Programmiamo videogiochi

di Marco Pesce

Eccoci di ritorno dopo la pausa dello scorso numero (cause di forza maggiore). Nella puntata precedente abbiamo affrontato un discorso molto generale sulle capacità di Amiga. In questa entreremo in alcuni dettagli relativi al funzionamento del Blitter e con il listato (in Assembler) di queste pagine saremo in grado di utilizzarlo in modalità BOB, ovvero potremo realizzare degli sprite semi-software da utilizzare in seguito per la costruzione dei nostri videogame

Per utilizzare il Blitter ci serviremo dell'accesso diretto ai suoi registri, in modo da sfruttarlo al meglio, mentre per tutte le operazioni di contorno, come l'apertura di schermi grafici o il caricamento da disk drive faremo uso delle routine del sistema operativo. Per accedere a quest'ultimo occorre addentrarsi nella selva delle "librerie". Ogni gruppo specifico di funzioni del sistema operativo è racchiuso in una particolare libreria; esistono perciò la libreria grafica, quella di Intuition, quella del DOS, ecc. In generale noi ci serviremo proprio di queste tre librerie (anche nel listato di questo mese). Per utilizzare una libreria e quindi le funzioni che essa contiene, occorre "aprirla" in modo che il sistema operativo saprà che la stiamo utilizzando. Quando si apre una libreria il S.O. ci restituisce un puntatore che dovrà essere utilizzato unitamente a degli "scostamenti" della locazione base, ognuno relativo ad una specifica routine. Paradossalmente per aprire una libreria occorre un'altra libreria, che comunque è sempre aperta: EXEC. Il suo puntatore base si trova alla locazione ram 4 (quattro). Le prime righe del nostro listato si preoccupano di disabilitare le IRQ (in questo caso lo scostamento di cui parlavamo prima è di −120) in modo da evitare qualunque tipo di multitasking (dannoso per la maggior parte dei videogiochi), quindi tra l'altro anche il funzionamento del puntatore del mouse. Le successive righe aprono la Graphics library, la Intuition library e la DOS library. I puntatori vengono memorizzati in locazioni tampone (il listato è scritto con il Macroassembler dell'AmigaDOS).

Per poter utilizzare il Blitter ovviamente occorrono degli screen sui quali "disegnare"; a tal proposito ci sarà utile la subroutine "openscreen" che tramite Intuition (è la maniera più sbrigativa) apre tutti gli schermi che vogliamo. Essa fa uso di una semplice struttura nella quale sono contenute informazioni relative alle caratteristiche dello schermo, ovvero:

dimensione X
dimensione Y
numero di colori
modo grafico

Per i nostri esperimenti utilizzeremo schermi in 320x200, 16 colori, bassa risoluzione.

Quando tale subroutine viene chiamata lo schermo apparirà sopra tutti gli altri e sarà completamente pulito (tranne il fatto che sarà visibile il puntatore allo schermo, ma questo si elimina modificando la copper list, cosa che vedremo in seguito); verranno restituiti in opportune locazioni i valori di alcuni fondamentali parametri relativi ad esso, indispensabili per il successivo utilizzo. Essi sono fondamentalmente:

PUNTSCREEN: puntatore alla struttura schermo
RASTPORT: puntatore alla struttura rastport
VIEWPORT: puntatore alla struttura viewport
BITP1: puntatore al primo bitplane
BITP2: puntatore al secondo bitplane
BITP3: puntatore al terzo bitplane
BITP4: puntatore al quarto e ultimo bitplane

Successivamente alla chiamata della routine "openscreen" è bene (a meno che non vogliamo utilizzare un solo screen) copiare i contenuti di dette locazioni in altre locazioni tampone, magari con il nome ricavato semplicemente aggiungendo un indice del numero di schermo che vogliamo assegnargli (ad esempio PUNTSCREEN1, B1BITP1, ecc.).

