissare il cursore
            GOSUB newcursor
            RETURN

160 GOSUB980:GOSUB200:IF PX<>-1 THEN180
170 GOSUB540:GOSUB550:GOTO190
180 HL = P1 - 1 : HC = PZ - 1 : GOSUB 600
190 GOSUB610:GOSUB980:GOSUB620:GOSUB660:CC = 0:LL = 0:GO-
    SUB840:RETURN
```

La routine "setframe" ottiene le dimensioni della cornice e la disegna al centro dello schermo. Può anche accettare "block input" cioè un input diretto di una figura composto di zero e uno. Questo input può provenire dalla tastiera o da un sistema di registrazione e archivio esterno su cui era stato registrato.

Figura 9.16 — Preparare la cornice

```
#Ottenere le dimensioni della figura — ritornare PX = — 1 per il block input

primes      repeat {
              PRINT "LINES, COLUMNS:";          #chiedere le dimensioni
              GOSUB stringin                    #ottenere una risposta
              IF XX$ = "" THEN                  #RETURN segnala un
                                                block input
                {PX = — 1: RETURN}
              else{
                GOSUB getprimes                 #analizzare la stringa
                                                d'input
                IF PX = 0 THEN                  #sintassi non corretta
                  XX = — 1
                else{
                  GOSUB checkprimes             #controllare i numeri
                  IF XX = 1 THEN{               #se non è numero primo
                    GOSUB askok                 # domandare se va be-
                                                ne lo stesso
                    IF OK = 1 THEN
                    XX = 0
                    }
                  }
                }
              }until (XX = 0)
            P1 = PX: P2 = PY                    #PX, PY diventano linee
                                                e colonne
            RETURN
```

```
200 PRINT"LINES,COLUMNS: ";:GOSUB1010:IFXX$ < >""THEN220
210 PX=—1:RETURN
220 GOSUB310:IFPX < >0THEN240
230 XX=—1:GOTO200
240 GOSUB420:IFXX < >1THEN270
250 GOSUB290:IFOK < >1THEN270
260 XX=0
270 IFXX < >0THEN200
280 P1=PX:P2=PY:RETURN
```

La routine "primes" interagisce con il giocatore per ottenere le dimensioni della cornice della figura. Per il gioco degli Incontri Spaziali, le dimensioni devono essere dei numeri primi, altrimenti la routine richiede conferma da parte del giocatore prima di accettarli.

Figura 9.17 — Ottenere le dimensioni della figura

La parte di schermo su cui appare la figura

Nella parte di schermo su cui appare la figura vi sono tre parti distinte:

- il riquadro
- i puntini che formano la figura
- il cursore.

Il riquadro viene disegnato una volta soltanto, da routines che vengono chiamate da "setframe" (Figura 9.16). La routine "center" (Figura 9.25) determina la posizione del punto che si trova sulla linea zero, colonna zero, cosicchè il riquadro venga centrato sullo schermo. La routine "drawframe" (Figura 9.26) disegna un riquadro rettangolare intorno alla parte di schermo su cui apparirà la figura. La chiamata a "drawframe" in "setframe" viene preceduta da una chiamata a "clearscreen", così da pulire lo schermo per chiamare "drawframe".

I puntini sono l'immagine dell'array D. La routine "drawdots" (Figura 9.27) lo garantisce inserendo un puntino a ciascuna posizione di linea-e-colonna (LL,CC) per cui D(LL,CC) <> 0, ed uno spazio bianco in ogni posizione (LL,CC) per cui D(LL,CC) = 0. L'ordine in cui questa operazione viene effettuata avrà un effetto anche sullo schermo. Per esempio, quando si usa il comando "L" per ottenere la trasformazione in Gioco della Vita della figura, questa cambierà una linea per volta.

```
#Domandare se va bene anche se i numeri non sono primi – risposta 0 = no 1 = sì

askok       PRINT "NON-PRIMES OK?";              #porre la domanda
            GOSUB onech                          #ottenere la risposta
            IF X$ = "N" THEN
              OK = 0                             #"N" = no
            else
              OK = 1                             #qualsiasi altro carattere = sì
            RETURN

290 PRINT"NON-PRIMES OK?";:GOSUB990:OK=1:IFX$ = "N" THENOK
    = 0
300 RETURN
```

La routine "askok" chiede al giocatore se va bene anche se le dimensioni che sono state inserite non sono numeri primi. Se c'è stato un errore da parte del giocatore, una risposta di "N" farà richiedere le dimensioni.

Figura 9.18 — Domandare conferma per un input composto di numeri non primi

```
#Ottenere dimensioni da XX$ — PX = 0 se c'è errore interrompere
getprimes   LL = LEN(XX$)                          #lunghezza dell'input
            IF LL < 3 THEN                         #troppo breve
              errorexit
            else {
              NP = 0                               #nessun numero primo
                                                   per adesso
              FOR SX = 1 TO LL — 1
                X$ = MID$(XX$,SX,1)                #il prossimo carattere di
                                                   XX$
                GOSUB sepcheck                     #separa?
                IF SP = 0 THEN                     #no
                  NP = 1                           #segnalare "per primo"
                else {                             #si
                  IF NP = 0 THEN                   #prima del primo nume-
                                                   ro primo
                    IF X$ < > " " THEN             #spazio portante OK
                      errorexit
                  else
                    PX = VAL(LEFT$(XX$,SX—1))      #primo numero primo
                    PY = VAL(MID$(XX$,SX+1))       #secondo numero primo
                    RETURN
                    }
                    }
                NEXT SX
              errorexit                            #non ne ho trovato una
              }                                    coppia
```

```
310 LL=LEN(XX$):IFLL>=3THEN330
320 PX=0:RETURN
330 NP=0:FORSX=1TOLL—1:X$=MID$(XX$,SX,1):GOSUB400:IFSP
    <> 0THEN350
340 NP=1:GOTO390
350 IFNP<>0THEN380
360 IFX$=" "THEN390
370 PX=0:RETURN
380 PX = VAL(LEFT$(XX$,SX — 1) ):PY = VAL(MID$(XX$,SX + 1) ):RE-
    TURN
390 NEXTSX:PX=0:RETURN
```

La routine "getprimes" estrae una coppia di dimensioni dalla stringa XX$ ritornata dalla routine "stringin". Il formato dell'input deve essere: numero, separatore, numero, dove per separatore si intende o una virgola o uno spazio vuoto. Per esempio "7,5" e "7 5" sono entrambi formati accettabili.

Figura 9.19 — Analizzare l'input delle dimensioni

Un approccio alternativo sarebbe quello di svuotare l'intera parte dello schermo su cui appare la figura, e poi disegnarci i puntini. Un vantaggio sarebbe la riduzione del tempo necessario per il disegno della figura, e questo cambiamento si può fare con una semplice modifica della routine "drawdots". La scelta tra le due alternative è soltanto questione di preferenza personale.

La routine "drawdots" viene chiamata inizialmente da "setframe" (Figura 9.16). Poi viene chiamata soltanto dalla routine "life" (Figura 9.44). Le routines che implementano i comandi per il disegno non fanno che mostrare — o cancellare — un puntino per volta usando il controllo del cursore, piuttosto che ridisegnando l'intera figura. Le routines di disegno che cancellano o che disegnano i puntini sono illustrate nella Figura 9.32. Queste poi chiamano le subroutines "showdot" e "zapdot" (Figura 9.27) per il cambiamento sullo schermo.

Il cursore è un elemento completamente diverso dal cursore CRT che è controllato dal vostro computer ed è usato per guidare il vostro input di programmi e di dati.

Questo cursore è controllato interamente dal programma degli Esseri Spaziali. Questo viene effettuato controllando:

— la locazione del cursore della figura

— la scelta del carattere del cursore

— l'illuminazione intermittente

— il ritorno del contenuto dello schermo quando si leva il cursore.

La locazione del cursore della figura viene controllata dalle variabili della linea e della colonna LL e CC. Queste vengono fissate a zero nella routine "setframe" (Figu-

