Programmiamo videogiochi (3)

di Marco Pesce

Con questa puntata metteremo nelle mani dei lettori gli ultimi strumenti indispensabili e quindi sufficienti, se usati unitamente a quelli delle scorse puntate, a realizzare un videogioco di una certa levatura; ovviamente i risultati dipendono in misura maggiore dalla vostra fantasia che, come in tutte le discipline, è di gran lunga più importante della tecnica vera e propria

Gli argomenti che tratteremo riguardano la gestione del joystick, l'utilizzo del processore audio, il controllo del Copper e l'allocazione della RAM, il tutto realizzato in ambiente Linguaggio Macchina, con accesso diretto ai registri del computer. E in effetti questi sono gli unici fondamenti che ancora vi mancano; abbiamo già parlato di come si aprono schermi, di come si utilizza il blitter per disegnarci, di come si caricano dati dal disco e di come si usano in generale le librerie dell'Amiga. Puntiamo la nostra attenzione su quei particolari dispositivi di controllo esterni detti volgarmente joystick. Niente di ipercomplicato ma qualcosa di insolito ci aspetta se vogliamo sapere quali interruttori sono chiusi e quali aperti. Le porte 1 e 2 sono collegate con le locazioni, rispettivamente $DFF00A e $DFF00C (in esadecimale). Per sapere se gli interruttori di destra o di sinistra sono chiusi o aperti occorre testare rispettivamente i bit 1 o 9 (se posti a uno l'interruttore è chiuso, ovvero il joystick è spostato in quella direzione). Per le direzioni alto e basso interviene la stranezza e occorre effettuare un OR esclusivo tra i bit 1 e 0 per sapere se l'interruttore della direzione «verso il basso» è chiuso e un OR esclusivo tra i bit 9 e 8 per la direzione «verso l'alto». Dal momento che non esiste una istruzione specifica del 68000 che effettua l'OR esclusivo (XOR) dobbiamo arrangiarci con mezzi propri. Le seguenti sono due subroutine che risolvono il problema per la porta 2, testandone il contenuto e saltando alla giusta routine:

```
MOVE.W $DFF00C,D0
AND.W #3,D0
CMP.W #1,D0
BEQ ESEGUI GIU'
CMP.W #2,D0
BEQ ESEGUI GIU'
RTS

MOVE.W $DFF00C,D0
AND.W #$0300,D0
CMP.W #$0100,D0
BEQ ESEGUI SU
CMP.W #$0200,D0
BEQ ESEGUI SU
RTS
```

Per conoscere lo stato del tasto Fire è sufficiente testare il bit 7 della locazione $BFE001 per il joystick in porta 2 e il bit 6 della stessa locazione per il joystick in porta 1; se posti a 1 il fire è premuto. Sul joystick non c'è molto altro da dire; se vogliamo testare una direzione diagonale del tipo «in alto a sinistra» basta testare l'interruttore di sinistra e verificare che contemporaneamente sia chiuso anche quello «verso l'alto»; discorso simile va fatto anche per le altre direzioni diagonali.

Occupiamoci ora della gestione del Copper e delle sue «liste».

Abbiamo già accennato al fatto che il Copper altro non è che un piccolo coprocessore che si occupa di operare in funzione del cosiddetto raster o, se preferite, pennello elettronico. Il suo operato è fondamentale soprattutto in vista del fatto che la gestione della pagina grafica di Amiga presenta delle particolarità di rilievo rispetto ad altri computer. Da sottolineare che senza il Copper nessuna pagina grafica potrebbe mai essere visualizzata. Il principio di funzionamento è il seguente: tramite l'istruzione WAIT il Copper attende che una determinata posizione del raster sia raggiunta; a tal punto entrano in azione le istruzioni MOVE (in quantità più o meno elevata, a seconda della funzione da eseguire), che trasferiscono dei dati in formato WORD nei registri di input/output situati a partire dalla locazione $DFF000. La posizione del pennello elettronico è definibile in termini di coordinata x e di coordinata y, quindi con una precisione molto spinta. In figura 1 potete osservare un elenco di alcuni registri fondamentali da utilizzare in abbinamento al Copper; è omesso il prefisso «$DFF—» anche in virtù del fatto che nelle istruzioni del Copper non va indicato (perché dato per scontato). Vediamo come si «assemblano» le istruzioni. Innanzi tutto esistono 2 puntatori (indipendenti) che ci consentono di scegliere la locazione di partenza del nostro «programma», limitata comunque ai primi 512 Kbyte (regola valida per tutti i chip custom di Amiga). I puntatori sono due, ma se ne utilizza uno alla volta,