La routine Blity

E veniamo dunque al succo del listato, ovvero la routine Blity. Essa è stata realizzata essenzialmente in modo da trasferire sezioni di schermo rettangolari in un altro (o anche nello stesso) schermo, con l'accortezza di "incastra-

```
* ASSEMBLER


      moveq #0,d0
      move.l 4,a6
      jsr -120(a6)
      lea nome(pc),a1
      moveq #0,d0
      move.l 4,a6
      jsr -552(a6)
      move.l d0,Ilib
      lea nome2(pc),a1
      moveq #0,d0
      move.l 4,a6
      jsr -552(a6)
      move.l d0,Glib
      lea nome3(pc),a1
      moveq #00,d0
      move.l 4,a6
      jsr -552(a6)
      move.l d0,doslib


;******************************************


      jmp main
openscreen:move.l Ilib,a6
      jsr -198(a6)
      move.l d0,puntscreen
      move.l Glib,a6
      move.l puntscreen,a0

;queste istruzioni prendono gli
;attributi dello screen

      adda.l #44,a0
      move.l a0,viewport
      adda.l #40,a0
      move.l a0,rastport
      move.l viewport,a4
      adda.l #04,a4
      move.l (a4),colormap
      move.l colormap,a4
      adda.l #04,a4
      move.l (a4),colortab
      move.l rastport,a4
      adda.l #04,a4
      move.l (a4),bitmap
      move.l bitmap,a4
      adda.l #08,a4
      move.l (a4),bitp1
      move.l bitmap,a4
      adda.l #12,a4
      move.l (a4),bitp2
      move.l bitmap,a4
      adda.l #16,a4
      move.l (a4),bitp3
      move.l bitmap,a4
      adda.l #20,a4
      move.l (a4),bitp4
      rts


 ;CARICA LA SCHERMATA

 openfile
       move.l doslib,a6
       move.l #nomepic,d1
       move.l #1005,d2
       jsr -30(a6)
       move.l d0,handle

 ; Carica il file pezzo a pezzo
 ; (ora carica i colori ...16)

       move.l handle,d1
       move.l #inbuf,d2
       move.l #44,d3
       jsr -42(a6)

 ; Carica i bitplane

       move.l handle,d1
       move.l bitp1,d2
       move.l #8000,d3
       jsr -42(a6)
       move.l handle,d1
       move.l bitp2,d2
       move.l #8000,d3
       jsr -42(a6)
       move.l handle,d1
       move.l bitp3,d2
       move.l #8000,d3
       jsr -42(a6)
       move.l handle,d1
       move.l bitp4,d2
       move.l #8000,d3
       jsr -42(a6)

 ; Adesso ha finito e quindi chiude il file


       move.l doslib,a6
       move.l handle,d1
       jsr -36(a6)
       rts

 ;******************************************

 blity
       move.l Glib,a6
       jsr -456(a6);  ownblitter!!!
       clr.l d0
       clr.l d1
       clr.l d2
       move.w sposx,d0
       lsr.l #3,d0
       move.w sposy,d1
       move.b sxscr,d2
       mulu d2,d1
       add.l d0,d1
       move.l d1,sdeltapos
       clr.l d0
       clr.l d1
       clr.l d2
       move.w dposx,d0
       lsr.l #3,d0
       move.w dposy,d1
       move.b dxscr,d2
       mulu d2,d1
       add.l d0,d1
       move.l d1,ddeltapos

       clr.l d0
       clr.l d1
       clr.l d2
       move.b ampx,d0
       lsl.l #1,d0
       move.b sxscr,d1
       sub.l d0,d1
       move.w funzione,d2
       cmp.w #$0fca,d2
       bne trust1
       subi.l #02,d1
 trust1:move.w d1,smod
       clr.l d0
       clr.l d1
       move.b ampx,d0
       lsl.l #1,d0
       move.b dxscr,d1
       sub.l d0,d1
       cmp.w #$0fca,d2
       bne trust2
       subi.l #02,d1
 trust2:move.w d1,dmod

        clr.l d0
        move.w dposx,d0
        and.w #$000f,d0
        move.b d0,shift

  prepma:move.w #$ffff,$dff044
        move.w #0,$dff046

  prpblt:btst #14,$dff002
        bne prpblt
        move.l sdeltapos,d0
        add.l d0,sbit5
        add.l d0,sbit1
        move.l sbit1,$dff04c
        move.l sbit5,$dff050
        move.l ddeltapos,d0
        add.l d0,dbit1
        move.l dbit1,$dff048
        move.l dbit1,$dff054
        jsr cont1
  pau2  btst #14,$dff002
        bne pau2
        move.l sdeltapos,d0
        add.l d0,sbit2
        move.l sbit2,$dff04c
        move.l sbit5,$dff050
        move.l ddeltapos,d0
        add.l d0,dbit2
        move.l dbit2,$dff048
        move.l dbit2,$dff054
        jsr cont1


  pau3  btst #14,$dff002
        bne pau3
        move.l sdeltapos,d0
        add.l d0,sbit3
        move.l sbit3,$dff04c
        move.l sbit5,$dff050
```

