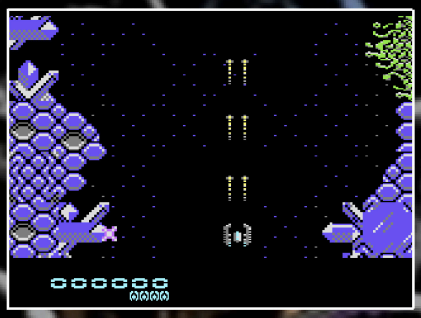
RetroMagazine

semplicemente retro

Numero 16 - Anno 3 - Luglio/Agosto 2019 - WWW.RETROMAGAZINE.NET - Pubblicazione gratuita



Benvenuta Ivy Claire

Nucleo 447
Diario di sviluppo e download (C64)

Retroeventi

Giappone 4ª puntata: Hard Off / Book Off

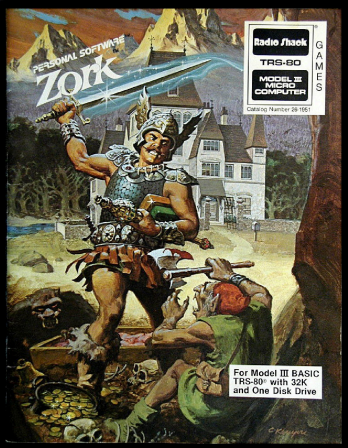
Calcolare in multipla precisione-parte II

Piange il telefono (C64)

I file binari del DOS Atari

HOLLYWOOD
The Cross-Platform Multimedia Application Layer

Introduzione ad Hollywood (prima parte)



Zork I in italiano

Retrocomputing: nuove prospettive dal passato

Siamo alle soglie del 2020 e la *computer science* (o informatica da noi in Italia) rappresenta una disciplina più che mai in voga e ormai alla base della quasi totalità delle attività umane. Non c'è settore della società, dell'economia, dell'educazione, della politica che negli ultimi 40 anni e oltre, non sia stato investito e, in alcuni casi, profondamente mutato da strumenti hardware e software costruiti per immagazzinare e gestire bit d'informazione. Persino il denaro ed i registri pubblici sono ormai (e non si può far altro che prevedere che lo saranno sempre di più) nient'altro che sequenze binarie. Anno dopo anno constatiamo di trovarci nel pieno dell'era dell'informazione ed ogni giorno che passa, una delle pagine di "Being digital" ("Essere digitali") di Nicholas Negroponte (1995, Knopf Inc., USA) si avvera, prende vita e si trasforma in un servizio o in un prodotto che concorre all'economia o al benessere della società. E come ogni disciplina, prodotto della inesauribile energia della mente dell'Uomo, ci sono una storia ed un'evoluzione di cui tener conto, da conservare e proteggere dal tempo e dall'oblìo. Dal telaio di Ada Lovelace, alla macchina di Turing, dall'avvento di macchine elettromeccaniche all'invenzione dei transistor, dai primi calcolatori fino ai computer quantici, una sterminata distesa di studi, progressi, scoperte ed implementazioni che raggiungono non solo i centri di calcolo specializzati o le basi militari ma anche l'uomo comune, costituisce uno dei presupposti essenziali della futura vita dell'intera umanità.

Noi di RetroMagazine ci limitiamo a gravitare attorno al cosiddetto "retrocomputing", pubblicando articoli e recensioni su prodotti hardware e software, che ripercorrono mezzo secolo e oltre di evoluzione della *computer science*. E a volte lo facciamo senza rimarcare fino in fondo l'importanza e la vastità del numero di possibili implicazioni di un'attività che è stato il fondamento di progressi scientifici e pratici, di nuovi sviluppi economici, della nascita di nuove industrie e di professioni. Insomma, sì, è vero che scriviamo e leggiamo RetroMagazine per un particolare articolo sul nostro home computer preferito oppure per la recensione di un vecchio videogame, per il reportage di un evento o per l'intervista ad un mostro sacro dei videogiochi, ma per un attimo vi invito a fare un passo a lato, ad allontanarvi un po' con la visione e a "zoomare" all'indietro.

Vi apparirà come d'incanto un mondo tutto nuovo da esplorare che include una serie di estensioni e sviluppi a cui probabilmente non avevate pensato finora.

La definizione ufficiale di "retrocomputing" (fonte Garzanti) è la seguente: "L'attività di reperire computer di vecchie generazioni, che hanno rappresentato una fase importante nell'evoluzione dell'informatica, ripararli, rimetterli in funzione e conservarli." La sintesi è assolutamente necessaria per un dizionario, ma noi sappiamo che retrocomputing significa molto di più, soprattutto perché, nel tempo, il numero di appassionati è cresciuto, giocoforza, e con loro il numero di attività ad esso collegate. Ad oggi sarebbe riduttivo parlare di retrocomputing limitandoci alla definizione. Pensate soltanto quante attività sono legate all'hardware, dalla semplice scoperta e recupero, dalla riparazione all'uso effettivo, il collezionismo *tout-court*, i progetti di *modding*, l'implementazione di funzioni esterne usando le *user port*, i riproduttori basati su FPGA, la produzione di nuovi retrocomputer (ad es. la reingegnerizzazione di macchine basate su CPU a 8 bit), l'adattamento di periferiche moderne ai vecchi modelli (*storage*, stampa, audio, video, ecc.).

E che dire della parte software? Una miniera infinita di vecchio software da reperire, riscoprire e conservare nei vari formati, la disponibilità di emulatori sempre più accurati, lo sviluppo di nuovi giochi e applicazioni per le piattaforme "classiche" a 8/16 bit, il moltiplicarsi di strumenti di coding sui sistemi moderni che hanno come target i vecchi sistemi (*crosscoding*), i tool software più efficienti rivisti e corretti alla luce di nuove tecniche di programmazione, i *code repository* con tutorial e guide per imparare a programmare le vecchie macchine, i *demo*, gli *intro* e gli *hack* grafici, sonori e computazionali che ancor oggi sbalordiscono gli utenti riuscendo a strizzare in pochi byte effetti e prestazioni inimmaginabili date le limitatissime risorse hardware a disposizione.

E poi ancora: eventi, festival, club e circoli, competizioni videoludiche, archivi digitali di riviste e libri specializzati, musei dell'informatica funzionante, mostre itineranti, mercatini e aste online e molto, molto altro. Un movimento in crescita nei numeri, un'energia in costante aumento in tutto il mondo, anche in quelle regioni d'Europa e del pianeta che in un primo tempo erano rimaste indietro per quanto concerne la diffusione di computer, periferiche e strumenti digitali a livello home o business. E noi di RetroMagazine, nel nostro piccolo, vogliamo rappresentare un altro tassello importante del retrocomputing, per come lo abbiamo definito qui in forma più estesa rispetto al dizionario: una delle riviste moderne dedicate alla storia

dell'informatica, con un'incredibile quantità di materiale da riscoprire e condividere. Certo, ci occupiamo di passato, da alcuni definito obsoleto, ma in verità si tratta di un passato vivo e vivace, in grado di insegnare molto anche alle nuove generazioni.
Quindi, cari lettori, "zoomate" anche voi all'indietro e perdetevi nel vasto territorio delle attività legate al retrocomputing che abbiamo cercato di condensare in poche righe in questo editoriale. Mettete su il cappello da esploratore e approfondite insieme a noi uno o più campi di questa meravigliosa disciplina. Oppure trovatene uno nuovo e contribuite a rendere ancora più interessante ed avvincente questo nostro mondo.
Il divertimento è assicurato.

David La Monaca / Cercamon

E parlando di nuove prospettive e di nuove generazioni, cogliamo l'occasione per farvi partecipi di un lietissimo evento che riguarda il fondatore di questa rivista.

RetroMagazine è lieta di annunciare la nascita di una nuova, piccola fan! La redazione di RetroMagazine abbraccia calorosamente i suoi genitori, Francesco e Paola, augurando alla loro piccola Ivy Claire tanta salute, gioia, felicità e tutto ciò che di meglio possa avere dalla vita. Auguri!!!

SOMMARIO

Hanno collaborato alla stesura di questo numero

- *Lorenzo Ventura*
- *Gianluca Girelli*
- *Leonardo Vettori*
- *David La Monaca*
- *Giorgio Balestrieri*
- *Alberto Apostolo*
- *Michele Ugolini*
- *Francesco Fiorentini*
- *Marco Pistorio*
- *Leonardo Giordani*
- *Daniele Brahimi*
- *Ermanno Betori*
- *Starfox Mulder*
- *Giuseppe Fedele*
- *Francesco Clementoni a.k.a. Arturo Dente*
- *Luca Cusani*
- *Querino Ialongo*

Immagine di copertina realizzata da Flavio Soldani

Smurf Crash

di Lorenzo Ventura

Quando ero bambino, in famiglia avevamo un ColecoVision. Le cartucce erano molto costose, sicché l'acquisto di un gioco nuovo era un evento destinato solo a ricorrenze speciali. Da ciascun titolo cercavamo così di cavare il massimo possibile, spesso reinventando il modo di giocarci con regole nostre.

Ciò era quasi scontato: a differenza di alcuni eccellenti titoli di Activision e Imagic, molti dei giochi prodotti da Coleco, pur offrendo grafica e sonoro molto curati, si esaurivano presto. Naturalmente la tecnologia della console poneva dei chiari limiti alla varietà di grafica e situazioni, ma lo stesso, anche su console meno sofisticate, i giochi migliori sopperivano introducendo una modulazione delle dinamiche, per cui l'interesse, foss'anche ridotto alla sola sfida di battere il proprio high-score, non svaniva nel volgere di pochi pomeriggi.

Invece, quasi tutti i giochi di Coleco proponevano quattro livelli di difficoltà completamente statici: con un minimo di pratica, indifferentemente da quale si fosse scelto, non era difficile protrarre la partita indefinitamente: quale differenza con gli originali arcade, a cui nel comparto grafico si avvicinavano parecchio, come nel caso di Donkey Kong e Zaxxon!

A questo riguardo, l'iconico gioco dei Puffi risultava perfettamente in media. La sera di Natale 1983, ricordo di essere rimasto sorpreso e indispettito vedendo mia sorella, che allora non aveva neanche cinque anni, completare il gioco dopo un paio di partite (parte del dispetto nasceva dal fatto che io, invece, non avevo capito come si facesse a far saltare il puffo). Nonostante la mancanza di sfida, i Puffi mi piacevano, forse proprio per la quasi totale assenza di quel senso di minaccia imminente connaturato all'azione della maggioranza dei videogiochi arcade.

L'avventura del puffo è letteralmente una passeggiata in un agreste paesaggio flip screen, dove, a parte la cattura da parte di un corvo che fa sporadiche comparsate, l'unica possibilità di perdere una vita consiste nello schiantarsi contro gli sparsi ostacoli fissi, quali cancelletti, cespugli acuminati, stalagmiti. Oltre alla bella grafica, il gioco dei Puffi ha una caratteristica particolare.

La disposizione degli ostacoli sul percorso è stabilita casualmente all'ingresso in ciascuna schermata: se si ha l'impressione che siano arrangiati in modo fastidioso, si può convenientemente uscire dalla schermata e rientrarvi con l'intento di ottenere una combinazione più accessibile.

Pur essendo il gioco flip screen, al passaggio da un quadro all'altro lo schermo scorre per fare entrare la nuova scena; ciò avviene con un'animazione un po' a strattoni, e nel frattempo si sente la musica stonare e rallentare. Chiaramente, ciò accade perché nello stampare l'intero schermo in una posizione sfalsata, come succede durante ciascun fotogramma della carrellata, il codice del gioco non fa in tempo a chiamare con la frequenza necessaria la subroutine che suona la musica.

Al tempo, l'impressione che ne traevo era che realizzare questa transizione di ambienti fosse una faccenda particolarmente impegnativa per il gioco: e di qui il pensiero di provare a portarlo a superare un fantomatico limite di rottura era diretto.

Il bello è che ci si riusciva: muovendo il puffo in una nuova schermata, ritornando nella precedente, riandando nell'altra e infine ripetendo, dopo qualche iterazione il gioco andava in crash, mostrando questo kill screen:

Da notare il punteggio di 919500 punti (googlabile!), ragguardevole ma comunque facilmente raggiungibile in poco più di un'ora di gioco.