```
#Controllare per trovare il separatore

sepcheck    IF X$ = "" OR X$ = "," THEN
              SP = 1
            else
              SP = 0
            RETURN

400 SP=0:IFX$=""ORX$=","THENSP=1
410 RETURN
```

La routine "sepcheck" esamina un carattere per determinare se questo sia o meno un separatore. I caratteri separatori riconosciuti da questa routine sono la virgola e lo spazio, ma si potrebbero aggiungere altri se si volesse.

Figura 9.20 — Controllare per trovare il separatore

ra 9.16) e la routine "life" (Figura 9.44), così da piazzare il cursore nell'angolo in alto a sinistra del quadro. I comandi di disegno lo fanno poi muovere. Le routines illustrate nella Figura 9.33 effettuano i cambiamenti sulle variabili coordinate LL e CC. Il movimento vero e proprio sullo schermo avviene nella routine "cursin" (Figura 9.36).

La routine "cursin" viene chiamata da "drawcmd" (Figura 9.28) in modo da ottenere il comando seguente dal giocatore. "Cursin" prima di tutto sceglie il carattere del cursore chiamando "newcursor" (Figura 9.32). Poi esegue un loop nel quale mostra alternativamente il carattere del cursore che ha scelto e un carattere vuoto mentre aspetta che arrivi un input dalla tastiera. Questo fa sì che appaia come un'illuminazione intermittente del cursore. La frequenza dell'intermittenza viene determinata dalla variabile FQ, che è inserita nella routine "init" (Figura 9.38).

```
#Controllare PX, PY – return XX = –1 per rigetto, 0 per numeri primi, 1 per non numeri non primi

checkprimes IF PX > ML OR PY > MC THEN                    #al di fuori della gamma
              XX = – 1
            else {
              PP = PX: GOSUB primetest
              IF PR = 0 THEN
                XX = 1
              else {
                PP = PY: GOSUB primetest
                IF PR = 0 THEN
                  XX = 1
                else
                  XX = 0
              } }
            RETURN

420 IFPX<=MLANDPY<=MCTHEN440
430 XX=–1:RETURN
440 PP=PX:GOSUB480:IFPR<>0THEN460
450 XX=1:RETURN
460 PP=PY:GOSUB480:XX=0:IFPR=0THENXX=1
470 RETURN
```

La routine "checkprimes" esamina i numeri passatigli nelle variabili PX e PY, e da un'indicazione nella variabile XX. XX è fissata a –1 se uno dei numeri supera il numero massimo permesso, a 0 se entrambi i numeri sono numeri primi, o a 1 se uno dei numeri non è primo.

Questo controllo viene effettuato dalla routine "primtest" che viene chiamata una volta per ciascun numero.

Figura 9.21 — Controllare che le dimensioni siano numeri primi

La routine "cursin" illustrata nella Figura 9.36 usa la possibilità Pet TI per controllare l'alternanza. Per questa ragione, la variabile FQ viene specificata in "jiffies" (cioè sessantesimi di secondo), le unità che usa l'orologio del Pet TI. Per quanto riguarda la versione per il TRS–80, il loop "repat...until" viene sostituito con un loop "FOR...NEXT" che usa dei limiti

```
1 TO FQ*TN
```

dove TN ha un valore empiricamente determinato che risulta nell'esecuzione del loop in FQ jiffies. Inoltre, la versione per il TRS–80 usa il comando

```
X$ = INKEY$
```

laddove la versione per il Pet usa

```
GET X$
```

```
#Controllare PP e ritornare PR = 1 se PP è primo, PR = 0 se non lo è

primetest   IF PP = 2 OR PP = 3 OR PP = 5 THEN
              primereturn
            else IF PP < 7 OR PP is even THEN
            notprimereturn
            else {
              FOR I = 3 TO INT(SQR(PP) ) STEP 2      #provare dei divisori
                                                     possibili
                IF I divides PP THEN
                  notprimereturn
                NEXT I
              primereturn
              }

480  IFPP=2ORPP=3ORPP=5THENPR=1:RETURN
490  IFPP<7ORPP/2=INT(PP/2)THENPR=0:RETURN
500  FOR I = 3 TOINT (SQR(PP) ) STEP Z
510  IFPP/I < >INT(PP/I)THEN530
520  PR=0:RETURN
530  NEXTI:PR=1:RETURN
```

La routine "primetest" esamina il numero che è stato dato nella variabile PP e ritorna un'indicazione nella variabile PR. PR è fissata a 1 se il valore di PP è primo, a 0 se non lo è.

Figura 9.22 — Controllo di PP

```
#Matrice per block input

blockin     RETURN                          #mantenere la figura attuale

#Matrice per la presentazione di
gruppi di numeri primi
choices     RETURN                          #mantenere le dimensioni attuali

#Matrice per l'output del flusso
dei bits componenti il messaggio
outstream   GOSUB clearscreen
            FOR LL = 0 TO HL
              FOR CC = 0 TO HC
                IF D(LL,CC) = 1 THEN
                  PRINT "1";
                else
                  PRINT "0";
                NEXT CC
              NEXT LL
            RETURN

540 RETURN
550 RETURN
560 GOSUB980:FORLL = 0TOHL:FORCC = 0TOHC:IFD(LL,CC) <>
    1THEN580
570 PRINT"1";:GOTO590
580 PRINT"0";
590 NEXTCC:NEXTLL:RETURN
```

La routine "blockin" è una matrice che ritorna senza fare niente. L'effetto è che è possibile richiamare il tracciato precedente. L'uso previsto per "blockin" è quello di poter registrare una stringa di zero e di uno che precedentemente erano registrati su un sistema esterno o che sono stati inseriti dal giocatore per mezzo della tastiera.

La routine "choices" è una matrice da usare con "blockin". La sua inattività permette, accoppiata a quella di "blockin" di richiamare il tracciato precedente. La routine "outstream" è una matrice che stampa una sequenza di uno e di zero corrispondente alla figura. Lo scopo di "outstream" è quello di conservare la sequenza di zero e di uno su di un sistema di registrazione esterno (disco, cassetta) da usare poi con "blockin".

Figura 9.23 — Matrici di registrazione su sistemi esterni per Esseri Spaziali

La versione per l'Apple non può usare questa tecnica, poichè nell'Apple il comando

```
GET X$
```

fa lampeggiare il cursore CRT, e attende che il giocatore inserisca un carattere.

Dunque nell'Apple, "cursin" consiste in una chiamata a "cursor" con C e L fissate a CZ + CC e LZ + LL seguita dal comando

```
GET X$
```

Il punto finale da discutere è il ritorno del contenuto dello schermo. La chiamata a "uncursor" (Figura 9.32) che ha luogo immediatamente prima che "cursin" ritorni, fa sì che il carattere giusto (come deciso dal contenuto dell'array D) sia mostrato dove il cursore stava lampeggiando.

Questo completa la nostra discussione della parte di schermo su cui appare la figura. Ora occupiamoci di studiare i comandi per il programma del gioco degli Esseri Spaziali.

I comandi

Il programma del gioco degli Esseri Spaziali risponde ad un numero di comandi ad un solo carattere. Qui di seguito parleremo di:

— codifica usata per includere informazioni riguardanti i comandi primari e secondari in un solo numero