(continua a pag. 242)

(segue da pag. 241)

```
        move.l ddeltapos,d0
        add.l d0,dbit3
        move.l dbit3,$dff048
        move.l dbit3,$dff054
        jsr cont1

pau4    btst #14,$dff002
        bne pau4
        move.l sdeltapos,d0
        add.l d0,sbit4
        move.l sbit4,$dff04c
        move.l sbit5,$dff050
        move.l ddeltapos,d0
        add.l d0,dbit4
        move.l dbit4,$dff048
        move.l dbit4,$dff054
        jsr cont1
        move.l Glib,a6
        jsr -462(a6);  disownblitter!!!
        rts ; CON QUESTO FINISCE LA ROUTINE BLITY!!!


cont1:clr.l d0
        move.b shift,d0
        lsl #08,d0
        lsl #04,d0
        move.w d0,$dff042
        or.w funzione,d0;  $0fca=bob (copia inc.)
                        ;  $05cc=copia esatta(con b e d)
        move.w d0,$dff040
        move.w smod,$dff064
        move.w smod,$dff062
        move.w dmod,$dff060
        move.w dmod,$dff066
        clr.l d0
        clr.l d1
        move.b ampx,d0
        move.w funzione,d1
        cmp.w #$0fca,d1
        bne trust
        add.b #1,d0
trust   move.w ampy,d1
        lsl.l #06,d1
        add.w d0,d1
        move.w d1,$dff058 ; questo e' blitsize!!!
        rts

schermo1 dc.l 0
schermo2 dc.l 0
schermo3 dc.l 0
schermo4 dc.l 0
schermo5 dc.l 0
schermo6 dc.l 0
schermo7 dc.l 0
schermo8 dc.l 0
schermo9 dc.l 0
schermo10 dc.l 0

b1bitp1 dc.l 0
b1bitp2 dc.l 0
b1bitp3 dc.l 0
b1bitp4 dc.l 0

b2bitp1 dc.l 0
b2bitp2 dc.l 0
b2bitp3 dc.l 0
b2bitp4 dc.l 0

b3bitp1 dc.l 0
b3bitp2 dc.l 0
b3bitp3 dc.l 0
b3bitp4 dc.l 0

b4bitp1 dc.l 0
b4bitp2 dc.l 0
b4bitp3 dc.l 0
b4bitp4 dc.l 0

b5bitp1 dc.l 0
b5bitp2 dc.l 0
b5bitp3 dc.l 0
b5bitp4 dc.l 0

b6bitp1 dc.l 0
b6bitp2 dc.l 0
b6bitp3 dc.l 0
b6bitp4 dc.l 0

b7bitp1 dc.l 0
b7bitp2 dc.l 0
b7bitp3 dc.l 0
b7bitp4 dc.l 0

b8bitp1 dc.l 0
b8bitp2 dc.l 0
b8bitp3 dc.l 0
b8bitp4 dc.l 0

b9bitp1 dc.l 0
b9bitp2 dc.l 0
b9bitp3 dc.l 0
b9bitp4 dc.l 0

b10bitp1 dc.l 0
b10bitp2 dc.l 0
b10bitp3 dc.l 0
b10bitp4 dc.l 0

rastport1 dc.l 0
viewport1 dc.l 0

rastport2 dc.l 0
viewport2 dc.l 0

rastport3 dc.l 0
viewport3 dc.l 0

rastport4 dc.l 0
viewport4 dc.l 0

rastport5 dc.l 0
viewport5 dc.l 0

rastport6 dc.l 0
viewport6 dc.l 0

rastport7 dc.l 0
viewport7 dc.l 0

rastport8 dc.l 0
viewport8 dc.l 0

rastport9 dc.l 0
viewport9 dc.l 0

rastport10 dc.l 0
viewport10 dc.l 0



funzione dc.w 0
sbit1 dc.l 0
sbit2 dc.l 0
sbit3 dc.l 0
sbit4 dc.l 0
sbit5 dc.l 0
dbit1 dc.l 0
dbit2 dc.l 0
dbit3 dc.l 0
dbit4 dc.l 0
sposx dc.w 0
sposy dc.w 0
dposx dc.w 0
dposy dc.w 0
ampx dc.b 0
ampy dc.w 0
sxscr dc.b 0
dxscr dc.b 0
sdeltapos dc.l 0
ddeltapos dc.l 0
smod dc.w 0
dmod dc.w 0
shift dc.b 0



colortab dc.l 0
bitmap dc.l 0
bitp1 dc.l 0
bitp2 dc.l 0
bitp3 dc.l 0
bitp4 dc.l 0
colormap dc.l 0
inbuf dc.w 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
colorbuf dc.l 0,0,0,0,0,0,0,0
rastport dc.l 0
viewport dc.l 0
handle dc.l 0
nomepic dc.b '00pic',0
nome3 dc.b 'dos.library',0
nome  dc.b 'intuition.library',0
nome2 dc.b 'graphics.library',0
dati  dc.w 0,0,320,200,4,1,64,$10f,0,0,0,0,0,0,0,0
dati2  dc.w 0,0,364,200,4,1,64,$10f,0,0,0,0,0,0,0,0
Glib  dc.l 0
Ilib  dc.l 0
puntscreen dc.l 0
doslib dc.l 0
```