A qualche decennio di distanza, mi è venuta la curiosità di capire esattamente cosa c'è dietro a codesto crash. Nell'indisponibilità della console reale, ho sopperito ricorrendo all'emulazione, peraltro fondamentale per poter venire a capo del mistero.
Come mi ricordavo, per eseguire il trucco, è necessario giocare al livello di difficoltà 3 o 4, circostanza che suggerisce un collegamento con il malefico corvo, che non compare nei livelli di difficoltà minori.

Avendo a che fare con un crash del programma, non mi aspettavo di ottenere esattamente lo stesso effetto visto sulla console reale. La maggior parte delle macchine a 8 bit, accedendo a una locazione di memoria fuori dallo spazio di indirizzamento, piuttosto

che fermarsi in uno stato terminale, continua imperterrita a macinare istruzioni, seguendo flussi di esecuzione potenzialmente senza senso: il program counter avanza inarrestabile, magari incappando di tanto in tanto su routine che causano qualche effetto osservabile, come scrivere a video o, eventualmente, il reset della macchina.
Anche se l'esecuzione di codice dal comportamento non specificato è comunque deterministica, la maggior parte degli emulatori non realizza il livello di fedeltà necessario a replicare l'esecuzione di un programma che gira in queste condizioni anomale sull'hardware reale. Per esempio, sotto il veterano Colem di Marat Fayzullin, dopo il crash, lo schermo si riempie di bande colorate che si sovrascrivono incessantemente; versioni correnti del mame, invece, si arrestano su di un kill screen differente da quello originale, che comunque compare un attimo prima per qualche istante.

Per affrontare l'analisi del crash, mi sono fatto un ripasso dell'architettura della console, che sintetizzo qui di seguito.
Il Colecovision è una macchina semplice, basata interamente su componenti standard, a differenza di molte altre console e home computer dell'epoca. La CPU è uno Zilog Z80, mentre grafica e audio sono generati da due chip di Texas Instrument. A titolo di curiosità, riporto che la poco conosciuta console Creativision di VTech, commercializzata in Italia da Zanussi (!), ha un'architettura essenzialmente identica, ad eccezione della scelta di una CPU differente. Lo spazio di indirizzamento del Colecovision è suddiviso come segue:

```
0000h-1FFFh 8 KB ROM di sistema incorporata nella console
7000h-73FFh 1 KB RAM
8000h-BFFFh 16 KB ROM cartuccia
```

La ROM di sistema contiene varie routine di supporto, p.es. per leggere l'input, gestire gli sprite, impostare timer e via dicendo; oltre a semplificare il lavoro del programmatore, permettendo di astrarre il codice dei giochi dai dettagli dell'hardware, il fatto che queste utilità siano incluse nella console consente di risparmiare preziosissimo spazio nella rom della cartuccia. L'uso di una libreria di routine facilita anche il compito del reverse engineer, in quanto, manuale alla mano (google), per dirimere la struttura del codice basta andare a cercarsi nel disassemblato del gioco le chiamate a funzioni chiave come la lettura del joystick, la sincronizzazione con il vblank, e così via.

Come emulatore ho scelto di utilizzare il mame, che implementa un più che discreto debugger, con breakpoint e watchpoint condizionali oltre agli usuali comandi per ispezionare codice e memoria: praticamente tutto quanto occorre per condurre l'esercizio che mi ero proposto.

Per prima cosa, ho cercato l'indirizzo del punteggio facendo una ricerca in RAM del valore mostrato dal segnapunti. All'inizio mi pareva non si trovasse, ma poi ho osservato che le ultime due cifre mostrate sono sempre fisse a '00', e che pertanto il valore che dovevo cercare era al netto delle centinaia.

Ho così trovato due copie del segnapunti, alle locazioni di memoria 71FAh e 72B5h. Impostando dei watch point, ho individuato il codice che le accede in scrittura. Il valore alla prima locazione, in particolare, è semplicemente una copia dell'altro. A differenza dell'indirizzo di destinazione, inchiavardato nel codice, il puntatore sorgente è caricato da una variabile, ospitata all'indirizzo 718Dh, come si vede dal disassemblato:

```
A400 LD IX,(718Dh)
A404 LD H,(IX+01h)
A407 LD L,(IX+00h)
A40A LD (72B5h),HL
```

Esaminando il codice, si vede che la dereferenziazione disaccoppia la logica del gioco da quella di presentazione. Come usava una volta, nei Puffi si può giocare una partita in cui due giocatori si alternano ogni volta che uno dei due perde una vita. La variabile all'indirizzo 718Dh punta al punteggio del giocatore corrente, che viene copiato all'indirizzo 72B5h, che funge da master per l'aggiornamento del video.

Il passo successivo è stato impostare un watch point all'indirizzo 72B5h, condizionato sulla scrittura di un valore diverso da quello corrente. Come previsto, il balletto del puffo a cavallo degli schermi è risultato in una condizione intercettata dal watch point: e difatti il punteggio all'indirizzo 72B5h valeva 23EBh, ovvero, convertendo in decimale e aggiungendo i due zeri impliciti, il "numero magico" 919500. Bingo! Dal PC si vede che la sovrascrittura è effettuata dall'istruzione all'indirizzo A40Ah. Il registro IX contiene il valore 0605h, un indirizzo della ROM, da cui in ultima analisi proviene il valore che sporca il punteggio.

Chiaramente il problema è nel puntatore all'indirizzo 718Dh. Dopo aver lasciato andare avanti l'esecuzione e constatato l'atteso crash, ho rifatto il giro impostando un watch point condizionale sull'indirizzo 718Dh. Riproducendo la condizione di errore, ho trovato il PC all'indirizzo 1070h, nella ROM di sistema, dunque. Risalendo il call stack, si vede che lo statement "applicativo" (nella ROM della cartuccia) di provenienza è questo:

```
BBDC call $1FCD
```

che, verificando sulla documentazione del Colecovision, è la funzione per impostare un timer. Finalmente, i pezzi del puzzle si incastrano al loro posto. L'API dei timer consente di definire dei timeout,

di cui venire notificati successivamente. Ai timeout pendenti corrispondono delle strutture dati che sono salvate in una lista ospitata a un indirizzo in RAM a scelta del programmatore. Ai livelli di difficolta 3 e 4, cambiando schermata, il gioco dei Puffi imposta un breve timeout, allo scadere del quale far scendere in campo il corvo: ma non lo rimuove dalla lista se, prima che il timeout scada, si esce dalla schermata. In questo modo, muovendosi avanti e indietro a cavallo di due schermate, la lista si allunga sino a sovrascrivere il puntatore all'indirizzo 718Dh.

Capire il dettaglio di come il gioco successivamente vada in crash è stata la parte più difficile dell'esercizio, e in fondo la meno interessante. Onde evitare di annoiare a morte chi mi avesse seguito fin qui, riassumo: contiguo al puntatore alla struttura dati del giocatore corrente, c'è un flag che essenzialmente rappresenta la condizione di esistenza in vita del puffo, che finisce per essere sporcato in concomitanza con la sovrascrittura del puntatore. Pertanto, quando il timer danneggia le strutture dati, il contesto di gioco viene reinizializzato come se si fosse persa una vita, e nel far questo viene chiamata con dei parametri invalidi (in quanto calcolati a partire dal solito puntatore) la routine di sistema "putobj". Questa è una specie di memcpy, e finisce per invalidare l'intero contenuto della RAM, mandando completamente a monte l'esecuzione del programma.

E questo è quanto. Risolvere il puzzle mi ha dato qualche soddisfazione, non ultima quella di poter dire "ora so perché".

Introduzione ad Hollywood - prima parte

di Gianluca Girelli della redazione di BITPLANE

Introduzione

Hollywood e' un linguaggio di programmazione orientato al multimedia che puo' essere usato per creare applicazioni grafiche in modo semplice ed efficace. E' stato sviluppato con l'idea di rendere la creazione di software il piu' facile possibile e pertanto e' indicato sia per il principiante che per l'utente piu' smaliziato.

Hollywood dispone di una libreria di funzioni molto estesa (che comprende al momento circa 500 diversi comandi) che semplifica la creazione di giochi 2D, presentazioni grafiche e applicazioni varie. L'autore, Andreas Falkenhan, ne ha iniziato lo sviluppo nel 2002 e quindi questo prodotto, giunto ora alla versione 4.8, costituisce oggi un pacchetto maturo, stabile ed affidabile.

Hollywood e' un prodotto commerciale pubblicato da Airsoft Softwair. Una delle caratteristiche di maggior pregio di Hollywood e' il "cross-compiler" interno, un compilatore che permette cioe' di sviluppare software per piu' piattaforme contemporaneamente senza dover cambiare una sola linea di codice. Questo sistema funzione per ogni piattaforma su cui il software e' stato implementato. Per esempio, e' possibile compilare un pacchetto per Mac OS X mentre

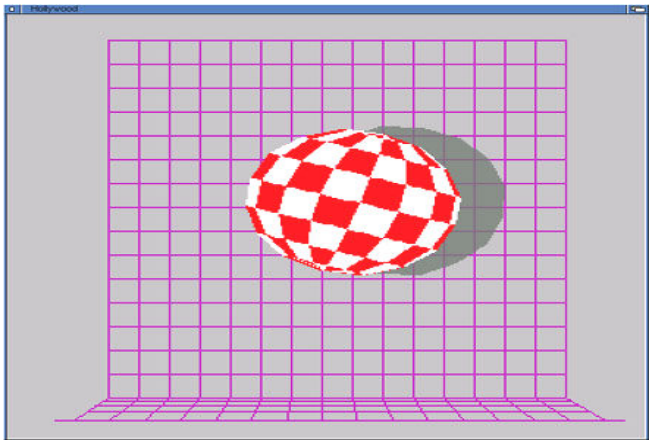


Fig.1 - Boing Ball demo rifatto con Hollywood

si sta usando la versione Windows del programma.

Pur essendo molto "leggero" (circa due megabytes), Hollywood e' un linguaggio di programmazione molto potente, che non richiede la presenza di componenti esterni. Per questo motivo, e' ideale per creare programmi che devono funzionare direttamente senza altre aggiunte. Hollywood supporta al momento le architetture seguenti: AmigaOS 3 (M68K), AmigaOS 4 (PPC), AROS (i386), MorphOS (PPC), WarpOS (M68K/PPC) e Windows (i386). Inoltre, Hollywood puo' creare eseguibili anche per Linux (i386) e Mac OS X (i386 e PPC), ma per tali architetture non e' al momento

Fig.2 - Font demo su paesaggio in background

disponibile in versione "stand-alone". Bisognera' quindi ricorrere alla versione AmigaOS o a quella per Windows.

Uno sguardo in profondità

Hollywood e' stato sviluppato basandosi quasi interamente su Lua, il linguaggio di scripting sviluppato in Brasile da Roberto Ierusalimschy, Waldemar Celes e Luiz Henrique de Figueiredo. Usando quindi il kernel Lua 5.0.2., potenziato con le librerie JPEG, PNG e con il PTPlay di Ronald Hof, Timm S. Mueller e Per Johansson, l'autore ha creato un pacchetto che ha il grosso vantaggio di occuparsi da solo del "framework" del progetto da sviluppare. Non dobbiamo quindi preoccuparci se la nostra applicazione deve girare per M68K, PowerPC o i386. Con Hollywood, funzionera' su tutte le architetture. Allo stesso modo, lo scheduler interno dell'interprete si occupera' da solo di sincronizzare immagini e musica, indipendentemente dal fatto che la nostra applicazione giri su una CPU a 50MHz o ad 1GHz. A questo punto, l'utente piu' smaliziato avra' notato che abbiamo parlato sia di compilatore che di interprete, il che suona in modo quanto meno bizzarro a chi mastica

Fig.3 - Transizione "a stella" tra due foto diverse

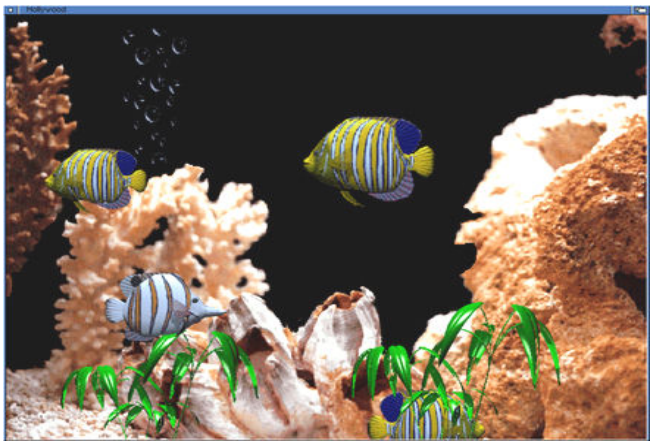


Fig.4 - Demo screensaver "Acquario"

almeno un po' di informatica. In effetti, se il nostro codice deve girare sulla piattaforma su cui Hollywood e' in esecuzione, esso viene semplicemente tradotto in un "bytecode" customizzato, che sara' direttamente letto ed eseguito dall'interprete. Se invece vogliamo esportare il nostro programma su un'altra piattaforma (oppure il codice e' completo e non necessita di altre modifiche) procederemo alla compilazione del tutto. In tal modo verra' preparato un pacchetto autocaricante(come gli "exe" di Windows) che contiene una versione "player" di Hollywood. Bastera' un doppio click sull'icona del nostro file per farlo partire.