```
#Svuotare l'array dei puntini

cleardot    FOR LX = 0 TO HL
              FOR CX = 0 TO HC
                D(LX,CX) = 0
                NEXT CX
              NEXT LX
            RETURN

600  FORLX = 0TOHL:FORCX = 0TOHC:D(LX,CX) = 0:NEXTCX:NE-
     XTLX:RETURN
```

La routine "cleardot" inizializza l'array D con tutti zero. D conterrà un 1 per ciascuna posizione in cui nella figura appare un puntino.

Figura 9.24 — Svuotare l'array dei puntini

— meccanismo per l'inserimento di altri comandi

— l'associazione di certi caratteri con determinati comandi.

La routine "drawcmd" illustrata nella Figura 9.28 illustra la codifica usata per i comandi. "Drawcmd" inizia con una chiamata a "cursin" in modo da avere un carattere ASCII singolo, che "drawcmd" usa (e come lo usa verrà descritto tra breve) per aggiungere l'elemento dell'array C(CX) che contiene il numero che codifica l'informazione del comando. Il computo di C1 e C2 si basa sulla supposizione che il numero abbia la forma

$$n_2 + 100 \times n_1$$

n_2 è minore di 100. Il numero n_2 diventa il valore di C2 e n_1 diventa il valore di C1. La codifica di questi due numeri di comando in uno solo ha luogo nella routine "init" (Dalla Figura 9.38 alla 9.40).

Le informazioni primarie e secondarie riguardanti i comandi vengono usate così: Prima di tutto il contenuto di C1 viene interpretato nella routine principale (Figura 9.15). Ci sono tre codici di comandi primari:

— il codice per "exit" (uscita)

— un codice per il comando di disegno

— il codice per "miscellaneous" (vari).

Nel caso di "exit", non vengono usate le informazioni secondarie.

Per il comando di disegno, le informazioni secondarie vengono interpretate in "movecursor" (Figura 9.31). Per i comandi "miscellaneous" (vari), sono interpretate in "miscdo" le informazioni secondarie (Figura 9.30).

```
#Calcolare l'angolo in alto a sinistra per centrare la cornice

center      CZ = INT((SC - HC - 3)/2) + 1        #spazio per la cornice
            LZ = INT((SL - HL - 3)/2) + 1
            RETURN

610 CZ=INT((SC - HC - 3)/2) + 1:LZ = INT((SL - HL - 3)/2) + 1:
    RETURN
```

La routine "center" determina la posizione dove apparirà l'angolo in alto a sinistra della cornice così che la figura compaia esattamente al centro dello schermo.

Figura 9.25 — Centrare la figura sullo schermo

```
#Disegnare la cornice intorno all'area della figura

drawframe   L = LZ - 1:C = CZ: GOSUB cursor          #in alto a sinistra
            FOR CC = 0 TO HC                         #margine in alto
              PRINT TH$;
              NEXT CC
            PRINT "down";
            FOR LL = 0 TO HL                         #margine a destra
              PRINT RV$; "cursor back"; "cursor down"
              NEXT LL
            L = LZ: C = CZ - 1: GOSUB cursor         #in alto a sinistra
            FOR LL = 0 TO HL                         #margine a sinistra
              PRINT LV$; "cursor back"; "cursor down";
              NEXT LL
            PRINT "right";
            FOR CC = 0 TO HC                         #margine in basso
              PRINT BH$;
              NEXT CC
            RETURN

620*  L = LZ - 1:C = CZ : GOSUB940 : FORCC = 0TOHC : PRINTTH$;:
      NEXTCC : PRINTCHR$(17);
630*  FORLL=0TOHL:PRINTRV$;CHR$(157);CHR$(17);:NEXTLL
640*  L=LZ:C=CZ-1:GOSUB940:FORLL=0TOHL:PRINTLV$;
      CHR$(157);CHR$(17);:NEXTLL:PRINTCHR$(29);
650   FORCC=0TOHC:PRINTBH$;:NEXTCC:RETURN
```

La routine "drawframe" disegna la cornice che si trova intorno all'area della figura sullo schermo. Vengono usati i caratteri per muovere il cursore nel Pet e nel TRS-80, ma si possono usare anche delle routines molto simili sull'Apple. I comandi per muovere il cursore sono gli strumenti usati su questo sistema.

I comandi in BASIC per l'Apple sono:

```
620   L=LZ-1:C=CZ:GOSUB940: FOR CC=0 TO HC: PRINT TH$;:
      NEXT CC
630   FOR LL=0 TO HL:L=LZ+LL:C=CZ-1:GOSUB940: PRINT LV$;
640   C=CZ+HC+1:GOSUB940:PRINT RV$;: NEXT LL
650   L=LZ+HL+1:C=CZ:GOSUB940: FOR CC=0 TO HC: PRINT BH$;:
      NEXT CC: RETURN
```

* Nella versione per il TRS-80, i caratteri per muovere il cursore all'indietro, verso il basso e verso destra sono rispettivamente CHR$(24), CHR$(26) e CHR$(25).

Figura 9.26 — Incorniciare la figura

```
#Disegnare i puntini come indicato dall'array D
drawdots    FOR LL = 0 TO HL
              L = LL + LZ: C = CZ: GOSUB cursor      #posizionare il cursore
                                                     all'inizio della linea
              FOR CC = 0 TO HC
                IF D(LL,CC) < > 0 THEN
                  PRINT DT$;                         #stampare "puntino"
                else
                  PRINT " ";                         #stampare spazio vuoto
                NEXT CC
              NEXT LL

660 FORLL=0TOHL:L=LL+LZ:C=CZ:GOSUB940
670 FORCC=0TOHC:IFD(LL,CC) < >0THENPRINTDT$;:GOTO690
680 PRINT " ";
690 NEXTCC:NEXTLL:RETURN
```

La routine "drawdots" ridisegna la figura, piazzando un puntino o uno spazio vuoto a ciascuna posizione, cancellando cosi i puntini preesistenti. Questa routine viene usata per disegnare la nuova generazione nel Gioco della Vita trasformando la figura e anche per ridisegnare un tracciato precedente che è stato richiamato.

Figura 9.27 — Disegnare la figura

```
#Ottenere e decodificare un comando per disegnare ad un solo carattere
drawcmd     GOSUB cursin                           #far lampeggiare il cur-
                                                   sore, attendere l'input
            CX = ASC(X$) - CB                      #indice all'array C
            IF NOT 0 < = CX < = MX THEN
              CX = DX                              #valore d'errore se non
                                                   è fuori gamma
            C1 = INT(C(CX)/100)                    #comando primario
            C2 = C(CX) - 100*C1                    #comando secondario
            RETURN

700 GOSUB1080:CX=ASC(X$)-CB:IFCX>MXORCX<0THENCX=DX
710 C1=INT(C(CX)/100):C2=C(CX)-100*C1:RETURN
```

La routine "drawcmd" accetta un input ad un solo carattere proveniente dal giocatore e lo traduce in un comando primario C1 e in un secondario C2. Il comando primario viene interpretato nella routine principale (Figura 9.15). Per la maggior parte dei comandi primari, il comando secondario viene interpretato dalla routine "movecursor", (Figura 9.31). Per i comandi primari "vari", il comando secondario viene interpretato nella routine "miscdo" (Figura 9.30).

Figura 9.28 — Decodificare un comando

La routine "miscdo" è il punto in cui vengono integrati dei nuovi comandi nel programma. Il Gioco della Vita fornisce il modello per questo processo, poichè la sua integrazione nel programma è stata fatta dopo che il gioco degli Incontri Spaziali era già completo. I passi necessari per integrare il Gioco della Vita in questo programma sono stati:

- la definizione del comando "L" come "miscellaneous, number 1" (vario, numero 1)
- il riconoscimento da parte di "miscdo" di un valore C2 di 1 come un segnale per la chiamata di "life"
- fornire la routine "life" (Figura 9.44) e le sue subroutines "neighbors" (Figure 9.41, 9.42) e "lifedeath" (Figura 9.43).