ovviamente. Le istruzioni sono composte da due word distinte. L'istruzione WAIT ha la prima word con il bit meno significativo posto a 1 (sempre), mentre i bit da 15 a 8 sono quelli che specificano la posizione verticale del pennello elettronico (bit meno significativi); i bit da 7 a 0 specificano la posizione orizzontale (bit più significativi). Da notare che essendo solo 8 i bit che specificano la posizione verticale, quindi insufficienti per ricoprire l'estensione del raster che è di ben oltre 256 linee, occorre usufruire di un artificio per le posizioni più elevate; è sufficiente aspettare fino alla linea 256 e poi imporre una nuova «attesa», in modo che la somma delle due dia il risultato voluto, quindi, per esempio, per aspettare fino alla linea 300 occorre imporre una WAIT che attenda fino alla linea 256 e poi una nuova WAIT che attenda fino alla linea 44. La seconda WORD ha il bit meno significativo posto sempre a zero; il bit 15 è un particolare bit da utilizzare per il funzionamento del Copper in abbinamento al blitter e normalmente è posto a 1. I bit da 14 a 8 sono bit di confronto per la posizione verticale, così come i bit da 7 a 1 lo sono per la posizione orizzontale; la loro funzione è quella di escludere bit dal confronto se posti a zero; si tratta di un discorso un po' particolare che magari affronteremo in seguito («loop» di istruzioni eseguiti con il Copper...), per il momento diciamo che devono essere settati a 1 (tutti).

Le istruzioni «MOVE» si assemblano impostando il bit meno significativo della WORD a zero. I bit da 8 a 1 specificano il registro «destinazione» (ad esempio uno di quelli in fig. 1). I bit da 15 a 9 non vengono utilizzati, ma devono essere posti a zero. In sostanza la prima word di una istruzione «MOVE» equivale numericamente al registro destinazione, che essendo sempre pari ha il bit zero sempre impostato a zero e non arrivando per estensione ad occupare i bit da 15 a 9 li ha sempre impostati a zero. La seconda WORD di questa istruzione contiene il valore da trasferire nel registro specificato con la prima; tale valore è ovviamente a 16 bit.

```
004; bit 0, posizione verticale del pennello elettronico (bit piu'
significativo)
006; bit 15-8, posizione verticale del pennello elettronico (bit meno
significativi); bit 7-0 posizione orizzontale del pennello elettronico (7
bit piu' significativi)

080; locazione di start della prima copper list (3 bit piu' significativi)
082; locazione di start della prima copper list (15 bit meno significativi)
084; locazione di start della seconda copper list (3 bit piu'
significativi)
086; locazione di start della seconda copper list (15 bit piu'
significativi)
088; restart della prima copper list (quando viene letto)
08A; restart della seconda copper list (quando viene letto)

0E0; puntatore al primo bit plane (3 bit piu' significativi)
0E2; puntatore al primo bit plane (15 bit meno significativi)
0E4; bit plnae 2 (bit alti)
0E6; bit plane 2 (bit bassi)
0E8; bit plane 3 (bit alti)
0EA; bit plane 3 (bit bassi)
0EC; bit plane 4 (bit alti)
0EE; bit plane 4 (bit bassi)
0F0; bit plane 5 (bit alti)
0F2; bit plane 5 (bit bassi)
0F4; bit plane 6 (bit alti)
0F6; bit plane 6 (bit bassi)

180; palette colore numero 0
182; palette colore 1
184; palette colore 2
186; palette colore 3
188; palette colore 4
18A; palette colore 5
18C; palette colore 6
18E; palette colore 7
190; palette colore 8
192; palette colore 9
194; palette colore 10
196; palette colore 11
198; palette colore 12
19A; palette colore 13
19C; palette colore 14
19E; palette colore 15
1A0; palette colore 16
1A2; palette colore 17
1A4; palette colore 18
1A6; palette colore 19
1A8; palette colore 20
1AA; palette colore 21
1AC; palette colore 22
1AE; palette colore 23
1B0; palette colore 24
1B2; palette colore 25
1B4; palette colore 26
1B6; palette colore 27
1B8; palette colore 28
1BA; palette colore 29
1BC; palette colore 30
1BE; palette colore 31
```