Essendo un prodotto completamente integrato in AmigaOS e non dipendente da un hardware specifico (in particolare i custom chip grafici e sonori dei sistemi Amiga "classic"), Hollywood gira anche su emulazioni come Amithlon o WinUAE con notevole velocita'. E' inoltre provvisto di una GUI che permette di controllare il programma facilmente ed efficacemente. La versione Windows di Hollywood e' dotata di un vero e proprio ambiente IDE, mentre la versione Amiga puo' utilizzare allo scopo la "Special Edition" di CodeBench.

Le possibilita' sono quasi illimitate. L'elenco dei comandi disponibili varia dalla gestione del DOS (AmigaDOS!) alla completa manipolazione di grafica a suono con decine di effetti di transizione disponibili, incluse le trasparenze con "masking" e "alpha channel". I "datatypes" sono pienamente supportati, cosi' come i font standard bitmap ed i "true type". Tutti i font vettoriali sono corretti con l'antialiasing. Manipolare le immagini e' estremamente facile grazie non solo all'uso di comandi come "rotate", "scale" e "stretch", ma anche a comandi del tipo: "scale to gray level", "replace colors" e "mix pictures". In altre parole ... tutto quello che vi serve per la moderna manipolazione delle immagini.
Comandi dedicati permettono di gestire le animazioni con "sprite", "layers" e "double buffering", oltre alla sicronia con il sonoro. Anche chi si interessa di "game development" sara' quindi pienamente soddisfatto dall'uso di Hollywood. Se non avete particolare esigenze di velocita', per le quali e' comunque necessario ricorrere a linguaggi come il "C", creare sprite di 16.7 milioni di colori con trasparenze e collisioni esatte pixel-pixel e' (quasi) un gioco da ragazzi.
Hollywood e' una applicazione "high-end" che ha bisogno di un ambiente moderno per lavorare, nel senso che la modalita' di schermo attiva deve essere hi-color o truecolor. Dopo il boot del sistema, inoltre, sono necessari almeno 16 megabyte di memoria libera. Per quanto riguarda il sonoro, il sistema e' perfettamente integrato e "fully retargetable" attraverso AHI e permette di suonare moduli ProTracker, semplici suoni o MP3 (in modo nativo su Windows o attraverso la libreria mpega.library su AmigaOS). E' possibile suonare un numero virtualmente illimitato di suoni di qualsiasi lunghezza allo stesso tempo. Naturalmente a 16-bit e 44.1KHz.

Dulcis in fundo, Hollywood supporta completamente ARexx tramite l'ormai arcinoto sistema delle "porte". Cio' permette di potenziare ulteriormente i propri "scripts" e di creare applicazioni che possono essere controllate via ARexx. Tutto questo puo' essere fatto con sole poche linee di codice. Di nuovo, massima flessibilita' con la minima quantita' di lavoro.

Primi passi con il linguaggio

Come da tradizione, ogni tutorial che si rispetti per qualsiasi linguaggio di programmazione inizia con il programma "Hello World!". Questo tutorial non fa eccezione, ed ecco quindi la versione del programma in Hollywood:

```
Print("Hello World!")
WaitLeftMouse()
End()
```

Supponiamo di salvare lo script come "MyScript.hws" e di eseguirlo da console con: "1> Hollywood MyScript.hws". Il sistema aprira' a questo punto per default una finestra di 640x480 pixel con la scritta Hello World! La finestra rimarra' attiva finche' non verra' premuto il tasto sinistro del mouse (WaitLeftMouse) oppure selezionato il gadget di chiusura. Se volessimo dotare la nostra finestra di una i

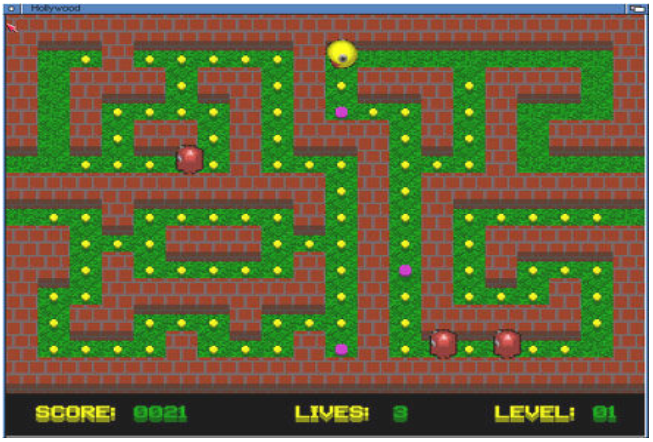


Fig.5 - Holly-Man: Pac-Man portato in Hollywood

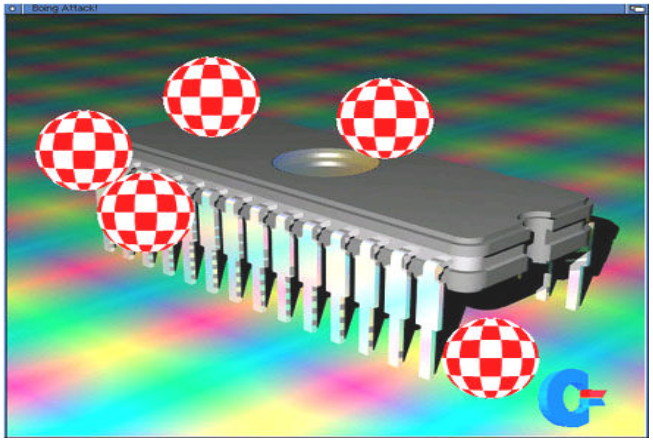


Fig.6 - Boing Balls e "rotating Commodore logo" su foto di sfondo. Sonoro a 16-bit incluso

mmagine di sfondo, molto semplicemente non dovremmo far altro che modificare il nostro script in questa maniera:

```
@BGPIC 1, "FancyBackground.jpg"
Print("Hello World!")
WaitLeftMouse()
End()
```

L'istruzione "@BGPIC" e' un "pre-processor command", cioe' un comando che pre-carica in memoria la nostra immagine assegnandola al background numero 1. E' possibile precaricare piu' backgrounds e selezionarli successivamente con l'istruzione "DISPLAY".
Per ritornare a quanto detto nel paragrafo precedente, lanciare lo script da console ne provoca la traduzione in "bytecode" e la successiva esecuzione da parte dell'interprete. In alternativa, possiamo compilare lo script selezionando la voce appropriata dal menu della GUI e produrre un pacchetto che, nel caso del secondo frammento di codice, conterra' anche la nostra foto di background. Un doppio click sull'icona del pacchetto mandera' il nostro programma in esecuzione.

Nei prossimi articoli andremo piu' in profondita' con esempi che illustrino maggiormente le spiccate caratteristiche MAL (Multimedia Application Layer) del linguaggio. Per ora limitiamoci a dire che il linguaggio, essendo essenzialmente interpretato, e' facile da programmare come l'amato/odiato BASIC ma, essendo derivato da Lua e utilizzando costrutti evoluti come le "meta-tabelle", supporta anche la programmazione ad oggetti!

Conclusioni

Come si puo' intuire da questo breve articolo, Hollywood e' un prodotto di prima scelta in grado di riportare Amiga alle sue radici... il Multimedia! Viene fornito con un compilatore che puo' salvare le vostre applicazioni come eseguibili Amiga "stand-alone" che girano senza bisogno di librerie esterne. In piu', ogni versione di Hollywood puo' scrivere eseguibili per ogni piattaforma supportata: si puo' usare una versione 68K per creare eseguibili per AmigaOS4 e viceversa! E' quindi possibile pubblicare software per OS4 senza nemmeno avere un Amiga in funzione. Dalla versione 3.0, poi, e' possibile compilare programmi anche per Microsoft Windows e Apple Mac OS. Hollywood viene consegnato su un CD-ROM che contiene oltre 50 esempi pronti all'uso che servono sia a dimostrare la potenza del linguaggio, sia a creare una sorta di libreria per gli sviluppatori che cosi' possono immediatamente iniziare a lavorare con il sistema. La documentazione a corredo e' completa ed esaustiva (oltre 1 megabyte di testo!), viene fornita in formato AmigaGuide e copre ogni aspetto del linguaggio. Programmare con Hollywood puo' sembrare complicato ma e' in realta' molto facile. I numerosi esempi e l'esistenza del forum dove gli utenti pubblicano i loro lavori sono di enorme aiuto sia al neofita che allo smanettone. Abilita' di programmatore non sono necessarie, a meno di volersi cimentare con applicazioni molto complesse. In ogni caso, Hollywood e' attualmente uno dei migliori prodotti commerciali per Amiga, se non il migliore addirittura. Sicuramente sapra' dare enormi soddisfazioni ad ogni utente. Nel prossimo tutorial entreremo piu' in dettaglio con la programmazione del sistema iniziando ad esplorare maggiormente le caratteristiche multimediali del linguaggio.

Bibliografia

Webpage dell'autore:
http://www.airsoftsoftwair.com/

Portale Hollywood ufficiale:
http://www.hollywood-mal.com/

Forum Hollywood ufficiale:
http://forums.hollywood-mal.com/

Webpage ufficiale di CodeBench:
http://codebench.co.uk/

Un ringraziamento particolare a **Gianluca Girelli** ed alla redazione di **BITPLANE** per aver concesso a RetroMagazine la possibilita' di pubblicare questo articolo tra le sue pagine.

Fig.7 - Fighter-game 2D dell'autore dell'articolo, programmato su AmigaOS4 e compilato per Windows

I file binari del DOS Atari

di Baktra
(traduzione ed adattamento a cura di David La Monaca/Cercamon)

Salve a tutti i lettori di RetroMagazine e in particolare a tutti i fan delle macchine Atari 8-bit! Ho avuto l'idea di scrivere questo breve articolo in riposta ad un post/sondaggio apparso su un forum dedicato ai computer Atari in cui si chiedeva quanto i lettori sapessero su un particolare tipo di file, i file binari. Ringrazio Cercamon e RetroMagazine per l'opportunità e spero che la lettura dell'articolo sia di vostro gradimento e che possiate imparare qualcosa in più sulle nostre meravigliose macchine Atari.

Fig.2 - L'Atari 800, primo grande successo di Atari

Panoramica

• *Scopo*

Lo scopo dei file binari di caricamento è quello di memorizzare codice macchina e dati in forma compatta, con una struttura semplice e ben definita. Nella maggior parte dei casi i file binari di caricamento vengono utilizzati per memorizzare un formato eseguibile dei programmi. L'esempio ideale di file binari di caricamento, che sfrutta completamente le caratteristiche del formato, è l'eseguibile di un gioco. File come questi in genere contengono sia il codice macchina (il motore del gioco), sia i dati di lavoro del gioco stesso (grafica, set di caratteri, display list e livelli e musica).