```
#Comandi di disegno primari (indicizzati da C1)

space     GOSUB movecursor                  #muovere soltanto il
                                            cursore
          RETURN

mark      GOSUB putmark                     #piazzare un puntino,
                                            poi muovere il cursore
          GOSUB movecursor
          RETURN

unmark    GOSUB takemark                    #cancellare un puntino,
                                            poi muovere il cursore
          GOSUB movecursor
          RETURN

miscmd    GOSUB miscdo                      #altre funzioni
          RETURN

720  GOSUB780:RETURN
730  GOSUB800:GOSUB780:RETURN
740  GOSUB810:GOSUB780:RETURN
750  GOSUB760:RETURN
```

Queste sono le routines che vengono chiamate dalla routine principale (Figura 9.15) per interpretare i comandi primari. Ogni routine prima di tutto compie un'azione, (o non fa nulla), poi chiama un'altra routine per interpretare il comando secondario.

Figura 9.29 — Processo d'interpretazione del comando primario

```
#Routine di comandi vari

miscdo      IF C2 = 1 THEN                    #C2 = 1 codifica "Gioco
                                                della Vita"
              GOSUB life
            RETURN

760  IFC2=1THENGOSUB1450
770  RETURN
```

La routine "miscdo" è il punto nel programma in cui si possono aggiungere altri comandi. Per esempio, se si volesse un comando per cancellare l'intera figura, si potrebbe aggiungere una dichiarazione DATA alla routine d'inizializzazione (Figura 9.39) specificando un comando primario di 4 (vario) ed un comando secondario di 2. Poi si potrebbe alterare "miscdo" per riconoscere a valore C2 di 2 e per chiamare una subroutine composta di chiamate a "cleardots" e "drawdots". Poi, non appena il giocatore ha inserito il carattere specificato nella dichiarazione DATA, l'intera figura (ma non la cornice) scomparirebbe.

Figura 9.30 — Azioni varie

```
#Inviare la routine per posizionare il cursore

movecursor  IF 1 < = C2 < = 4 THEN            #controllare la gamma
                                                di C2
              ON C2 GOSUB
                cursleft,                     #muovere il cursore
                                                verso sinistra
                cursright,                    #muovere il cursore
                                                verso destra
                cursup,                       #muovere il cursore
                                                verso l'alto
                cursdown                      #muovere il cursore
                                                verso il basso
            RETURN

780  IFC2<1ORC2>4THENRETURN
790  ONC2GOSUB860,880,900,920:RETURN
```

La routine "movecursor" interpreta il comando secondario per i comandi primari: marcare, eliminare il segno e spazio. Il comando secondario determina dove il cursore deve spostarsi dopo aver seguito il comando primario.

Figura 9.31 — Specificare la posizione del cursore

```
#Routines per mostrare o per cancellare i caratteri dei puntini

putmark     GOSUB showdot              #inserire un puntino nella figura
            D(LL,CC) = 1               # a LL, CC e ricordarselo
            RETURN                     # in D

takemark    GOSUB zapdot               #cancellare il puntino a LL, CC
              D(LL,CC) = 0             # e azzerare la sua posizione in D
            RETURN

uncursor    IF D(LL,CC) = 1 THEN       #se il cursore era sovrapposto ad un puntino
              GOSUB showdot            # rimpiazzare il puntino
            else
              GOSUB zapdot             #altrimenti, eliminare tutti i segni
            RETURN

newcursor   IF D(LL,CC) = 1 THEN       #se il cursore è sovrapposto al puntino,
              CS$ = 0D$                # indicarlo
            else
              CS$ = ND$                #altrimenti, cursore normale
            RETURN

800 GOSUB1170:D(LL,CC)=1:RETURN
810 GOSUB1180:D(LL,CC)=0:RETURN
820 IFD(LL,CC)=1THENGOSUB1170:RETURN
830 GOSUB1180:RETURN
840 CS$=ND$:IFD(LL,CC)=1THENCS$=OD$
850 RETURN
```

Queste routines mostrano o cancellano dei caratteri dei puntini. Inoltre ai puntini normali che compongono la figura, ci sono due caratteri usati per il cursore. Uno che viene usato quando il cursore è in una posizione precedentemente vuota, e uno che viene usato quando il cursore è sovrapposto ad un puntino.

Figura 9.32 — Mostrare i puntini ed il cursore

```
# Routines per muovere il cursore

cursleft    IF CC > 0 THEN
              CC = CC - 1
            else
              CC = HC
            RETURN

cursright   IF CC < HC THEN
              CC = CC + 1
            else
              CC = 0
            RETURN

cursup      IF LL > 0 THEN
              LL = LL - 1
            else
              LL = HL
            RETURN

cursdown    IF LL < HL THEN
              LL = LL + 1
            else
              LL = 0
            RETURN

860  IFCC>0THENCC=CC-1:RETURN
870  CC=HC:RETURN
880  IFCC<HCTHENCC=CC+1:RETURN
890  CC=0:RETURN
900  IFLL>0THENLL=LL-1:RETURN
910  LL=HL:RETURN
920  IFLL<HLTHENLL=LL+1:RETURN
930  LL=0:RETURN
```

Queste routines calcolano la nuova posizione (LL,CC) del cursore dopo un comando per muoverlo. La posizione del cursore ha un valore che viene mantenuto nella gamma 0 < = LL < = HL e 0 < = CC < = HC. I valori "avvolgenti" vanno da HC o HL a zero quando superano il limite, e da 0 a HC o HL quando diminuiscono sotto lo zero.

Figura 9.33 — Muovere il cursore

```
#Posizionare il cursore alla colonna C, linea L

cursor      PRINT "home";                    #muovere verso l'angolo a sinistra in alto
            C = C mod SC                     #tenere sullo schermo
            L = L mod SL
            IF C > 0 THEN                    #se non è troppo a sinistra
              FOR ZZ = 1 TO C
                PRINT "right";               #PRINT (stampare) i caratteri per il cursore verso destra
              NEXT ZZ
            IF L > 0 THEN                    #se non è la linea in alto
              FOR ZZ = 1 TO L
                PRINT "down";                #PRINT (stampare) i caratteri per il cursore verso il basso
              NEXT ZZ
            RETURN

940* PRINTCHR$(19);:C=C-SC*INT(C/SC):L = L-SL*INT(L/SL)
950* IFC>0THENFORZZ=1TOC:PRINTCHR$(29);:NEXTZZ
960* IFL>=0THENFORZZ=1TOL:PRINTCHR$(17);:NEXTZZ
970  RETURN
```

La routine "cursor" è simile ad altre routines già comparse in questo libro. Qui, il cursore CRT viene posizionato alla colonna ed alla linea specificate nelle variabili C e L; in altre parti di questo libro, le discussioni vengono prese dalle variabili CC e LL. Negli Esseri Spaziali, CC e LL non possono essere usate come discussioni per "cursor" perchè vengono già usate per il cursore per la figura. Questo dimostra il problema centrale, con l'uso delle subroutines in BASIC non ci sono questi problemi, perché le discussioni vengono passate alle subroutines usando delle variabili globali. I comandi in BASIC per la versione per l'Apple sono:

```
940  C=C-SC*INT(C/SC):L=L-SL*INT(L/SL)
950  HTAB C+1
960  LTAB L+1
970  RETURN
```

---

* Nella versione per il TRS-80 di queste linee, i caratteri per spostare il cursore sono rappresentati da CHR$(28), CHR$(25) e CHR$(26).

Figura 9.34 — Posizionare il cursore CRT