Listato per il funzionamento hardware del Blitter.

re" la figura nello sfondo, grazie all'ausilio di una maschera che indichi i pixel da accendere (colori dello sprite) e quelli da lasciare invariati (sfondo). Detta maschera sarà contenuta in uno schermo ad un solo bitplane.

Grazie a questa capacità la routine è più che sufficiente a realizzare sprite animati. Se stiamo utilizzando uno schermo con scrolling software è sufficiente anche utilizzarla così com'è, ovvero stampando sul fondale la figura (o le figure) che ci interessa, senza preoccuparsi del fatto che bisogna ripristinare lo sfondo ad ogni spostamento dello sprite, dal momento che lo schermo sarà ridisegnato ad ogni ciclo di animazione (cosa che potrebbe anche essere falsa se usate uno scrolling software particolare). Se invece lo schermo è fisso dobbiamo preoccuparci anche di catturare lo sfondo e poi stamparci sopra lo sprite per poi ristampare lo sfondo quando lo sprite verrà spostato.

Vediamo quali input occorrono alla routine Blity (tra parentesi è indicato il formato del dato, ovvero Byte, Word o Longword):

SPOSX (W): posizione X della figura nello schermo sorgente (allineata con byte, quindi solo valori multipli di 8)
SPOSY (W): posizione Y (qualunque valore)
DPOSX (W): posizione X della figura nello schermo destinazione (qualunque valore).
DPOSY (W): posizione Y (qualunque valore)
AMPX (B): ampiezza X della figura (in byte)
AMPY (W): ampiezza Y (in pixel)
FUNZIONE (W): modalità di funzionamento (vedi in seguito)
SBIT1 (L): puntatore primo bitplane dello schermo sorgente
SBIT2 (L): secondo bitplane
SBIT3 (L): terzo
SBIT4 (L): quarto
SBIT5 (L): bitplane dello schermo delle maschere
DBIT1 (L): puntatore primo bitplane dello schermo destinazione
DBIT2 (L): secondo bitplane
DBIT3 (L): terzo
DBIT4 (L): quarto
SXSCR (B): ampiezza in byte dello schermo sorgente
DXSCR (B): ampiezza in byte dello schermo destinazione.