• *Origine*

I file binari di caricamento come li conosciamo adesso sono stati introdotti con il DOS II versione 2.0S. Anche la versione precedente del DOS di Atari supportava i file binari di caricamento, ma il loro formato era differente.

Fig.1 - L' ATARI 130XE

• *Estensione*

Non c'è un'estensione dedicata per i file binari di caricamento. Le estensioni tradizionali e più usate sono .COM, .OBJ e .SYS. Negli ultimi anni l'estensione .XEX è diventata molto popolare.

Struttura dei file binari di caricamento

• *Intestazione*

Ogni file binario di caricamento inizia con un'intestazione di soli due byte ($FF $FF). L'header serve in pratica a dire semplicemente: "Ciao, sono un file binario di caricamento. Piacere di conoscerti, Sig. Caricatore di File Binari!". Se un programma sta per elaborare un certo file binario e i suoi primi due byte non corrispondono a $FF, allora il sistema emette un messaggio di errore, di solito "ERROR 175 – BAD LOAD FILE".

• *Segmenti*

I file binari sono costituiti da uno o più segmenti (a volte chiamati anche sezioni). Un segmento rappresenta un blocco contiguo di dati da caricare in un'area specifica della memoria del computer.

• *Header di segmento*

L'intestazione del segmento è lunga 4 byte e contiene due indirizzi:
- indirizzo del primo byte del segmento (FIRST)
- indirizzo dell'ultimo byte del segmento (LAST)

L'intestazione del segmento determina l'indirizzo di memoria del computer in cui occorre memorizzare i dati contenuti del segmento stesso. Un'intestazione di segmento può essere preceduta opzionalmente da

Fig.3 - Il raro modello Atari 1200XL

due byte $FF.

• ***Dati del segmento***

I dati di ogni segmento seguono immediatamente l'header. Quelli bravi in matematica avranno già capito che la lunghezza dei dati sarà sempre LAST+FIRST+1 byte. E quindi non c'è modo di creare un segmento lungo 0 byte.

• ***Regole generali***

Non c'è una limitazione precisa sul numero di segmenti che un file binario può contenere. L'unico limite sta soltanto nella capacità dei supporti di archiviazione esterna (floppy disk o hard disk). I segmenti possono utilizzare anche indirizzi sovrapposti. Non c'è alcun impedimento nel caricare un segmento agli indirizzi $2000-$3000 e successivamente di caricarne un secondo nell'area $2100-$3100.

Caricamento di file binari

Il processo di caricamento di file binari nella memoria del computer viene chiamato Caricamento Binario. Questo processo viene eseguito dai loader (caricatori) binari, routine o programmi appositi che effettuano l'operazione di trasferire i blocchi di dati nella memoria del computer. Il DOS II contiene un loader che si attiva selezionando l'opzione L. BINARY LOAD. Anche altri sistemi operativi su disco includono i propri loader, come ad esempio Q-MEG, un sistema operativo su ROM alternativo disponibile per i computer Atari, e il Turbo BASIC XL includono loader binari.

• ***Indirizzi speciali***

Ci sono due vettori (indirizzi a 2 byte) che rivestono un significato speciale:

• ***Runad ($02E0, $02E1)***

Dopo che il loader ha caricato tutti i segmenti di un file binario (cioè quando il lavoro del loader è terminato), viene eseguito un salto indiretto - JMP ($02E0). Se il file binario da caricare contiene un codice macchina da eseguire dopo il caricamento completo, allora il file binario include un segmento da caricare in RUNAD, che contiene l'indirizzo della prima istruzione di quel determinato codice macchina.

• ***Initad ($02E2, $02E3)***

Quando il loader carica un segmento che modifica i byte del vettore INITAD, esegue un salto indiretto JMP ($02E2) ed esegue il codice macchina puntato dal vettore INITAD. Il codice macchina può restituire il controllo al loader e quando il controllo viene restituito (con l'istruzione RTS), il loader può continuare a caricare il file binario. Il vettore INITAD viene utilizzato per eseguire codice macchina durante il processo di caricamento.

Qual è la differenza fra RUNAD e INITAD? RUNAD viene utilizzato una sola volta, dopo che tutti i segmenti del file binario sono già stati caricati in memoria. INITAD viene usato tante volte quanti sono i segmenti che cambiano i byte del vettore INITAD.

• ***Segmenti RUN e INIT***

Un segmento che carica solo due byte in RUNAD viene chiamato segmento RUN. Un segmento che carica solo due byte su INITAD viene chiamato segmento INIT.

Caricamento binario – Pseudo-codice

Lo pseudo-codice seguente mostra come lavora un loader:

```
01 APRI IL FILE DI INPUT
02 CONTROLLA L'HEADER. SE L'HEADER NON E' $FF $FF ALLORA
FINE: MOSTRA ERRORE
03 REM **** CICLO PRINCIPALE ****
04 SE SIAMO ALLA FINE DEL FILE ALLORA VAI A LINEA 10
05 LEGGI HEADER DI SEGMENTO
06 LEGGI DATI DEL SEGMENTO, MEMORIZZA DATI AGLI INDIRIZZI
DATI DA HEADER
07 SE INITAD E' CAMBIATO ALLORA JMP(INITAD)
08 VAI A LINEA 04
09 REM **** FINE ****
10 JMP (RUNAD)
```

Dopo aver dato un'occhiata allo pseudo-codice, due domande si pongono immediatamente:

Che succede se non c'è un segmento RUN?

In questo caso, il loader dovrebbe semplicemente terminare l'elaborazione dopo il caricamento di tutti i segmenti contenuti nel file binario. Alcuni loader approssimativi effettuano il salto al primo indirizzo del primo segmento caricato, ma questo è di fatto sbagliato. Ricordate che i file binari senza segmento RUN possono contenere solo dati.

Come viene restituito il controllo al loader dopo l'istruzione JMP (INITAD)?

Ovviamente l'istruzione JMP non memorizza alcun indirizzo di ritorno che potrebbe essere utilizzato

dall'istruzione RTS. Purtroppo la CPU 6502 non supporta il salto indiretto ad una subroutine. Una possibile soluzione a questo problema è la seguente:

```
handle_init     JSR do_init
init_ret        JMP continue
do_init         JMP (INITAD)
```

In seguito è possibile tornare usando l'istruzione RTS, che restituirà il controllo e l'esecuzione continuerà a partire da init_ret.

Dispositivi di memorizzazione

I file binari sono stati progettati per essere caricati da dispositivi che permettono l'accesso casuale ai dati – quindi dischetti, dischi rigidi, RAM disk. In teoria, questi dispositivi sono l'ambiente naturale e "ufficiale" per il caricamento di file binari. Tuttavia, nella pratica troverete loader binari progettati espressamente per cassette o cartucce. Fortunatamente, la maggior parte dei file binari funziona con questi dispositivi "ufficialmente non supportati". Basta ricordare che le cassette e le cartucce non sono ambienti naturali per i file binari.

Esempio di file binario

Supponiamo di avere un gioco composto come segue:
- Motore del gioco (codice macchina)
- Set di caratteri (grafica)
- Display list (controllo della grafica)

Vogliamo anche implementare queste caratteristiche:
- Nessun segnale acustico durante il caricamento del nostro gioco. Il segnale acustico è fastidioso. - Visualizzare uno schermo verde durante il caricamento. Il verde è proprio bello.

Il caricamento avverrà allora in questo modo:

Intestazione del file binario:

```
FF FF
```

Header di segmento per il segmento che smette di emettere segnali acustici e visualizza uno schermo verde, caricato nell'area $4000-$400C:

```
00 40 0C 40
```

Codice macchina che disabilita il suono e imposta lo sfondo verde (si noti il codice opcode di RTS $60 posto alla fine):

```
A9 00 85 41 A5 B4 8D C6 02 8D C8 02 60
```

Intestazione del segmento INIT:

```
E2 02 E3 02
```

Dati del segmento INIT. Serve per eseguire il codice inserito in $4000:

```
00 40
```

Segmento con motore del gioco (caricato a $4000-$5C44). Non abbiamo più bisogno del codice che disabilita i segnali sonori e imposta lo sfondo verde. E' già stato eseguito, quindi perché non sovrascriverlo?

```
00 40 44 5C
89 45 A2 00 A9 28 20 0A 44 AD A5 55 AE A6 55 20
89 .. .. .. .. 60
```

Segue il segmento con il set dei caratteri, caricato in $6000-$63FF:

```
00 60 FF 63
00 00 00 00 00 00 00 00 3F 2C 1C 2D CE FF FF
CE .. .. .. .. 00
```

Poi c'è la lista di visualizzazione delle schermate, caricata nell'area $7000-700E:

```
00 70 0E 70
70 70 70 42 64 64 02 02 02 02 02 02 41 00 70
```

Ora abbiamo bisogno di aggiungere un segmento RUN, così il motore del gioco verrà eseguito dopo che tutti i segmenti saranno stati caricati. Il punto d'ingresso del motore del gioco si trova all'indirizzo $5C12:

```
E0 02 E1 02
12 5C
```

Conclusioni

La struttura di un file binario è molto semplice e logica. I vettori INITAD e RUNAD aggiungono una certa complessità ma permettono anche al file binario di diventare uno strumento estremamente potente. Un esempio di queste capacità è il gioco Ridiculous

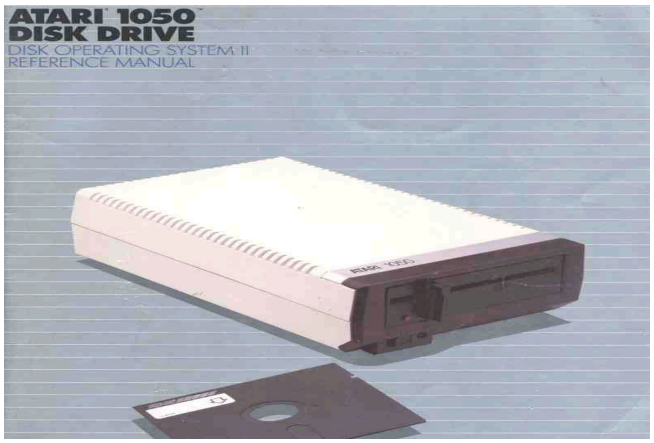


Fig. 4 - Atari 1050 Disk Drive da 5" 1/4

Reality. Si tratta di un gioco complesso, in grado di caricare nuovi livelli durante lo svolgimento del gioco, ma non è nient'altro che un singolo file binario con segmenti di tipo INIT sapientemente utilizzati.

Argomenti avanzati

Ci sono alcune caratteristiche e funzioni che ho scoperto sulla base della mia esperienza acquisita durante la scrittura di loader di file binari o di software che analizza ed elabora file binari. La struttura flessibile dei file binari lascia spazio anche ad alcuni casi limite o del tipo "cosa succede se…". Vediamone qualcuno:

Dimensione negativa del segmento

E se avete un header di segmento in cui il byte FIRST è maggiore di LAST? Il comportamento resta indefinito e dipende strettamente dal codice di un particolare loader. Alcuni loader possono visualizzare un messaggio d'errore, altri si avviluppano attorno all'indirizzo $FFFF e caricano una notevole quantità di dati. Il risultato finale tipico in queste occasioni è un completo blocco del sistema (lockup).

RUNAD e INITAD parziali

Immaginate un segmento che cambia solo un byte del vettore INITAD o di quello RUNAD. Per esempio la sequenza: E2 02 E2 02 12. Questo giustifica un salto? Da un lato il vettore è stato modificato, ma d'altro canto l'indirizzo del salto è solo parzialmente definito. Chissà cosa può succedere, l'esito è totalmente imprevedibile.

Combinare un segmento RUN con un INIT

A volte ci si può imbattere in un segmento del tipo: E0 02 E3 02 AA BB CC DD. In questo modo si combina l'impostazione dei vettori RUNAD e INITAD. Il comportamento è indefinito, ma di solito funziona esattamente come previsto.

Segmenti RUN multipli

Un file binario può contenere più segmenti che modificano il vettore RUNAD. Quel che succede è che dopo aver caricato tutti i segmenti disponibili, il loader salta all'ultimo indirizzo RUNAD trovato.