```
#Pulire lo schermo

clearscreen   PRINT "clr";
              RETURN

#Input ad un solo carattere
onech         repeat
                GET X$
                until (X$ < > "")

#Input di stringa

stringin      XX$ = ""
              repeat {
                GOSUB onech
                IF X$ < > "delete" THEN {
                  PRINT X$;
                  IF X$ < > "return" THEN
                    XX$ = XX$ + X$
                  else break
                  }
                else IF LEN(XX$) < > 0 THEN {          #se ci sono dei caratteri
                  PRINT "delete string";               #cancellarne l'ultimo
                  XX$ = LEFT$(XX$, LEN(XX$) - 1)
                  }
                }
              RETURN
```

```
 980  PRINTCHR$(147);:RETURN
 990  GETX$:IFX$=""THEN990
1000  RETURN
1010  XX$=""
1020  GOSUB990:IFX$=CHR$(20)THEN1050
1030  PRINTX$;:IFX$=CHR$(13)THEN1070
1040  XX$=XX$+X$:GOTO1020
1050  IFLEN(XX$)=0THEN1020
1060  PRINTX$;:XX$=LEFT$(XX$,LEN(XX$) - 1):GOTO1020
1070  RETURN
```

Queste tre routines sono identiche ad altre già apparse in questo libro. (Vedi Figura 4.10 per le versioni per l'Apple e per il TRS-80).

Figura 9.35 — Routines di servizio ben note

```
#Inserire il carattere mentre il cursore lampeggia — versione per il Pet

cursin      GOSUB newcursor                    #fissare il carattere del
                                               cursore
            CF = 1                             #loop di lampeggio
            repeat{
              IF CF = 1 THEN
                GOSUB showcursor               #cursore funzionante
              else
                GOSUB showblank                #cursore fermo
              CF = – CF: TZ = TI               #rigirare la "bandierina", ottenere l'orario
              repeat{                          #attendere per l'input
                GET X$
                IF X$ < > "" THEN
                  {GOSUB uncursor: RETURN}     #rimuovere il cursore
                } until (TI – TZ > = FQ)       #ora di cambiare il cursore
            }

showcursor  C = CZ + CC: L = LZ + LL: GOSUB cursor
            PRINT CS$;:RETURN

1080  GOSUB 840:CF=1
1090  IFCF=1THENGOSUB1150:GOTO1110
1100  GOSUB1160
1110  CF=–CF:TZ=TI
1120  GETX$:IFX$< > ""THENGOSUB820:RETURN
1130  IFTI – TZ < FQTHEN1120
1140  GOTO1090
1150  C=CZ+CC:L=LZ=LL:GOSUB940:PRINTCS$;:RETURN
1160  C=CZ+CC:L=LZ+LL:GOSUB940:PRINT" ";:RETURN
```

La routine "cursin" e le sue due subroutines "showcursor" e "showblank" fanno lampeggiare il cursore, usando il sistema di conteggio del tempo nel Pet (TI). Sull'Apple e sul TRS-80 bisognerà usare un metodo diverso. Il comando GET nell'Apple fa lampeggiare il cursore (non come il GET del Pet). Nel TRS-80 è necessario usare un sistema di "ritardo controllato" o "loop di tempo". La versione per il TRS-80 compare nella Figura 9.36a. I comandi in BASIC per l'Apple sono:

```
1080  C=CZ+CC: L=LZ+LL: GOSUB940: GOSUB990: RETURN
```

Figura 9.36 — Far lampeggiare il cursore durante l'input

```
1080  GOSUB 840:CF=1
1090  IFCF=1THENGOSUB1150:GOTO1110
1100  GOSUB1160
1110  CF=-CF
1120  FORQQ=1TOCJ*FQ:X$=INKEY$:IFX$<>""
      THENGOSUB820:RETURN
1130  NEXTQQ
1140  GOTO1090
1150  C=CZ+CC:L=LZ+LL:GOSUB940:PRINTCS$;:RETURN
1160  C=CZ+CC:L=LZ+LL:GOSUB940:PRINT"";:RETURN
```

Figura 9.36a — BASIC per "Cursin" da usare sul TRS-80

```
#Mostrare un puntino a LL, CC

showdot     L = LZ + LL: C = CC + CZ: GOSUB cursor
            PRINT DT$;
            RETURN

#Cancellare il puntino a LL, CC

zapdot      L = LZ + LL: C = CC + CZ: GOSUB cursor
            PRINT " ";
            RETURN

1170  C=CZ+CC:L=LZ+LL:GOSUB940:PRINTDT$;:RETURN
1180  C=CZ+CC:L=LZ+LL:GOSUB940:PRINT" ";:RETURN
```

Le routines "showdot" e "zapdot" mostrano o cancellano un puntino alla posizione determinata dalle variabili LL e CC.

Figura 9.37 — Mostrare e cancellare i puntini

```
#Inizializzazione di Esseri Spaziali

init        READ SL, SC: DATA lines, columns      #grandezza dello scher-
                                                  mo
            READ BH$, TH$, LV$, RV$               #segmenti di linea per
            DATA "bottom", "top", "left", "right" #  la cornice della figura
            READ OD$, ND$: DATA "ondot", "nodot"  #caratteri del cur ore
                                                  per la figura
```

Figura 9.38 — Inizializzare Esseri Spaziali *(continua)*

```
READ FQ: DATA frequency                    #frequenza di lampeggiamento del cursore

READ DT$: DATA "dot"                       #puntino della figura
READ MC, ML, RL                            #larghezza massima ideale della figura, e

DATA maxcols, maxlines, memory limit       # limite imposto dalla memoria

MC = min(MC, SC - 2, RL)                   #numero massimo delle colonne nella figura

ML = min(ML, SL - 2, RL)                   #numero massimo delle righe nella figura

DIM D(ML, MC)                              #array dei puntini per la figura

READ CB, CM: DATA lochar, hichar           #gamma dei comandi in caratteri ASCII

MX = CM - CB: IF MX < 1 THEN STOP          #taglia dell'array dei comandi

DIM C(MX)
FOR CY = 0 TO MX
  C(CY) = misc * 100 + no-op               #riempire l'array dei comandi con nessuna operazione

  NEXT CY
repeat{                                    #riempire l'array dei comandi

  READ CH                                  #codice ASCII per il carattere

  IF CH < > - 1 THEN
    CY = CH - CB                           #indice per questo carattere

    IF NOT 0 < = CY < = MX THEN STOP
    READ C1, C2                            #codici dei comandi
    C(CY) = 100 * C1 + C2                  #primari e secondari
    }
  }until (CH = - 1)                        #- 1 fa terminare
#dichiarazioni DATA per i comandi (vedi Figura 9.39)
#Formato: codice DATA ASCII, comando primario, comando secondario
READ CH: DATA defaultchar                  #per i comandi fuori gamma

DX = CH - CB                               #indice all'array C
RETURN


  (I comandi in BASIC compaiono nella Figura 9.40)
```

Figura 9.38 — Inizializzare Esseri Spaziali *(Fine)*

```
#Dichiarazioni DATA per la tavola dei comandi (dalla routine d'inizializzazione)
#Il formato è: un (carattere), comando primario, comando secondario.
#un (carattere) cioè un codice ASCII per quel carattere.
#— 1 Invece di un codice ASCII per finire la lista
#Il comando primario assume i valori seguenti:
#     0     exit      (la figura appare come zero e uno)
#     1     space     (muovere il cursore per la fuga)
#     2     mark      (segnare un puntino e muovere il cursore)
#     3     unmark    (cancellare un puntino e muovere il cursore)
#     4     misc      (altri comandi)