Figura 1 - Elenco di alcuni registri di INPUT/OUTPUT (tutti con ampiezza word).

```
bit 15; SET/RESET- controlla se i canali devono essere accesi o spenti
bit 14; BBUSY- stato BUSY del blitter
bit 13; BZERO- stato ZERO del blitter
bit 12 e 11; non utilizzati
bit 10; BLTPRI- priorita' del blitter sul 68000 (se posto a 1)
bit 9; DMAEN- abilitazione generale del DMA
bit 8; BPLEN- abilitazione dei bit plane
bit 7; COPEN- abilitazione del copper
bit 6; BLTEN- abilitazione del blitter
bit 5; SPREN- abilitazione degli sprite
bit 4; DSKEN- abilitazione del Disk Drive
bit 3; AUD3EN- abilitazione del canale audio 3
bit 2; AUD2EN- abilitazione del canale audio 2
bit 1; AUD1EN- abilitazione del canale audio 1
bit 0; AUD0EN- abilitazione del canale audio 0
```

Figura 2 - Registro di controllo del DMA.

Spiegato il funzionamento delle istruzioni vediamo cosa possiamo effettivamente farci. La funzione che esse dovranno assolutamente svolgere è quella di riazzerare i puntatori dei bitplane ad ogni scansione del raster. Lo ripetiamo ancora una volta: i puntatori ai bit plane di Amiga sono dinamici, ovvero vengono incrementati durante la visualizzazione della pagina grafica, di conseguenza se non fossero riportati al loro valore iniziale, dopo la prima visualizzazione, verrebbero visualizzate le aree di memoria successive con gli effetti catastrofici che potete ben immaginarvi. Il problema si potrebbe risolvere con un sistema di interruzioni tramite 68000, il Macroassembler Amigados impostate la seguente struttura dati:

```
DCOPLIST dc.w $e0, 0, $e2, 0, $e4, 0, $e6, 0, $e8, 0, $ea, 0, $ec, 0, $ee,
0, $ffff, $fffe
```

ripuntando i bitplane ogni qualvolta il raster raggiunge la posizione limite della pagina grafica, ma sarebbe uno sforzo inutile; abbiamo la comodità di avere il Copper, perché non dovremmo utilizzarlo. Ovviamente questo è solo l'impiego più «banale»; ad esempio possiamo fare in modo di avere lo schermo suddiviso in più modi grafici (come credete che siano realizzati gli screen a «tendina» di intuition?), o una sfumatura del fondale in 256 colori utilizzando un solo bitplane! Molte delle «presentazioni» che si vedono sui dischetti dei vari «pirati» sfruttano il Copper per realizzare incredibili arcobaleni animati..., ma diamo un esempio pratico di utilizzo. Realizziamo uno schermo grafico a 4 bitplane con il Copper usufruendo dei bitplane ricavati tramite l'apertura sotto Intuition, in modo da «immobilizzare» lo screen e da eliminare la possibilità di scrolling con il puntatore del mouse. I registri che ci interessano sono i 4 (8 Word) a partire dalla locazione $DFF0E0. In essi dovranno essere depositati i puntatori ai bitplane ogni volta che il pennello elettronico ricomincerà a disegnare la pagina grafica. Dobbiamo innanzitutto ricavarci i 4 puntatori ai bitplane e questo ve l'ho insegnato la scorsa puntata, poi dobbiamo preparare la Copper list. La prima «istruzione» sarà una MOVE e non una WAIT come molti di voi si aspetterebbero e questo perché non dobbiamo aspettare nessuna posizione del pennello elettronico, ma agire subito. Se usate

Essa ci servirà da base ed è standard per qualunque screen a 4 bitplane, bassa risoluzione. Impostate anche la prossima linea dati:

```
COPLIST dc.l 0
```