Caricamento di file binari con piccoli difetti

A volte si possono riscontrare due tipici difetti nei file binari:

1. Dati del segmento oltre la fine del file binario – Supponiamo che l'ultimo segmento del file binario debba essere lungo N bytes secondo l'intestazione del segmento, ma che vi siano soltanto M bytes (con M < N) rimasti nel file. Un loader minuzioso mostrerebbe un messaggio d'errore adeguato. Tuttavia i loader di solito leggono i dati fino alla fine del file. Ciò comporta due spiacevoli conseguenze: a) Il file è danneggiato ma il sistema non lo comunica all'utente – b) Si sta effettivamente utilizzando un file danneggiato. Immaginate cosa succederebbe durante una partita ad un gioco che inevitabilmente si bloccherà quando siete all'ultimo livello a causa del file corrotto. L'opzione migliore è ovviamente quella di sbarazzarsi del file e procurarsene una versione corretta e funzionante. Se possibile, bisognerebbe anche contattare l'autore del programma (se si tratta di un titolo rilasciato di recente) e comunicargli l'accaduto chiedendo delle correzioni.

2. Dati estranei alla fine del file binario – Immaginate che l'ultimo segmento del file binario sia seguito da alcuni dati "spazzatura" che non costituiscano un segmento valido. Tipicamente accade che degli zeri extra siano presenti alla fine del file come risultato di un'estrazione di una cartuccia. Se siamo in presenza di un loader meticoloso, il sistema mostrerebbe un messaggio d'errore. Un loader comune cercherebbe semplicemente di elaborare i dati nel miglior modo possibile. Il comportamento che ne segue è sostanzialmente indefinito e dipende molto dai dati spazzatura che si trovano in coda al file. Questi potrebbero infatti costituire un segmento valido (anche se indesiderato) oppure confondere del tutto il loader. Questo tipo di difetto può rivelarsi innocuo oppure (nel caso peggiore) causare un blocco totale del sistema. La soluzione migliore è quella di sbarazzarsi del file. In alternativa si può tentare di recuperare il file rimuovendo i dati spazzatura con un editor esadecimale.

Classificazione dei file binari

La classificazione che segue non è affatto autorevole. L'ho inventata io stesso nella speranza che torni utile a qualcuno (e a me). In effetti la mia classificazione riflette la mia esperienza di compatibilità con il caricamento da nastro.

– Monolitico

Questi file binari contengono un segmento dati ed un segmento RUN. La loro struttura è molto semplice e lineare e non ci sono segmenti INIT. I file binari di tipo monolitico possono essere facilmente caricati da qualsiasi dispositivo di memorizzazione. La compressione di questi file tende ad essere conveniente e più efficace che per altri tipi. Funziona alla grande con il caricamento da cassetta.

– Monolitico con segmento INIT

Come i file monolitici, ma possono contenere un segmento INIT. Si riscontra la maggior parte dei vantaggi che si hanno con i file di tipo monolitico. Anche questi file si caricano facilmente da nastro e somigliano a dei file di boot.

– Segmentato senza segmenti INIT

I file binari di questo tipo possono contenere molti

Fig. 5 - Schermata principale del DOS 2.5

segmenti, ma nessun segmento INIT. Anche questi sono facili da caricare via nastro e hanno un alto tasso di efficienza se compressi.

– Segmentato

File binari senza alcuna restrizione.

Metodi ottimali

Secondo me, i seguenti costituiscono i metodi ottimali per creare file binari.

• Tenere a mente l'ambiente d'uso:

meglio progettare il file binario in modo che sia adatto all'ambiente in cui verrà utilizzato.

• Mantenere il DOS :

se il programma ha bisogno del DOS, non caricate i segmenti in aree di memoria usate dal DOS ed evitate che il programma stesso distrugga il funzionamento del sistema operativo. Inoltre aggiungete al programma un'opzione di uscita al DOS.

• Liberarsi del DOS :

se il programma dev'essere caricabile sotto DOS, ma dopo il caricamento ha bisogno della RAM occupata che esso occupa per le sue esigenze, allora è meglio non caricare segmenti in aree utilizzate dal DOS. Dopo che il programma si sarà trasferito in memoria si può fare quello che si vuole. Se sovrascrivete l'area del DOS, date la possibilità all'utente di abbandonare il programma attraverso COLD START, così non è costretto a spegnere la macchina.

• MiniDOS :

se il programma binario consuma tanta memoria anche durante il caricamento, è consigliabile utilizzare un DOS oppure un loader in miniatura. Cercate MiniDOS, MicroDOS, MyPicoDOS o XBIOS per implementarne uno per il vostro programma.

• Cassetta :

se volete che il vostro programma si possa caricare da nastro, allora evitate qualunque azione che possano rovinare il caricamento. Quando il vostro codice sta armeggiando con PACTL assicuratevi di non spegnere il motore del registratore accidentalmente. Meglio scegliere un loader adatto per il caricamento da nastro, che supporti la struttura del file binario. Se ci sono segmenti INIT, meglio essere certi che il loader per il nastro li supporti correttamente. E' necessario anche controllare se il loader ha bisogno di routine presenti in ROM o meno. E, per fare un favore agli utenti con registratore a cassette, provate a comprimere il file binario.

• BASIC :

è buona regola disattivare il BASIC automaticamente. Se il programma funziona solo con il BASIC disabilitato, la cosa migliore è iniziare il caricamento con un segmento che disabilita il BASIC e proseguire con un segmento INIT per poi eseguirlo. Gli utenti XL/XE lo adoreranno, perché il mylar sotto il tasto OPTION durerà più a lungo. Quando si vuole disabilitare il BASIC è meglio farlo nel modo giusto: disattivare la ROM BASIC, resettare il flag del BASIC, regolare l'indirizzo RAMTOP e quindi riaprire il dispositivo E:. In questo modo l'utilizzo della memoria schermo è in linea con il BASIC disabilitato.

La struttura

È importante strutturare il file binario per soddisfare le esigenze del programma. Evitate i casi limite e le piccole imprecisioni. E ricordate, inserite meno segmenti possibili. Naturalmente se il vostro programma ha bisogno di più segmenti, allora usateli pure. Basta che non ne abusiate o dividiate ulteriormente i segmenti quando non è necessario.

Non abusare del loader

Il loader può essere usato impropriamente per fare cose terribili. A volte i risultati possono essere davvero inaspettati.

Abuso #1 – Vertical Blank (VBLANK)

Supponiamo che si voglia impostare una display list (il

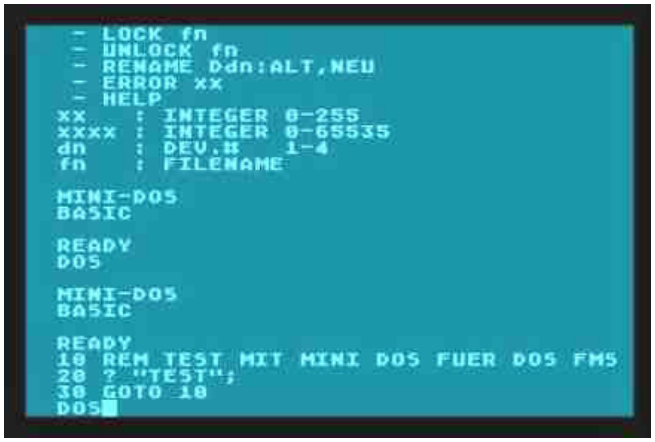


Fig.6 - Schermata d'uso del MiniDOS per Atari 8-bit

processore grafico delle macchine a 8-bit supporta questa particolare funzione per disegnare sullo schermo). Una possibilità è quella di caricare un segmento con una display list e poi caricare un altro segmento che cambi il vettore SDLSTL in modo che punti alla Display List. Andrebbe bene procedere in questo modo? No, non lo è. Per evitare risultati imprevedibili, il vettore SDLSTL dovrebbe essere cambiato solo durante il blank verticale. Ma i loader non aspettano il blank, quindi è meglio scrivere un pezzo di codice che attenda il VBLANK, poi imposti il vettore SDLSTL e infine utilizzi un segmento INIT per eseguire il codice. Questa tecnica è applicabile a tutte le azioni che richiedono l'attesa di un blank verticale.

Abuso #2 – Area di I/O e Pagina Zero

Un altro esempio di possibile abuso è il caricamento di segmenti nell'area di I/O ($D000-D7FF). Il caricamento di dati dei segmenti in quell'area potrebbe influenzare il dispositivo di storage da cui viene caricato lo stesso file binario. Molto meglio scrivere un pezzo di codice che apporti le modifiche e usare un segmento INIT per eseguire il codice. Inoltre non è affatto saggio caricare i segmenti nella Pagina Zero della memoria. Non si sa mai quali indirizzi in Pagina Zero vengono usati dal sistema operativo o dal loader.

Abuso #3 – Solo teorico

L'ultimo esempio di abuso è solo teorico. Immaginate di caricare un segmento nell'area I/O o in uno degli indirizzi RTCLOCK. Ciò che si scrive in questi indirizzi potrebbe non essere possibile rileggerlo in un secondo momento (a causa della natura stessa dell'area I/O o perché il sistema operativo cambia i byte in memoria). E se il loader calcola un checksum dopo aver messo in memoria tutti i dati dei segmenti, allora il controllo di checksum fallirà, perché i dati letti non sono più gli stessi. Può accadere solo in teoria? Sì, la maggior parte dei dispositivi utilizza un buffer o calcola il checksum prima di mettere i dati in memoria. Ma provate a farlo con il sistema di caricamento da nastro Turbo ROM PLUS… O, meglio, non fatelo!

I file binari sono preziosi

Se la natura del programma lo permette, è una buona idea distribuirlo come file binario. È una forma molto valida di distribuzione, perché se si dispone di un file binario, si può facilmente convertirlo in una versione per cassetta, disco avviabile o caricarlo da una delle moderne cartucce universali. La conversione verso altri formati è sempre più articolata.

Baktra è un programmatore ceco appassionato di sistemi Atari 8-bit. Potete consultare il sito web

http://baktra.webowna.cz

per saperne di più sui suoi lavori Atari e sulle sue attività nel mondo del retrocomputing.

Riferimenti

– Atari 1050 unità a dischi - Manuale per sistema operativo DOS II, Atari Publishing

– EXE File Format
https://www.atarimax.com/jindroush.atari.org/afmtexe.html

– Ridiculous Reality
https://www.rgcd.co.uk/2012/11/ridiculous-reality-atari-xexl.html

– MiniDOS
https://atariwiki.org/wiki/Wiki.jsp?page=MiniDOS

– Vertical Blank e Display List
https://www.atariarchives.org/agagd/chapter6.php

"Piange il telefono"

di Marco Pistorio

Corre l'anno 1993. E' un lunedì. Mario chiama l'amico Giuseppe. La chiamata telefonica dura esattamente due ore, dalle 12:00 alle 14:00. I due si trovano ad una distanza di circa 200 km. Il costo, orientativamente, di questa chiamata? Circa 64.643 di vecchie lire. All'indirizzo che segue:

https://www.infodata.ilsole24ore.com/2016/05/17/calcola-potere-dacquisto-lire-ed-euro-dal-1860-2015/

potete provare a valutare cosa significhi una cifra simile rapportata ad oggi.
Digitando 64643 lire ed impostando l'anno 1993 otterrete: Euro 53,70(!)
Un prezzo modico, non c'è che dire...
Come faccio a sapere che la chiamata costerebbe tanto?

Rovistando nei miei dischetti ho trovato un programma che scrissi personalmente in quegli anni sul mio fido Commodore 64 e che troverete insieme a questo articolo, articolo dove cercherò di ricordare insieme a voi questi ed altri "deliri" di allora!