#Comando secondario per space, mark o unmark:
#     1     left      (il cursore si muove verso sinistra)
#     2     right     (il cursore si muove verso destra)
#     3     up        (il cursore si muove verso l'alto)
#     4     down      (il cursore si muove verso il basso)

#Comando secondario per miscellaneous:
#     0     no-op     (non fare nulla)
#     1     life      (trasformare la figura secondo le regole del Gioco della Vi-
                      ta)

DATA a(4),       space,   left,    a(6),       space,   right
     a(8),       space,   up,      a(2),       space,   down
DATA a(shift 4), mark,    left,    a(shift 6), mark,    right
     a(shift 8), mark,    up,      a(shift 2), mark,    down
DATA a(0),       unmark,  left,    a(5),       unmark,  right
     a(7),       unmark,  up,      a(1),       unmark,  down
DATA a(L),       misc,    life
DATA a(E),       exit,    0,       -1
```

Queste dichiarazioni DATA definiscono i comandi ad un solo carattere riconosciuti dal programma. L'array C ha un solo inserimento per ogni codice ASCII nella gamma di possibili codici di comando definiti dalle variabili CB e CM. Il comando che corrisponde al valore di CB è codificato in C(0). Ogni comando è codificato da un numero a due cifre con base 100. Cioè se C1 e C2 contengono i codici primari e secondari da codificare in C(CY), allora il valore registrato in C(CY) è calcolato da 100*C1+C2.

L'uso di un —1 finale per terminare la lista rende possibile aggiungere o cancellare codici di comando con facilità.

Figura 9.39 — Creare i comandi

Diamo un'occhiata a questi passaggi, poichè vi potrebbe interessare di aggiungere degli altri programmi da voi inventati.

"L" è stata definita come "miscellaneous, number 1" dalla linea

```
DATA a (L), misc, life
```

nella Figura 9.39. Il significato di questi simboli viene spiegato nella Figura 9.39. Il codice che interpreta questa dichiarazione DATA è illustrato nella Figura 9.38. Ne parleremo più dettagliatamente tra breve.

Per far sì che "miscdo" riconosca un valore C2 di 1 come un segnale per chiamare "life" si è compreso una prova esplicita per un valore di 1. Se sarà necessario riconoscere altri comandi, bisognerà usare una struttura simile a quella di "movecursor" (Figura 9.3) anche per "miscdo". La costruzione ON...GOSUB è ideale per questo tipo di invio di comandi.

È stato anche abbastanza facile ottenere "life" e le sue subroutines. Le subroutines "lifedeath" e "neighbors" arricchiscono le regole del gioco in maniera molto semplice, nonostante bisognerà dedicare qualche momento in più per lo studio dell'array D per il conteggio dei "vicini". Si sarebbe potuto usare un array separato per il conteggio, ma sarebbe stato necessario avere molta più memoria a disposizione per il programma.

La routine "life" chiama "drawdots" e "newcursor" e usa le dimensioni della figura che sono state stabilite in precedenza da una chiamata a "setframe". Sotto tutti gli aspetti restanti, "life" è completamente autonoma.

Infine, parliamo della associazione di certi caratteri con comandi. Questa è la funzione del loop "repeat...until" nella routine "init". (Figura 9.38). Questo loop permette di usare un numero non specificato di dichiarazioni DATA. Queste dichiarazioni sono illustrate nella Figura 9.39. Ognuna contiene un codice ASCII, un codice per comando primario ed uno per quello secondario. Nel loop, il codice ASCII viene convertito in un indice di array, e i codici dei comandi primari e secondari vengono combinati in un numero singolo per archiviazione nell'array C all'indice che corrisponde al codice ASCII.

Se desiderate cambiare i codici che sono già stati assegnati ai comandi (per esempio per adattare una tastiera che non abbia i tasti dei numeri), le dichiarazioni DATA nella Figura 9.39 devono essere cambiate. Inoltre, se volete usare dei codici più grandi o più piccoli di quelli definiti da CM (character maz — carattere massimo) e CB (character base — carattere base) dovrete usare valori diversi per "lochar" e "hichar" in Figura 9.38. Questi sono i valori usati per limitare la grandezza dell'array C.

```
1190*   READSL,SC:DATA24,40
1210**  READBH$,TH$,LV$,RV$:DATA"—","_","","|","|"
1220**  READOD$,ND$:DATA"■","□"
1230**  READFQ:DATA18
1240**  READDT$:DATA"•"
1250*** READMC,ML,RL:DATA37,37,16
1260    IFMC>SC-2THENMC=SC-2
1270    IFMC>RLTHENMC=RL
1280    IFML>SL-2THENML=SL-2
1290    IFML>RLTHENML=RL
1300    DIMD(ML,MC)
1310**  READCB,CM:DATA48,185
1320    MX=CM-CB:IFMX<1THENSTOP
1330    DIMC(MX)
1340    FORCY=0TOMX:C(CY)=400:NEXTCY
1350    READCH:IFCH=-1THEN1380
1360    CY=CH-CB:IFCY<0ORCY>MXTHENSTOP
1370    READC1,C2:C(CY)=100*C1+C2:GOTO1350
1380    DATA52,1,1,54,1,2,56,1,3,50,1,4
1390**** DATA180,2,1,182,2,22,2,2,184,2,3,178,2,4
1400    DATA48,3,1,53,3,2,55,3,3,49,3,4
1410    DATA76,4,1
1420    DATA69,0,0,-1
1430    READCH:DATA52
1440    DX=CH-CB:RETURN
```

Questi sono i comandi in BASIC per la routine "init" illustrata nelle Figure 9.38 e 9.39. La routine "init" stabilisce i valori per diverse variabili che avrebbero potuto essere costanti in tutto il programma. L'uso di variabili facilita i cambiamenti necessari per adattare il programma a diversi sistemi di computer.

Le tecniche usate per fissare i valori dell'array C devono essere studiate.

---

* Nella versione per il TRS-80 di questa linea, le dimensioni dello schermo vengono rappresentate dai valori 16 e 64.

** Le versioni per l'Apple e per il TRS-80 di questa linea sono come segue:

```
1210 READ BH$,TH$,LV$,RV$: DATA "—","—","!","!"
1220 and 1230 omitted for Apple.
1220 READOD$,ND$:DATA"#", "*"          } for TRS-80
1230 READFQ:DATA18:READCJ:DATA .25     }
1240 READ DT$: DATA "0"
1310 READ CB,CM: DATA 0,127
```

*** Il valore di 16 per RL permette al programma di funzionare su Pet 8K. Per sistemi più grandi si può usare 100, dunque non c'è un limite per la memoria nella dimensione dell'array, poichè tutto quello che ci sta sullo schermo può stare nella memoria.

**** Nella versione per l'Apple e per il TRS-80 di questa linea, 180, 182, 184 e 178 sono sostituiti da 36, 38, 40 e 34.

Figura 9.40 — Inizializzare gli Esseri Spaziali in BASIC