Per preparare gli screen con gli sprite bisogna seguire alcuni accorgimenti. Innanzitutto lo sprite ha una ampiezza che è un multiplo di 8 (8, 16, 24, 32, ecc.) quindi non devono esserci sprite che si dividono uno stesso byte altrimenti in fase di stampa verrebbe prelevata anche una piccola parte dello sprite che non vogliamo stampare; un altro accorgimento è quello di preparare la schermata con le ombre in modo che quest'ultime coincidano, in termini di coordinate (dando per scontato che devono coincidere anche come forme!) con le figure vere e proprie. Preparata una o più schermate, occorre segnarsi tutte le ampiezze e le coordinate alle quali si trovano le varie figure, per poi passare queste informazioni al programma di gestione; è utile a tal proposito (se utilizzate il Deluxe Paint per disegnare) attivare il modo che visualizza le coordinate del cursore. Tutte le schermate devono essere salvate su un dischetto, ma il loro formato dovrà essere in seguito adattato (vedi più avanti il caricamento degli screen).

Vediamo ora come organizzare una semplice struttura per gestire un sistema di sprite con la Blity.

Supponiamo di voler avere a disposizione un set di 8 sprite, aventi ad esempio le seguenti dimensioni:

1 sprite da 32x40
4 sprite da 16x10
3 sprite da 40x50

Ho idealmente supposto di suddividere gli sprite in modo da averne 1 per il giocatore, 4 per le armi e 3 per i nemici (fate attenzione, non stiamo parlando di fotogrammi, ma di figure contemporaneamente sullo schermo).

Supponiamo ancora di utilizzare uno schermo di gioco con fondale fisso: in tal caso dobbiamo memorizzare di volta in volta le porzioni di schermo che verranno sporcate dagli sprite per poi ridisegnarle. La routine Blity può essere sfruttata in due modi diversi, a seconda del valore che assegnamo alla "variabile" FUNZIONE. Per un valore pari a $0FCA essa funzionerà come abbiamo spiegato prima, ovvero servendosi delle ombre per trasferire la porzione di schermo; per un valore pari a $05CC il Blitter non utilizzerà le ombre, quindi possiamo sfruttare ciò per ottenere una copia esatta della sezione catturata (utile appunto per prelevare lo sfondo prima di stamparci lo sprite). Ci occorrono due schermi grafici per le animazioni, entrambi contenenti il fondale, in modo da disegnare su uno mentre è visulizzato l'altro; questo perché il Blitter, per quanto veloce, non riesce a stare al passo con il pennello elettronico (specie se gli sprite sono grandi e numerosi) e diversamente si avrebbero degli antiestetici "sfarfallamenti".

La sequenza di operazioni da eseguire è la seguente:

1) prelevare tutte le porzioni di schermo che verranno sporcate dagli sprite (a seconda delle dimensioni dell'area) e depositarle in un buffer;
2) stampare gli sprite avendo l'accortezza di cominciare con quello che dovrà essere visualizzato sotto a tutti gli altri;
3) attivare lo schermo appena stampato;
4) prelevare le porzioni di schermo da un buffer diverso da quello usato nello schermo ora visualizzato e stamparle nello schermo nascosto;
5) tornare al punto 1.

Caricamento degli screen

L'ultimo problema da risolvere è quello del caricamento delle schermate. Tramite le librerie del DOS è possibile accedere al disk drive ed è quello che fa la subroutine "openfile". All'apertura di un file ci viene restituito un puntatore che poi dovremo utilizzare ogni volta che vogliamo caricare una sezione di quel file e che ci servirà poi per chiuderlo una volta terminate le operazioni.