Oggi siamo connessi praticamente 24 ore su 24, le tariffe che scegliamo normalmente sono flat, ma non è sempre stato così.
Cosa succedeva nei primi anni '90? Chi è la "famigerata" S.I.P.?
E' la compagnia telefonica unica di quegli anni in Italia. Fu attiva dal 1964 al 1994 come "Società Italiana Per l'esercizio delle telecomunicazioni", subito dopo la nazionalizzazione della industria elettrica italiana. La S.I.P. nacque infatti nel 1899 come azienda di produzione elettrica. S.I.P. era l'acronimo, originariamente, di "Società Idroelettrica Piemontese".
Successivamente, la S.I.P. divenne Telecom Italia SpA.
Telecom Italia nacque formalmente il 27 luglio 1994, con l'atto di fusione deliberato dalle assemblee del 19 maggio dello stesso anno di SIP con Iritel, Telespazio, Italcable e SIRM, società del gruppo STET già operative nel settore delle telecomunicazioni. Telecom Italia SpA operò fino al 26 Maggio 2016. A questa data diventa TIM SpA, ed opera a tutt'oggi con questo nome.
Ma torniamo ora a quei "verdi anni". Negli anni '90 la telematica in Italia iniziava a muovere i suoi primi passi.
Il sistema telefonico nazionale era passato dal sistema decadico a quello multifrequenza. Nel numero 4 di RetroMagazine troverete un articolo che riguarda proprio questo argomento.
La telefonia stava "migrando" gradualmente dall'analogico al digitale. Internet era già nato da molto tempo ma era un pò troppo acerbo, ben diverso da quello che conosciamo oggi.
Il protocollo HTTP, il "protocollo per il trasferimento dell'ipertesto" nasce alla fine degli anni '80 ed inizia a diffondersi.
Sono già molto diffusi da noi i BBS.
Un BBS (acronimo di Bulletin Board System) è un computer che utilizza un particolare software per permettere ad utenti esterni di connettersi ad esso attraverso la linea telefonica, dando la possibilità di utilizzare funzioni di messaggistica, file sharing centralizzato, ma anche giochi di ruolo testuali semplici, denominati M.U.D. (Multi Users Dungeons).
Per collegarsi ad un BBS gli appassionati erano costretti ad effettuare delle chiamate telefoniche che potevano rivelarsi parecchio costose, soprattutto quando il BBS si trovava in città diverse oppure in regioni diverse da quella dalla quale partiva la chiamata.
Ho già scritto qualcosa nei precedenti numeri di RetroMagazine riguardo i BBS. Invito gli interessati a rileggere ad esempio l'articolo "B.B.S. - Ritorno al passato" nel prececente numero 9 di RetroMagazine.
Appartenendo anch'io a questa categoria di appassionati, divenne necessario anche per me tenere sotto controllo i costi della bolletta telefonica.

I parametri che determinavano il costo delle chiamate erano: il giorno (feriale/prefestivo/festivo), la fascia oraria, la durata complessiva e la distanza tra il chiamante ed il chiamato.
La durata della conversazione è un elemento chiave.
Più dura la conversazione, più paghi, per esprimere il concetto in maniera semplificata.
Il sabato e la domenica il costo era normalmente ridotto.
Durante gli altri giorni della settimana era più alto e, all'interno della stessa giornata, alcune fasce orarie erano più costose di altre.
Veniva individuata una fascia dalle 08:00 alle 08:30, una dalle 08:30 alle 13:00, una dalle 13:00 alle 18:30, una dalle 18:30 alle 22:00 ed infine una dalle 22:00 fino alle 08:00 del mattino seguente.
Di notte le chiamate costavano meno mentre la fascia dalle 08:30 alle 13:00 era quella più calda, in tutti i sensi!
Le chiamate urbane, cioè quelle effettuate all'interno dello stesso centro urbano ovvero della stessa città erano quelle meno costose. Per questa tipologia di chiamate veniva applicata una tariffa denominata "T.U.T." ovvero Tariffa Urbana a Tempo.

Le chiamate interurbane sono invece chiamate tra centri urbani diversi ma aventi lo stesso prefisso. Qui i costi iniziano a salire.
Quando il prefisso del numero del chiamato è diverso da quello del chiamante parliamo di chiamate in teleselezione. Ecco il motivo per cui il prefisso telefonico veniva chiamato fino a relativamente poco tempo fa "prefisso teleselettivo", prima di diventare parte integrante del numero di telefono dell'abbonato, come è oggi.
Per la teleselezione vi erano diversi scaglioni basati sulla distanza tra i due estremi collegati.
Gli scaglioni erano: distanza entro i 15 Km, tra i 15 ed i 30 Km, tra i 30 ed i 60 Km, tra i 60 ed i 120 Km ed infine oltre i 120 Km.
Le chiamate in teleselezione erano normalmente le più "sanguinose".
La suddivisione territoriale in strutture urbane, settoriali, distrettuali, compartimentali, utilizzata come riferimento per il sistema tariffario, era dipendente dall'organizzazione funzionale e gerarchica della rete telefonica di tipo analogico. Oggi tale suddivisione è stata superata, tuttavia occorre conoscerla per comprendere la logica sulla quale si basa il calcolo del costo delle chiamate.
Il programma chiede il giorno della settimana in cui viene effettuata la chiamata, l'ora di inizio e di fine della chiamata (espressi entrambi nel formato ore, minuti e secondi), se trattasi di chiamata urbana, interurbana settoriale oppure in teleselezione, ed in quest'ultimo caso la distanza in km tra i due centri, e ne calcola il costo.
Tale costo è da considerarsi al netto di IVA, senza scatti residui ed è un costo in lire, anche perchè all'epoca della stesura di questo programma l'euro non esisteva ancora.
Il risultato dei calcoli non è ottenuto immediatamente in quanto scelsi un algoritmo basato su continue sottrazioni, algoritmo che è funzionale allo scopo ma decisamente lento.
Provate ad effettuare un paio di calcoli.
Vi renderete conto con quanta facilità il costo delle chiamate telefoniche diventava esorbitante.
Una famosa pubblicità della SIP di quegli anni con Massimo Lopez recitava: "Una telefonata...allunga la vita".
Per chi di voi volesse rinfrescarsi la memoria ecco un link relativo al video:

https://www.youtube.com/watch?v=DkbFv8qxMXs

Che ne pensate? Uno dei tanti casi di pubblicità ingannevole? Lascio a voi il giudizio :)

Ciao a tutti, amici lettori e...buone ferie!

```
10 rem ********************************
15 rem *                              *
20 rem *    calcolo costo chiamata    *
25 rem *                              *
30 rem *            urbana            *
35 rem *    interurbana settoriale    *
40 rem *        teleselezione         *
45 rem *                              *
50 rem *  written by marco pistorio   *
55 rem *                              *
60 rem * original soft  (c) 14/11/93  *
65 rem *                              *
70 rem ********************************
75 :
80 rem setta lo schermo e i colori
85 poke53280,0:poke53281,0:poke646,5
90 printchr$(147);
95 ex=0:ns=1
100 rem caricamento dati interni
101 data 0,28800,30600,46800,66600
102 data 79200,86400
103 for cl=1to7:read o(cl):next
104 :
105 rem inserimento dati esterni
106 print"giorno della settimana in cui"
107 print"e' stata effettuata la chiamata:";
108 input g$
109 input"inizio chiamata (hh-mm-ss):";oi$
110 input"fine chiamata   (hh-mm-ss):";of$
111 print:print
112 printtab(11)"- tipo di chiamata -
113 printtab(4)"1. urbana (t.u.t.)"
114 printtab(4)"2. interurbana settoriale"
115 printtab(4)"3. teleselezione"
116 print:print:print"scelta ?";
117 getsc$:ifsc$=""then117
118 k=val(sc$):ifk=0 or k>3 then 117
119 print sc$
120 if k=3 then input"distanza in km:";ds
121 :
125 rem calcolo corrispettivo s.i.p.
130 :
135 rem conversione orari in secondi
140 hi=val(left$(oi$,2))
143 mi=val(mid$(oi$,4,2))
145 si=val(right$(oi$,2))
147 hf=val(left$(of$,2))
150 mf=val(mid$(of$,4,2))
153 sf=val(right$(of$,2))
155 oi=si+(mi*60)+(hi*3600)
160 of=sf+(mf*60)+(hf*3600)
165 :
170 if g$="sabato" then 300
180 if g$="domenica" then 400
190 :
195 rem assegnazione fasce orarie (1)
200 if oi>=o(1) and oi<o(2) then fs=4:mx=o(2)
201 if oi>=o(2) and oi<o(3) then fs=2:mx=o(3)
202 if oi>=o(3) and oi<o(4) then fs=1:mx=o(4)
203 if oi>=o(4) and oi<o(5) then fs=2:mx=o(5)
204 if oi>=o(5) and oi<o(6) then fs=3:mx=o(6)
205 if oi>=o(6) and oi<=o(7) then fs=4:mx=o(7)
206 gosub 500:if ex=1 then 1000
207 goto 200
299 :
300 rem assegnazione fasce orarie (2)
301 if oi>=o(1) and oi<o(2) then fs=4:mx=o(2)
302 if oi>=o(2) and oi<o(4) then fs=2:mx=o(4)
303 if oi>=o(4) and oi<o(6) then fs=3:mx=o(6)
304 if oi>=o(6) and oi<=o(7) then fs=4:mx=o(7)
305 gosub 500:if ex=1 then 1000
306 goto 300
399 :
400 rem assegnazione fasce orarie (3)
401 if oi>=o(1) and oi<o(2) then fs=4:mx=o(2)
```

```
402 if oi>=o(2) and oi<o(6) then fs=3:mx=o(6)
403 if oi>=o(6) and oi<=o(7) then fs=4:mx=o(7)
404 gosub 500:if ex=1 then 1000
405 goto 400
499 :
500 on k gosub 600,700,800
505 return
599 :
600 rem corrispettivi t.u.t.
605 if of>mx then 610
608 o=of-oi:ex=1:goto 615
610 o=mx-oi
615 if fs=1 then fc=240
616 if fs=2 then fc=300
617 if fs=3 then fc=400
618 if fs=4 then fc=600
619 sc=o/fc:if sc<>int(sc) then sc=int(sc)+1
620 oi=oi+sc*fc:ns=ns+sc
621 if oi>o(7) then oi=o(7)-oi
625 return
699 :
700 rem corrispettivi interurbane sett.
705 if of>mx then 710
708 o=of-oi:ex=1:goto 715
710 o=mx-oi
715 if fs=1 then fc=84
716 if fs=2 then fc=120
717 if fs=3 then fc=168
718 if fs=4 then fc=240
719 sc=o/fc:if sc<>int(sc) then sc=int(sc)+1
720 oi=oi+sc*fc:ns=ns+sc
721 if oi>o(7) then oi=o(7)-oi
725 return
799 :
800 rem corrispettivi teleselezione
801 if of>mx then 803
802 o=of-oi:ex=1:goto 804
803 o=mx-oi
804 if ds>=0 and ds<15 then j=1
805 if ds>=15 and ds<30 then j=2
806 if ds>=30 and ds<60 then j=3
807 if ds>=60 and ds<120 then j=4
808 if ds>120 then j=5
809 on j goto 900,910,920,930,940
810 sc=o/fc:if sc<>int(sc) then sc=int(sc)+1
811 oi=oi+sc*fc:ns=ns+sc
812 if oi>o(7) then oi=o(7)-oi
813 return
899 :
900 rem teleselez. fino 15 km
901 if fs=1 then fc=35
902 if fs=2 then fc=72
903 if fs=3 then fc=96
904 if fs=4 then fc=144
905 goto 810
910 rem teleselez. 15-30 km
911 if fs=1 then fc=24
912 if fs=2 then fc=40
913 if fs=3 then fc=52.5
914 if fs=4 then fc=80
915 goto 810
920 rem teleselez. 30-60 km
921 if fs=1 then fc=15
922 if fs=2 then fc=22.5
923 if fs=3 then fc=35
924 if fs=4 then fc=45
925 goto 810
930 rem teleselez. 60-120 km
931 if fs=1 then fc=12.5
932 if fs=2 then fc=20
933 if fs=3 then fc=32
934 if fs=4 then fc=40
935 goto 810
940 rem teleselez. oltre 120 km
941 if fs=1 then fc=11.5
942 if fs=2 then fc=18.5
943 if fs=3 then fc=29.8
944 if fs=4 then fc=37
945 goto 810
1000 printchr$(147);
1010 print"---------------------------------------"
1011 print:
1012 print"numero scatti effettuati:";ns
1013 print"corrispettivo: lire ";ns*127
1014 print
1015 print"---------------------------------------"
```

Calcolare in multipla precisione-parte II

di Alberto Apostolo

Benvenuti alla seconda parte del percorso riguardante i calcoli in multipla precisione o precisione arbitraria, che affronterà il calcolo della funzione exp(x) ossia "e alla x" dove ***e*** rappresenta la nota costante di Eulero 2.718281828... un numero non meno nobile del più famoso pi-greco.
Nella parte I (RM 15) sono state spiegate le ragioni per le quali è utile il calcolo in multipla precisione (Crittografia , simulazione di sistemi fisici e chimici) e sono state anche presentate (per fini didattici) alcune semplici ma fondamentali routine scritte in C da utilizzare per piccoli esperimenti di programmazione con il compilatore freeware Pelles C.
Se si desidera adoperare un altro compilatore, occorre accertarsi che sia almeno "C99 compliant" affinché sia possibile dichiarare variabili intere a 64 bit di tipo "long long int".