Questa locazione memorizzerà la locazione di start della Copper list. Dal momento che la Copper list deve essere allocata nei primi 512 Kbyte non possiamo utilizzare direttamente la struttura dati DCOPLIST in quanto se il nostro computer è espanso il programma (quindi anche tale struttura) verrà caricato a partire dalla RAM più in alto (fast), con conseguente malfunzionamento. Per risolvere basta farci «dedicare» una zona RAM chip da Exec e trasferirvici dentro la struttura (debitamente completata) DCOPLIST. È giunto quindi il momento di aprire una parentesi sull'allocazione della RAM tramite Exec. Su Amiga infatti non è possibile impossessarsi liberamente di un blocco di RAM qualsiasi senza provocare come minimo un bel «Guru» a causa del sistema multitasking; la procedura da seguire è quella di «chiederla» a Exec, che senza problemi ce la riserverà esclusivamente a noi (sempre che ci sia) e ci dirà dove è situata. Quello che noi dobbiamo fare è specificare che tipo di RAM ci serve (CHIP o FAST), quindi la sua ampiezza. Il seguente è un esempio di «chiamata» della routine:

```
MOVE.L 4,A6
MOVE.L #36,D0 ; CHIEDIAMO 36 BYTE
MOVE.L #$10003,D1
JSR -198(A6)
MOVE.L D0,COPLIST; METTE IL PUNTATORE ALLA RAM IN "COPLIST"
```

Quella che ci viene data è una fettina di chip ram (come specificato dal valore posto in D1); per chiedere FAST o in alternativa CHIP (se la prima non c'è) basta sostituire il valore $10003 con il valore $10000. Esiste anche la possibilità di chiedere esclusivamente e forzatamente la FAST (impostando il valore $10002), ma ciò equivale a escludere il funzionamento su un Amiga inespanso. Chiudiamo pure la parentesi e torniamo alla Copper list. Ci resta ancora da sostituire al valore 0 di ogni MOVE della struttura DCOPLIST preparata in precedenza, il valore effettivo dei puntatori ai bitplane. Per semplicità suppongo che avete preparato una struttura (come suggerito la scorsa puntata) che contiene i bitplane degli schermi aperti, chiamando i bitplane con B1BITP1, B1BITP2, ecc. memorizzandoli in ordine crescente. Il seguente programmino trasferisce detti valori al punto giusto della DCOPLIST:

```
      MOVE.L #B1BITP1,A1
      MOVE.L #DCOPLIST1,A2
      ADD.L #2,A2
LOOP  MOVE.W (A1)+,(A2)+
      ADD.L #2,A2
      CMP.L #B2BITP1,A1
      BNE LOOP
```

B1BITP1 è quindi il primo bitplane del primo schermo da voi aperto, mentre B2BITP1 è il primo bitplane del secondo schermo da voi (si spera) aperto. A questo punto la DCOPLIST è pronta per diventare Copper list effettiva. Non ci resta che trasferirla in chip ram con il seguente programmino:

```
      move.l #9,d0
      move.l coplist,a0
      move.l #dcoplist,a1
lp    move.l (a1)+,(a2)+
      sub.b #1,d0
      bne lp
```

E ora l'ultima operazione; attivare la

```
0A0; locazione di start del canale 0 (3 bit piu' significativi)
0A2; locazione di start del canale 0 (15 bit meno significativi)
0A4; lunghezza del campione
0A6; periodo del campione
0A8; volume
0AA; dati
0B0; locazione di start del canale 1 (3 bit piu' significativi)
0B2; locazione di start del canale 1 (15 bit meno significativi)
0B4; lunghezza del campione
0B6; periodo del campione
0B8; volume
0BA; dati
0C0; locazione di start del canale 2 (3 bit piu' significativi)
0C2; locazione di start del canale 2 (15 bit meno significativi)
0C4; lunghezza del campione
0C6; periodo del campione
0C8; volume
0CA; dati
0D0; locazione di start del canale 3 (3 bit piu' significativi)
0D2; locazione di start del canale 3 (15 bit meno significativi)
0D4; lunghezza del campione
0D6; periodo del campione
0D8; volume
0DA; dati

Range del volume; 0-64 (64=max volume)
Range del periodo; 124-65535 (124=max frequenza)
Range della lunghezza; 0-65535 (in word, quindi max 128 Kbyte)
```

Figura 3 - Registri del chip audio (da aggiungere il prefisso $DFF...).

copper list; è sufficiente memorizzare in $DFF080 l'indirizzo di partenza (COPLIST) e «leggere» la locazione $DFF088 (per l'avviamento); il tutto se vogliamo utilizzare il primo dei due puntatori alle Copper list. È bene attendere una ben determinata posizione del raster prima di effettuare il trasferimento, per evitare inceppamenti; basta eseguire la seguente subroutine:

▼

```
LP1  MOVE.W $DFF004,D0
     AND.W #01,D0
     BNE LP1
     MOVE.W #20,$DFF096
     MOVE.L COPLIST,$DFF080
     MOVE.W $DFF088,D0
     MOVE.W #$2C81,$DFF08E; impostano le dimensioni
     MOVE.W #$F4B1,$DFF090;      dello schermo
```

La locazione $DFF096 è il controllo del DMA; nel nostro caso vengono disabilitati gli sprite quindi anche il fastidioso puntatore del mouse. Tra breve vedremo meglio come funziona tale importante registro. Possiamo a questo punto ritenere conclusa la trattazione del Copper; lascio a voi la sperimentazione di giochi con la palette colore.