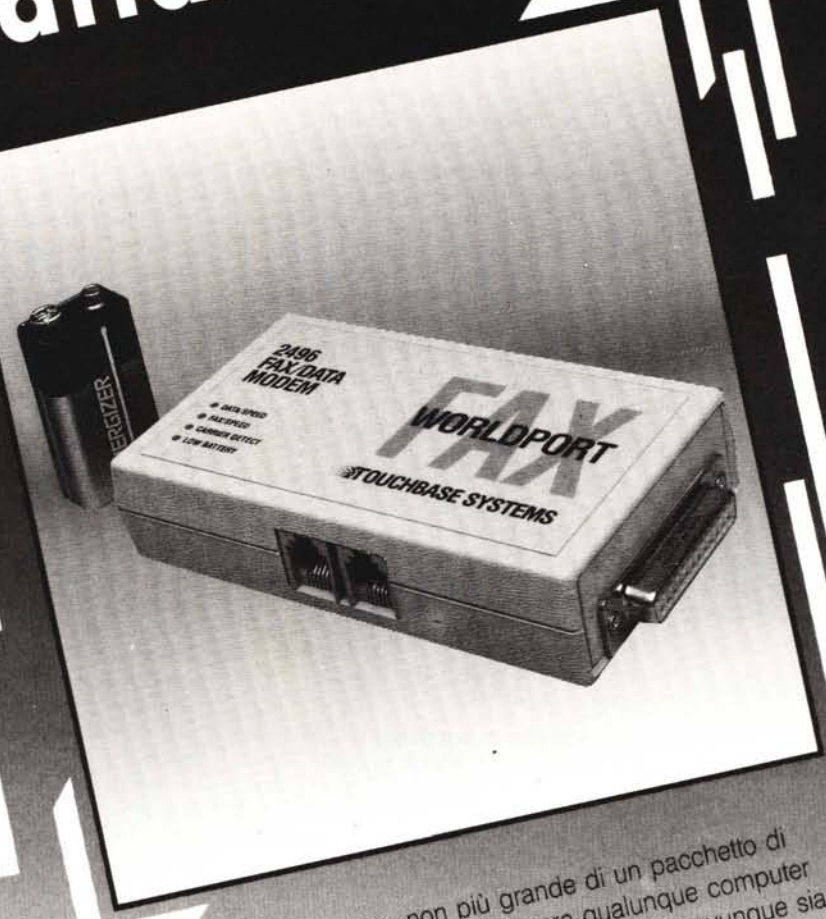
Per utilizzare la routine basta specificare il nome a partire dalla locazione "nomepic".

Il formato della schermata è il seguente:
— primi 44 byte: possono essere ridotti a 32 e quindi indicare solo la palette colore (2 byte per colore); i 12 byte aggiuntivi li possiamo utilizzare per memorizzare una nostra scritta in codice;
— successivi 8000 byte: primo bitplane
— successivi 8000 byte: secondo bitplane;
— successivi 8000 byte: terzo bitplane;
— successivi 8000 byte: quarto e ultimo bitplane.

Ovviamente dovremo fornire alla routine gli indirizzi del buffer per la palette colore e gli indirizzi dei puntatori ai bitplane. La routine è stata studiata in modo da essere utilizzata immediatamente dopo aver aperto uno schermo, visto che utilizza gli stessi puntatori della routine che svolge detto compito. E con questo abbiamo concluso con la descrizione del listato. È chiaro che esso può essere adattato alle vostre esigenze, ad esempio possono essere alterati i modi grafici o il numero dei bitplane da gestire, o anche il formato dei file delle schermate, questo ovviamente se siete in grado di comprendere quel tanto di linguaggio macchina necessario allo scopo; diversamente dovrete aspettare le prossime puntate, nelle quali approfondiremo il discorso. Per il momento arrivederci e buon lavoro.

MC

Worldport.
Gli affari in tasca



Pensate a un modem, non più grande di un pacchetto di sigarette, che vi permette di collegare qualunque computer — portatile o da tavolo — con qualunque altro, ovunque sia. E pensate alla possibilità di dotare il computer della funzionalità del fax, per comunicare con chiunque da un comune telefono.

Tutto questo è WORLDPORT, un oggetto indispensabile quando la mobilità e lo scambio di informazioni sono strumenti del successo. WORLDPORT è compatibile con i più diffusi programmi di comunicazione e viene fornito con il proprio software o "volendo" con il famoso CARBON COPY. I modem WORLDPORT sono disponibili anche nelle versioni a correzione di errore MNP 5 e Videotel.

WORLDPORT: un piccolo modem, grande come il mondo.

SMAU '90 4-8 ottobre
Pad. 19 - Stand A02/B03

TOUCHBASE SYSTEMS

Distributore per l'Italia:

Data Peripheral Italiana s.r.l.
20090 Segrate (MI) - Italy
Via L. da Vinci 21/23 - Tel. (02) 2137352 r.a.
Tlx 351490 DPI-I - Fax 2137831

AMIGUS