Breve storia della costante di Eulero

Nel 1618, durante lo studio sui logaritmi, John Napier (Nepèro) si imbatte in una costante poi denominata ***e*** da Leonhard Euler (Eulero) nel 1727. Nel 1737 Eulero dimostra che ***e*** è irrazionale e nel 1748 calcola le prime 23 cifre decimali [BBB04].
Nel 1873 Charles Hermite prova che ***e*** è trascendente (ovvero non può essere la soluzione di nessuna equazione algebrica a coefficienti interi). Nel giugno del 1949, lo scienziato John Von Neumann e il suo gruppo di collaboratori ottengono 2010 cifre decimali dopo 11 ore di elaborazione effettuate dal calcolatore elettronico ENIAC (fig. 1). Nel 2007 sono state calcolate le prime 10^11 cifre decimali di ***e***.

e = 2.71828 18284 59045 23536 02874 71352 66249 77572 47093 69995
95749 66967 62772 40766 30353 54759 45713 82178 52516 64274
27466 39193 20030 59921 81741 35966 29043 57290 03342 95260
59563 07381 32328 62794 34907 63233 82988 07531 95251 01901
15738 34187 93070 21540 89149 93488 41675 09244 76146 06680
82264 80016 84774 11853 74234 54424 37107 53907 77449 92069
55170 27618 38606 26133 13845 83000 75204 49338 26560 29760
67371 13200 70932 87091 27443 74704 72306 96977 20931 01416
92836 81902 55151 08657 46377 21112 52389 78442 50569 53696
77078 54499 69967 94686 44549 05987 93163 68892 30098 79312
77361 78215 42499 92295 76351 48220 82698 95193 66803 31825
28869 39849 64651 05820 93923 98294 88793 32036 25094 43117
30123 81970 68416 14039 70198 37679 32068 32823 76464 80429
53118 02328 78250 98194 55815 30175 67173 61332 06981 12509
96181 88159 30416 90351 59888 85193 45807 27386 67385 89422
87922 84998 92086 80582 57492 79610 48419 84443 63463 24496
84875 60233 62482 70419 78623 20900 21609 90235 30436 99418
49146 31409 34317 38143 64054 62531 52096 18369 08887 07016
76839 64243 78140 59271 45635 49061 30310 72085 10383 75051
01157 47704 17189 86106 87396 96552 12671 54688 95703 50354
02123 40784 98193 34321 06817 01210 05627 88023 51930 33224
74501 58539 04730 41995 77770 93503 66041 69973 29725 08868
76966 40355 57071 62268 44716 25607 98826 51787 13419 51246
65201 03059 21236 67719 43252 78675 39855 89448 96970 96409
75459 18569 56380 23637 01621 12047 74272 28364 89613 42251
64450 78182 44235 29486 36372 14174 02388 93441 24796 35743
70263 75529 44483 37998 01612 54922 78509 25778 25620 92622
64832 62779 33386 56648 16277 25164 01910 59004 91644 99828
93150 56604 72580 27786 31864 15519 56532 44258 69829 46959
30801 91529 87211 72556 34754 63964 47910 14590 40905 86298
49679 12874 06870 50489 58586 71747 98546 67757 57320 56812
88459 20541 33405 39220 00113 78630 09455 60688 16674 00169
84205 58040 33637 95376 45203 04024 32256 61352 78369 51177
88386 38744 39662 53224 98506 54995 88623 42818 99707 73327
61717 83928 03494 65014 34558 89707 19425 86398 77275 47109
62953 74152 11151 36835 06275 26023 26484 72870 39207 64310
05958 41166 12054 52970 30236 47254 92966 69381 15137 32275
36450 98889 03136 02057 24817 65851 18063 03644 28123 14965
50704 75102 54465 01172 72115 55194 86685 08003 68532 28183
15219 60037 35625 27944 95158 28418 82947 87610 85263 98139
55990 06738

Figura 1

Come calcolare exp(x)

Il calcolo di exp(x) si esegue per mezzo della sua serie di Taylor convergente

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Figura 2

per ogni valore reale di x (fig.2). Ovviamente si sommeranno solo N termini affinché il termine N-esimo sia molto piccolo e la somma calcolata sia "vicina" al vero valore di exp(x) quanto si desidera. Il valore x sarà in multipla precisione e quindi saranno necessarie due routine: la prima per inserire un numero in multipla precisione da tastiera e la seconda per moltiplicare tra loro due numeri in multipla precisione.

Input di una variabile in M.P.

La routine Inpx.C permette di assegnare un numero in multipla precisione come nelle calcolatrici tascabili. Oltre a digitare le cifre da 0 a 9, si può cancellare una cifra premendo il tasto

"C" , cambiare di segno con "-", abilitare la virgola con ".". Una volta digitato il numero, si preme "Q" per uscire dalla routine e proseguire con il resto del programma.
La pressione di un tasto è gestita dalla routine LeggiTasto.C . Le altre routine richiamate all'interno di Inpx.C si trovano sul numero 15 di RM (cfr. parte I).

La moltiplicazione lunga in m.p.

La molitplicazione lunga consiste nel moltiplicare un numero a per un numero b composto da più di una cifra. Un esempio del modo di procedere è mostrato in fig. 3. Tale

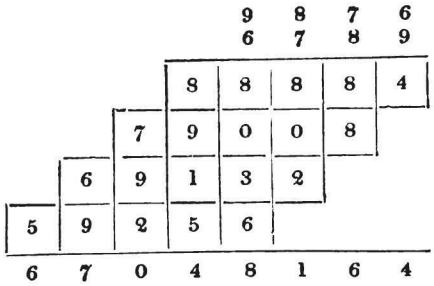


Figura 3

metodo era chiamato "scachero" a Venezia (per una certa somiglianza con il gioco degli scacchi)."bericuocolo" a Firenze e a Verona, "organetto" in altre città d'Italia [EPM47]. Per i curiosi, il bericuocolo era un dolcetto a base di spezie e pasta di albicocche. I bericuocoli erano confezionati in scatole suddivise in scomparti quadrati.
Tartaglia è stato uno dei primi matematici a usare quel metodo senza i numeri racchiusi nei quadrati. A partire dal secolo XVI il metodo si è diffuso in Aritmetica fino ad escludere tutti gli altri metodi allora conosciuti.
La routine Mmpx.C esegue la moltiplicazione tra due variabili in multipla precisione nel modo classico.
Lo svolgimento del prodotto avviene memorizzando il risultato parziale in un buffer di cifre dimensionato al doppio delle variabili in m.p.. Alla fine si controlla se si è verificato un overflow (le cifre eccedenti della parte intera devono essere zero) e

```
/*-------------------------------------------------------------------------*/
/* Funzione: digitazione di un numero in m.p.                              */
/*-------------------------------------------------------------------------*/
void Inpx(long beta,long l,long lpi,long lv,MultPrec *p_x,char *nomevbl)
{     long i,k,m;              /* indici (vbl di lavoro)                   */
      long z=10L;              /* vbl di lavoro                            */
      long f=1L;               /* cifre in base beta                       */
      long flag_dec = 0L;      /* 1=punto decimale attivo, 0=disatt.       */
      long cont_dec = 0L;      /* conta cifre dopo il punto decimale       */
/*                                                                         */
      int kbint = 0;           /* memorizza tasto premuto                  */
      long rrp=0L, *p_rrp=&rrp;/* riporto e puntatore riporto              */
      long rrd=0L, *p_rrd=&rrd;/* resto   e puntatore resto                */
      MultPrec  w;             /* variabile multipla precisione            */
      MultPrec  *p_w =&w;      /* puntatore var. mult.prec.                */

      while (z<beta){f++;z=z*10L;}
      Init(beta,l,lpi,p_w,0L,p_rrp);
      while((kbint != 'q') && (kbint != 'Q')){       /*q,Q per uscire */
          _clrscr();
          for (i = l;i >= 1L;i--) {
                           if ((*p_w).cifra[i] > 0L) {
                              if (w.segno == 0L) {w.segno = +1;}
              }
          }
          DisplayMultPrec(lv,lpi,p_w,nomevbl);
          printf("\n\n       (Q,q)= esci, (C,c)= cancella una cifra");
          printf(", (.)= punto decimale, (-) cambia segno\n\n");
          kbint = LeggiTasto();
          if (kbint == '.') {flag_dec = 1;}
          if (kbint == '-') {w.segno = -w.segno;}
          if (flag_dec == 0) {
             if ((kbint >= '0') && (kbint <= '9')
                               && (w.cifra[1] < (beta/10L)) ) {
                Mulz(beta,l,p_w,10,p_rrp);
                w.cifra[lpi] = w.cifra[lpi] + kbint - 48;
             }
             if ((kbint == 'c') || (kbint == 'C')) {
                Divz(beta,lpi,p_w,10,p_rrd); /* si parte da lpi */
             }
          } else {
             if ((kbint >= '0') && (kbint <= '9')
                               && (cont_dec < ((lv-lpi)*f)) ) {
                cont_dec++;
                z = (cont_dec-1) / f;
                k = lpi + z + 1;
                m = f - (cont_dec-1) + z*f;
                z=1;
                for (i = 1L;i<m;i++) {z=z*10L;}
                w.cifra[k] = w.cifra[k] + (kbint - 48)*z;
             }
             if ((kbint == 'c') || (kbint == 'C')) {
                z = (cont_dec-1) / f;
                k = lpi + z + 1;
                m = f - (cont_dec-1) + z*f;
                z=10;
                for (i = 1L;i<m;i++) {z=z*10L;}
                w.cifra[k] = w.cifra[k] - (w.cifra[k] % z);
                cont_dec--;
                if (cont_dec == 0){flag_dec=0;}
             }
          }
      }
      (*p_x) = w;
}
/* Funzione realizzata : restituisce codice ASCII tasto premuto       */
int LeggiTasto(void)
{    int c = 0;                  /* carattere                          */
      while (_kbhit()) _getch(); /* finche' ci sono caratteri          */
                                 /* li legge e li ignora               */
      c = _getch();              /* legge un tasto                     */
      return c;
}
```

Inpx.C

si assegna il prodotto e il segno alla variabile in m.p. che dovrà contenere il risultato.
La complessità computazionale è quadratica (cioè proporzionale a n^2 se n è la lunghezza dei due fattori). Naturalmente esistono metodi più efficienti come il metodo di Karatsuba (di complessità n^1.5) e il metodo FFT (Fast Fourier Transform di complessità n*log n) ma non saranno trattati in questa sede.

Il programma MP_EXPX

Il programma MP_EXPX esegue il calcolo della funzione exp(x). Dopo avere digitato la variabile x richiesta, si passa alla fase di calcolo e infine a mostrare il risutato finale. Il calcolo si interrompe se durante il ciclo si verificheranno degli overflow. La complessità computazionale è cubica (proporzionale a n^3 dove n è la lunghezza delle variabili m.p.).

La prossima volta...