```
#Contare le posizioni vicine

neighbors IF D(0,0) is odd THEN
            add 2 to D(0,1), D(1,0), D(1,1)
          IF D(0,HC) is odd THEN
            add 2 to D(0,HC - 1), D(1,HC), D(1,HC - 1)
          IF D(HL,0) is odd THEN
            add 2 to D(HL,1), D(HL - 1,0), D(HL - 1,1)
          IF D(HL,HC) is odd THEN
            add 2 to D(HL,HC - 1), D(HL - 1,HC), D(HL - 1,HC - 1)
          IF HC > 1 THEN
            FOR CC = 1 TO HC - 1
              IF D(0,CC) is odd THEN
                add 2 to D(0,CC - 1), D(0,CC + 1),
                D(1,CC - 1), D(1,CC), D(1,CC + 1)
              IF D(HL,CC) is odd THEN
                add 2 to D(HL,CC - 1), D(HL,CC + 1),
                D(HL - 1,CC - 1), D(HL - 1,CC), D(HL - 1,CC + 1)
              NEXT CC
          IF HL > 1 THEN
            FOR LL = 1 TO HL - 1
              IF D(LL,0) is odd THEN
                add 2 to D(LL - 1,0), D(LL + 1,0),
                D(LL - 1,1), D(LL,1), D(LL + 1,1)
              IF D(LL,HC) is odd THEN
                add 2 to D(LL - 1,HC), D(LL + 1,HC),
                D(LL - 1,HC - 1), D(LL,HC - 1), D(LL + 1,HC - 1)
              IF HC > 1 THEN
                FOR CC = 1 TO HC - 1
                  IF D(LL,CC) is odd THEN
                    add 2 to D(LL,CC - 1), D(LL,CC + 1),
                    D(LL - 1,CC - 1), D(LL - 1,CC), D(LL - 1,CC + 1),
                    D(LL + 1,CC - 1), D(LL + 1,CC), D(LL + 1,CC + 1)
                  NEXT CC
              NEXT LL
            RETURN
```

Figura 9.41 — Contare le posizioni vicine

```
1460 D=D(0,0)/2:IFD=INT(D)THEN1480
1470 D(0,1)=D(0,1) + 2:D(1,0)=D(1,0) + 2:D(1,1)=D(1,1) + 2
1480 D=D(0,HC)/2:IFD=INT(D)THEN1500
1490 D(0,HC - 1)=D(0,HC - 1)+2:D(1,HC)=D(1,HC)+2:D(1,HC -
     1)=D(1,HC - 1)+2
1500 D=D(HL,0)/2:IFD=INT(D)THEN1520
1510 D(HL,1)=D(HL,1)+2:D(HL-1,0)=D(HL-1,0)+2:D(HL-1,1)=
     D(HL - 1,1)+2
1520 D=D(HL,HC)/2:IFD=INT(D)THEN1550
1530 D(HL,HC - 1)=D(HL,HC - 1)+2:D(HL - 1,HC)=D(HL - 1,HC)+2
1540 D(HL - 1,HC - 1)=D(HL - 1,HC - 1)+2
1545 IFHC < = 1THEN1615
1550 FORCC=1TOHC - 1:D=D(0,CC)/2:IFD=INT(D)THEN1580
1560 D(0,CC - 1)=D(0,CC - 1)+2:D(0,CC + 1)=D(0,CC + 1) +
     2:D(1,CC - 1)=D(1,CC - 1) + 2
1570 D(1,CC)=D(1,CC) + 2:D(1,CC + 1)=D(1,CC + 1) + 2
1580 D=D(HL,CC)/2:IFD=INT(D)THEN1610
1590 D(HL,CC - 1)=D(HL,CC - 1) + 2:D(HL,CC + 1)=D(HL,CC + 1) + 2:
     D(HL - 1,CC)=D(HL - 1,CC) + 2
1600 D(HL - 1,CC - 1)=D(HL - 1,CC - 1) + 2:D(HL - 1,CC + 1)=D(HL
     - 1,CC + 1) + 2
1610 NEXTCC
1615 IFHL < = 1THENRETURN
1620 FORLL=1TOHL - 1:D=D(LL,0)/2:IFD=INT(D)THEN1650
1630 D(LL - 1,0)=D(LL - 1,0) + 2:D(LL + 1,0)=D(LL + 1,0) + 2:D(LL -
     1,1)=D(LL - 1,1) + 2
1640 D(LL,1)=D(LL,1) + 2:D(LL + 1,1)=D(LL + 1,1) + 2
1650 D=D(LL,HC)/2:IFD=INT(D)THEN 1675
1660 D(LL - 1,HC)=D(LL - 1,HC) + 2:D(LL + 1,HC)=D(LL + 1,HC) + 2:
     D(LL,HC - 1)=D(LL,HC - 1) + 2
1670 D(LL - 1.HC - 1)=D(LL - 1,HC - 1) + 2:D(LL + 1.HC - 1)=D(LL +
     1,HC - 1) + 2
1675 IFHC < = 1THENNEXTLL:RETURN
1680 FORCC=1TOHC - 1:D=D(LL,CC)/2:IFD=INT(D)THEN1730
1690 D(LL,CC - 1)=D(LL,CC - 1) + 2:D(LL,CC + 1)=D(LL,CC + 1) + 2
1700 D(LL - 1,CC - 1)=D(LL - 1,CC - 1) + 2:D(LL - 1,CC + 1)=D(LL -
     1,CC + 1) + 2
1710 D(LL + 1,CC - 1)=D(LL + 1,CC - 1) + 2:D(LL + 1,CC + 1)=D(LL +
     1,CC + 1) + 2
1720 D(LL - 1,CC)=D(LL - 1,CC) + 2:D(LL + 1,CC)=D(LL + 1,CC) + 2
1730 NEXTCC:NEXTLL:RETURN
```

La routine "neighbors" codifica il numero di posizioni vicine che ogni punto ha aggiungendo due volte il numero al valore (0 o 1) dell'inserimento dell'array D. La routine esamina l'intera figura, e dove trova un puntino aumenta il numero delle posizioni vicine. La routine è abbastanza complicata, poichè le posizioni vicino ai lati non hanno tutte le otto posizioni vicine. La routine potrebbe essere ridotta notevolmente usando una linea di confine falsa nell'array D poichè ogni punto avrebbe così le sue otto posizioni vicine. Posizioni nella linea di confine non avrebbero mai puntini.

Figura 9.42 — BASIC per "Neighbors"

```
#Prendere una decisione di vita o di morte

lifedeath  FOR LL = 0 TO HL
             FOR CC = 0 TO HC
               NB = INT(D(LL,CC)/2)          #2*posizioni vicine è
                                             registrato in D
               PT = D(LL,CC) - 2*NB          #il valore del puntino e
                                             di un bit basso
               IF case
                 NB < 2 OR NB > 3 THEN
                   D(LL,CC) = 0              #nessun puntino
                 NB = 3 THEN
                   D(LL,CC) = 1              #puntino
                 NB = 2 THEN
                   D(LL,CC) = PT             #mantenere il contenuto
                                             corrente
               NEXT CC
             NEXT LL
           RETURN
```

```
1740  FORLL=0TOHL:FORCC=0TOHC:D=D(LL,CC):NB = INT(D/2):PT =
      D - 2 * NB
1750  IFNB<2ORNB>3THEND(LL,CC)=0:GOTO1780
1760  IFNB=3THEND(LL,CC)=1:GOTO1780
1770  D(LL,CC)=PT
1780  NEXTCC:NEXTLL:RETURN
```

La routine "lifedeath" decide se una certa posizione deve o meno avere un puntino. Viene usato l'algoritmo seguente:

- Se la posizione ha meno di 2, o più di tre posizioni vicine, non apparirà nessun puntino.
- Se la posizione ha 2 posizioni vicine, allora la nuova generazione avrà un puntino in quella posizione e soltanto se la generazione precedente aveva lì un puntino.
- Se la posizione ha 3 posizioni vicine, allora ci sarà un puntino.

Poichè il nuovo array D viene costruito da quello precedente, il conto delle posizioni vicine scompare, e i valori dell'array sono di nuovo soltanto uno e zero.

Figura 9.43 — Decidere dove va inserito il puntino

SUGGERIMENTI E MIGLIORAMENTI

Ecco alcuni dei miglioramenti che si possono apportare al programma degli Esseri Spaziali:

- Incremento dei sistemi di archivio esterno per i modelli dei bits. Le routines "blockit", "choices" e "outstream" (Figura 9.23) sono le uniche che devono essere cambiate.
- Eliminazione della chiamata a "drawdots" in "setframe" (Figura 9.16) quando si è specificato un nuovo riquadro. Questo renderà l'inizializzazione del programma più veloce.
- Aggiunta di altri comandi di disegno.
  - Comandi che si muovono prima, e poi compiono l'azione.
  - Comandi che si muovono diagonalmente.
  - Un comando per "slacciare" l'ultimo comando.
- Modifica della routine "neighbors" per usare l'"avvolgimento". Cioè, contare (LL,HC) come una posizione vicina a (LL,O) o (O,CC) quale posizione vicina a (HL,CC). Usando questo metodo, a ciascun punto corrispondono otto posizioni vicine.
- Aumento della velocità delle trasformazioni che accadono durante il Gioco della Vita.