Siamo giunti all'ultimo argomento del mese: la gestione del chip audio. Come tutti sapete l'Amiga utilizza suoni campionati, o almeno questo è l'uso più frequente che subiscono i suoi chip; in questa sede ci occuperemo di come si sfrutta tale modalità.

Un qualunque suono campionato è costituito da una successione più o meno lunga di byte che indicano le escursioni da far effettuare tramite convertitore digitale/analogico all'altoparlante del monitor (o del televisore). Tale successione di byte può essere definita grazie a un campionatore esterno, corredato di apposito programma per la sua gestione, oppure può essere sintetizzato internamente al computer, con un algoritmo matematico. Noi supponiamo che il blocco di byte sia già pronto per l'uso e già presente in RAM (se non avete un campionatore non dovreste avere difficoltà a «rimediare» suoni da qualche programma musicale o da qualche videogame, spulciando nel dischetto). Innanzi tutto dobbiamo fornire a Paula (il chip audio) la locazione di partenza del blocco dati, memorizzandola in uno dei 4 registri relativi ai 4 canali audio. Il secondo passo è quello di definire la sua lunghezza. Il periodo si riferisce alla frequenza di riproduzione, che ci permetterà di eseguire note di diversa «altezza» con lo stesso «campione». Infine il volume, indipendente per ciascuno dei 4 canali. Tutto questo comunque non è sufficiente ad udire un suono in quanto occorre anche abilitare il relativo canale DMA. Approfittiamone per parlare un po' del registro che controlla tutti i canali DMA di Amiga. In figura 2 potete osservare una sommaria descrizione dei bit in esso compresi. Il bit 15 è fondamentale e serve a specificare se con la nostra operazione vogliamo attivare o disattivare il canale specificato; quando tale bit è a 1 tutti i canali DMA specificati, ponendo il loro bit rappresentativo a 1, verranno attivati, mentre se esso è posto a 0 gli stessi canali saranno disattivati; tale accorgimento ci permette di scrivere in questo registro senza preoccuparci di sapere lo stato degli altri canali che non ci interessano. Per attivare il canale DMA 0 (quello del canale audio 0) basta impostare nel registro controllo DMA il valore binario 1000000000000001 ($8001 in esadecimale). Solo se un canale DMA è attivo il chip ad esso relativo può funzionare. Il bit 9 è l'abilitazione generale; se si spegne tale flag tutti i DMA vengono bloccati, ma basta riaccenderlo per riportare la situazione alla normalità.

La stessa accortezza, avuta nel richiedere la RAM per la Copper list ad Exec, occorre per memorizzare i campioni del chip Paula. Il seguente è un esempio di utilizzo del canale 0:

```
MOVE.L LOCPARTENZA,$DFF0A0
MOVE.W #10000,$DFF0A4
MOVE.W #440,$DFF0A6
MOVE.W #40,$DFF0A8; max 64
MOVE.W #$8001,$DFF098
```

Tale sequenza fa partire il canale in un loop infinito; se vogliamo effettuare una sola «scansione» di dati, ovvero vogliamo che il campione venga eseguito una volta sola, basta impostare, dopo un ciclo raster:

```
MOVE.L LOCVUOTA,$DFF0A0
MOVE.W #1,$DFF0A4
```

e la sequenza avrà termine dopo il primo«passaggio».

Per realizzare un motivo musicale occorre campionare i singoli strumenti, e realizzare una lista delle note da eseguire, badando a rispettare i tempi, ma se volete sbrigarvela basta campionare un intero brano e mandarlo in loop, magari avendo l'accortezza di variarne il periodo ogni tanto, per rendere meno monotona la «cantilena» che si realizza con tale sistema. I canali possono essere sfruttati ovviamente anche per realizzare effetti sonori; in tal caso occorre utilizzare la modalità ad un solo «passaggio», per non ripetere all'infinito l'effetto impostato. E con questo abbiamo proprio finito. A presto.