```
#Trasformare il tracciato seguendo le regole del Gioco della Vita

life    GOSUB neighbors: GOSUB lifedeath:        #trasformare e disegnare
        GOSUB drawdots
        CC = 0: L = 0: GOSUB newcursor: RETURN   #angolo in alto a sinistra

1450 GOSUB1460:GOSUB1740:GOSUB660:CC = 0:LL = 0:GOSUB840:
     RETURN
```

La routine "life" viene chiamata per implementare una trasformazione da "miscdo". Prima di tutto viene effettuata una chiamata a "neighbors" per contare le posizioni vicine a ciascuna posizione nella figura (una posizione vicina è un puntino vicino ad un altro). Poi viene chiamata "lifedeath" per trasformare il contenuto dell'array D. (per esempio i puntini della figura) secondo le regole del Gioco della Vita, che dipendono dal numero di posizioni vicine a ciascuna posizione. Infine viene chiamato "drawdots" per mostrare la figura, e il cursore viene riposizionato nell'angolo in alto a sinistra, usando una chiamata a "newcursor".

Figura 9.44 — Applicare le regole del Gioco della Vita alla figura

SOMMARIO

Il gioco degli Incontri Spaziali dà la possibilità al giocatore di programmare dei messaggi con matrici a puntini. Questi messaggi devono essere trasmessi ad esseri nello spazio sotto forma di flussi di bits che sono il prodotto di due numeri primi. I messaggi di questo tipo hanno il massimo numero di possibilità di essere interpretati correttamente da colui che li riceve nonostante questi debba interpretarne il formato.

Il Gioco della Vita è stato "innestato" su quello degli Incontri Spaziali, cosicchè i messaggi in figure che vengono preparati possono essere trasformati seguendo le regole del Gioco della Vita.

Il programma degli Esseri Spaziali che implementa il gioco composto ha due aspetti importanti: la parte dello schermo su cui appare la figura e i comandi. I componenti della figura sono: il riquadro, i puntini e il cursore della figura. Le routines "setframe", "center" e "drawframe" partecipano nel disegno del riquadro. I puntini sono l'immagine dell'array D, e vengono mostrati sullo schermo grazie alla routine "drawdots". Il cursore viene controllato dalla routine "cursin" che ne fissa il carattere e la sua locazione, ne fa funzionare il lampeggio intermittente, e fa ritornare il contenuto dello schermo quando viene levato il cursore.

I comandi a singolo carattere riconosciuti dal programma degli Esseri Spaziali vengono definiti nella routine "init". Certi caratteri sono associati con determinati comandi in tutto l'array C, i cui indici derivano da caratteri ASCII e il cui contenuto sono dei numeri singoli che codificano delle informazioni di comandi primari e secondari. Le informazioni primarie scelgono tra "exit", "drawing" o "miscellaneous", e quelle secondarie usano gli altri due gruppi. Il Gioco della Vita viene codificato nella categoria "miscellaneous".

È possibile effettuare diversi miglioramenti ed anche aggiungere altri comandi, ma soprattutto è possibile migliorare il sistema di archiviazione esterna per le immagini codificate.

APPENDICE A

IL GRUPPO DEI CARATTERI ASCII

La tavola seguente mostra il gruppo "standard" dei caratteri ASCII. I codici che usano numeri decimali da 0 a 127 sono elencati in ordine, e con essi vengono anche indicati i caratteri corrispondenti.

Tutti i sistemi in BASIC usano delle variazioni di questo codice. Il programma seguente è utile per determinare i valori in codice decimale assegnati a determinati caratteri nel vostro sistema:

```
1 GOSUB 2: PRINT ASC(X$);"";: GOTO 1
2 (Sub-routine ad un solo carattere di input, come mostrata per l'Apple, per il
  Pet e per il TRS—80 nella Figura 2.11) I numeri delle linee 720 e 730 nella Fi-
  gura 2.11 diventano qui 2 e 3.)
```

Dopo aver inserito questo programma ed aver schiacciato RUN, potrete schiacciare qualsiasi tasto e vedere sullo schermo apparire il valore decimale del codice corrispondente al tasto che avete schiacciato.

Oltre ai codici da 0 a 127, il Pet ed il TRS—80 usano i codici da 128 a 255 per scopi particolari. Su di un TRS—80, i codici da 128 a 191 codificano i "blocchi grafici", e i codici da 192 a 255 codificano stringhe di spazi vuoti. Su di un Pet i codici da 128 a 255 corrispondono a dei caratteri "grafici" particolari. Il programma seguente vi permetterà di far apparire sul video i caratteri che corrispondono ad una vasta gamma di codici:

```
1 INPUT "RANGE":N1,N2
2 FOR XX=N1 TO N2: PRINT"(";CHR$(XX);")";: NEXT XX: PRINT: GOTO1
```

Il programma richiederà prima di tutto una coppia di numeri specificando la gamma desiderata e poi mostrerà i caratteri corrispondenti ai numeri in quella gamma.

Ogni carattere apparirà racchiuso da parentesi.

Gruppo di caratteri ASCII

CODICE	CARATTERE	CODICE	CARATTERE	CODICE	CARATTERE	CODICE	CARATTERE
0	NUL	32[1]		64	@	96[5]	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39[2]	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	TAB	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44[3]	,	76	L	108	l
13	CR	45	—	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125[6]	}
30	RS	62	>	94	↑	126	~
31	US	63	?	95[4]	←	127[7]	RUBOUT

[1] spazio
[2] quota singola
[3] virgola
[4] o sottolineare
[5] accento
[6] o ALT MODE
[7] o DEL

L. 20.000

Cod. 522A ISBN88-7056-124-0

Al di là di un "classico" approccio alla programmazione è possibile avvicinarsi al BASIC, e quindi ai personal computer, in modo nuovo: giocando.
È infatti nei giochi, la molla che spinge i più all'acquisto di un piccolo computer, che il lettore può ritrovare tutte quelle situazioni reali di programmazione che gli saranno indispensabili nella comprensione e realizzazione di qualsiasi applicazione interattiva del proprio computer, anche le più sofisticate.
Questo senza annoiarsi ma entrando da subito all'interno della materia per imparare a comprendere il BASIC, il proprio computer e i computer in genere.
Conoscerete le diverse strutture dei programmi, la codifica delle informazioni, e gli algoritmi normalmente usati.
Imparerete a costruire dei programmi interattivi, ad applicare i principi di sviluppo dei sistemi a microcomputer, così come ad utilizzare il Free Basic (un BASIC "strutturato", mezzo di descrizione di un programma e non un linguaggio) per un approccio sistematico alla programmazione.

GRUPPO EDITORIALE JACKSON

Richard Mateosian

GIOCARE CON IL BASIC

68