...Si calcolerà il reciproco di un numero x in multipla precisione e la radice quadrata di un numero in multipla precisione.

```
/*-------------------------------------------------------------------*/
/* Funzione: moltiplicazione tra numeri in m.p. c = a * b            */
/*-------------------------------------------------------------------*/
void Mmpx(long beta,long l,long lpi,MultPrec *p_c
                           ,MultPrec *p_a,MultPrec *p_b,long *p_ovf)
{
      long cifra[(2*l)+1];     /* buffer cifre in base beta          */
      long i,j,k;              /* indici (vbl di lavoro)             */
      long rr;                 /* vbl riporto                        */
      long izer = 0L;          /* flag = 1 se cifra<>0, 0 altrimenti */
      long long int ipro;      /* vbl lavoro                         */

      for (i=(l+1L);i<=(2L*l)+1;i++){cifra[i]=0L;}
      *p_ovf=0L;
      for (i = l;i >= 1L;i--) {
         rr=0L;
         if ((*p_b).cifra[i]>0L) {
            for (j = l;j >= 1L;j--) {
               k=i+j;
               ipro=(*p_b).cifra[i];              /*workaround PellesC*/
               ipro=rr+cifra[k]+((*p_a).cifra[j] * ipro);
               rr=ipro/beta;
               cifra[k]=ipro % beta;
            }
         }
         cifra[i]=rr;
      }
      for (i = 1L;i <= lpi;i++) {
         if (cifra[i]>0L) {
            *p_ovf=1L;
            break;
         }
      }
      for (i = 1L;i <= l;i++) {
         (*p_c).cifra[i]=cifra[i+lpi];
         if ((*p_c).cifra[i]>0L) { izer=1L; }
      }
      (*p_c).segno=(*p_a).segno * (*p_b).segno * izer;
}
```

Mmpx.C

Appendice

Si segnala un refuso presente nella parte I (RM 15). Nei paragrafi "Moltiplicazione corta variabili m.p." e "Divisione corta variabili m.p.", "10N" in realtà voleva indicare "10^N". Ci scusiamo con i lettori.

Bibliografia

[BBB04] L. Berggren, J.M. Borwein, P.B.Borwein," Pi, a source book", Springer 2004.
[EPM47] AA. VV., "The Encyclopedia of pure mathematics", John Joseph Griffin and Company, 1847.
[Sca82] L. Scaglianti, Argomenti di Analisi, CEDAM, 1982.
[Vöc10] B. Vöcking, Algorithms unplugged,Springer, 2010.


```
x      = +                                                                 000000000 000000000 000000012
           .556000000 000000000 000000000 000000000 000000000 000000000 000000000 000000000 000000000 000000000
            000000000 000000000 000000000 000000000 000000000 000000000 000000000

       (Q,q)= esci, (C,c)= cancella una cifra, (.)= punto decimale, (-) cambia segno


exp(x) = +                                                                 000000000 000000000 000283792
           .892684644 221301557 138576819 391430918 457398743 925140462 501347605 952180479 410737739 669854204
            225836883 806938540 327914226 826444417 570922634 853854849 425373403

Termini utilizzati =   214

Programma MP_EXPX : FINE ELABORAZIONE
```

Dimostrazione d'uso del programma MP_EXPX.C

```
/*************************************************************************/
/* Programma: MP_EXPX                                                    */
/* Funzione : Input di un numero X in aritmetica a multipla precisione*/
/*************************************************************************/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>       /* tastiera-video PC IBM amb. Windows MS-DOS */
#define LMAX         22L /* ATTENZIONE! IN C   RANGE = (0:LMAX- 1)    */
#define BASE 1000000000L /* base aritmetica in multipla precisione    */
#define NMAX      50000L /* numero max termini di una serie           */
/* Dichiarazione dei tipi di variabile strutturati                     */
typedef struct { long segno;              /* segno (-1, 0,+1)        */
                 long cifra[(LMAX+1)];    /* cifre in base beta      */

        } MultPrec;
/* Dichiarazione di funzione e subroutine                              */
int  LeggiTasto(void);
void Mulz(long,long,MultPrec *p_a,long,long *p_rp);
void Divz(long,long,MultPrec *p_a,long,long *p_rp);
void Suma(long,long,MultPrec *p_s,MultPrec *p_a,long *p_rp);
void Init(long,long,long,MultPrec *p_a,long,long *p_rp);
void DisplayMultPrec(long,long,MultPrec *p_a,char *s);
void Inpx(long beta,long l,long lpi,long lv,
                                  MultPrec *p_x,char *nomevbl);
void Mmpx(long beta,long l,long lpi,MultPrec *p_c
                       ,MultPrec *p_a,MultPrec *p_b,long *p_rp);
/* Programma principale                                                */
int main(void)
{
      long beta = BASE;        /* base aritmetica multipla precisione */
      long l    = LMAX;        /* lunghezza totale vettore mult.prec. */
      long lpi  = 3L;          /* lunghezza parte intera              */
      long lv   = l-2;         /* numero elementi visualizzabili      */
      MultPrec  s,p,x;         /* variabili multipla precisione       */
      MultPrec  *p_s =&s;      /* puntatore var. mult.prec.           */
      MultPrec  *p_p =&p;      /* puntatore var. mult.prec.           */
      MultPrec  *p_x =&x;      /* puntatore var. mult.prec.           */
      char      nomevbl[6]=""; /* nome variabile m.p. (per la stampa) */
      long rp,  *p_rp=&rp;     /* riporto e puntatore riporto         */
      long rd,  *p_rd=&rd;     /* resto   e puntatore resto           */
      long i;                  /* contatore (vbl. di lavoro)          */
      long ovf, *p_ovf=&ovf;   /* indicatore overflow e puntatore     */

/*                                                                    */
      _clrscr();               /* pulisci schermo                     */
      printf("Programma MP_EXPX : INIZIO ELABORAZIONE\n");
/*                                                                    */
      strcpy(nomevbl,"x ");
      Inpx(beta,l,lpi,lv,p_x,nomevbl);

      Init(beta,l,lpi,p_p,1L,p_rp);
      Init(beta,l,lpi,p_s,1L,p_rp);
      for (i = 1;i <= NMAX;i++) {
         Mmpx(beta,l,lpi,p_p,p_p,p_x,p_ovf);
         if (ovf != 0L) {printf("Overflow Mmpx, ovf=%9.9ld",ovf);break;}
         Divz(beta,l,p_p,i,p_rd);
         Suma(beta,l,p_s,p_p,p_rp);
         if (rp != 0L)  {printf("Overflow Suma, rp =%9.9ld",rp);break;}
         if (p.segno == 0L) {break;}
      }
      strcpy(nomevbl,"exp(x)");
      DisplayMultPrec(lv,lpi,p_s,nomevbl);
      printf("\n\nTermini utilizzati = %5d",i);

/*                                                                    */
      printf("\n\nProgramma MP_EXPX : FINE ELABORAZIONE\n\n");
      return 0;
}
/*-----------------------------------------------------------------------*/
/* Funzione : visualizza numero in multipla precisione base 10^9         */
/*-----------------------------------------------------------------------*/
void DisplayMultPrec(long lv,long lpi,MultPrec *p_a,char *nomevbl)
{     long i,j;
      printf("\n%-6s = ",nomevbl);
      j=lpi%10L;
      if ((*p_a).segno >  0L) {printf("+");}
      if ((*p_a).segno == 0L) {printf(" ");}
      if ((*p_a).segno <  0L) {printf("-");}
      if (j != 0L) {
         for ( i = j ; i < 10L ; i++ ) {printf("          ");} /*10bl */
      }
      for ( i = 1L ; i <= lv ; i++ ) {
         if (i == (lpi+1L)) {printf(".");} else {printf(" ");}
         printf("%9.9ld",(*p_a).cifra[i]);
         if ((i % 10L)==j) {printf("\n          ");}      /* 10 blank */
      }
}
/*-----------------------------------------------------------------------*/
/* Funzione: moltiplicazione corta a=a*z in multipla precisione          */
/*-----------------------------------------------------------------------*/
void Mulz(long beta,long l,MultPrec *p_a,long z,long *p_rp)
{     long i,izer,sz,az;
      long long int ipro,w;
      sz=(z > 0L)-(z < 0L);            /* segno di z                   */
      az=sz*z;                         /* valore assoluto di z         */
      izer=0L;
      *p_rp=0L;
      for (i = l;i >= 1L;i--) {
         ipro=(*p_a).cifra[i];         /*workaround bug ottim. PellesC*/
         ipro = (ipro * az) + *p_rp;
         *p_rp=ipro/beta;
         w = beta;                     /*workaround bug ottim. PellesC*/
         (*p_a).cifra[i] = ipro - (*p_rp * w);
         if ((*p_a).cifra[i] > 0L) {izer= 1L;}
      }
      (*p_a).segno=(*p_a).segno * sz * izer;
}
/*-----------------------------------------------------------------------*/
/* Funzione: divisione corta a=a/z in multipla precisione                */
/*-----------------------------------------------------------------------*/
void Divz(long beta,long l,MultPrec *p_a,long z,long *p_rd)
{     long i,izer,sz,az;
      long long int idiv,w;
      sz=(z > 0L)-(z < 0L);            /* segno di z                   */
      az=sz*z;                         /* valore assoluto di z         */
      izer=0L;
      *p_rd=0L;
      for (i = 1L;i <= l;i++) {
         idiv = beta;                  /*workaround bug ottim. PellesC*/
         idiv = (*p_rd * idiv) + (*p_a).cifra[i];
         (*p_a).cifra[i]=idiv / az;
         w = az;
         *p_rd = idiv - ( (*p_a).cifra[i] * w);
         if ((*p_a).cifra[i] > 0L) {izer= 1L;}
      }
      (*p_a).segno=(*p_a).segno * sz * izer;
}
/*-----------------------------------------------------------------------*/
/* Funzione: somma algebrica s=s+a in multipla precisione                */
/*-----------------------------------------------------------------------*/
void Suma(long beta,long l,MultPrec *p_s,MultPrec *p_a,long *p_rp)
{     long i,isom,izer=0L,icnf=0L,az;
      *p_rp=0;
      if ((*p_a).segno != 0L) {
         if ((*p_s).segno == 0L) {
            for (i = 1L;i <= l;i++) {(*p_s).cifra[i]=(*p_a).cifra[i];}
            (*p_s).segno = (*p_a).segno;
         } else {
             az = (*p_s).segno * (*p_a).segno;
             if (az == -1L) {
                for (i = 1L;i <= l;i++) {
                   if ((*p_s).cifra[i] > (*p_a).cifra[i]) {icnf= 1L;}
                   if ((*p_s).cifra[i] < (*p_a).cifra[i]) {icnf=-1L;}
                   if (icnf != 0L) {
                      (*p_s).segno = (*p_s).segno * icnf;
                      break;
                   }
                }
             }
```

```
            for (i = l;i >= 1L;i--) {
               if (az == -1L) {
                  if (icnf >= 0L) {
                     isom = *p_rp + (*p_s).cifra[i] - (*p_a).cifra[i];
                  } else {
                     isom = *p_rp + (*p_a).cifra[i] - (*p_s).cifra[i];
                  }
               } else {
                  isom = *p_rp + (*p_s).cifra[i] + (*p_a).cifra[i];
               }
               *p_rp = (isom >= beta) - (isom < 0L);
               (*p_s).cifra[i] = isom - *p_rp * beta;
               if ((*p_s).cifra[i] > 0L) {izer= 1L;}
            }
            (*p_s).segno=(*p_s).segno * izer;
         }
      }
}
/*---------------------------------------------------------------------*/
/* Funzione: inizializzazione con un numero z                          */
/*---------------------------------------------------------------------*/
void Init(long beta,long l,long lpi,MultPrec *p_a,long z,long *p_rp)
{     long i,irp;
      (*p_a).segno=(z > 0L)-(z < 0L);     /* segno di z               */
      for (i = 1L;i <= l; i++){ (*p_a).cifra[i] = 0L; }
      *p_rp= (*p_a).segno*z;              /* valore assoluto di z     */
      for (i = lpi;i >= 1L;i--) {
         irp=(*p_a).cifra[i]+*p_rp;
         *p_rp=irp / beta;
         (*p_a).cifra[i] = irp - (*p_rp * beta);
      }
}
/*---------------------------------------------------------------------*/
/* Funzione: digitazione di un numero in m.p.                          */
/*---------------------------------------------------------------------*/
void Inpx(long beta,long l,long lpi,long lv,MultPrec *p_x,char *nomevbl)
{     long i;                  /* indice (vbl di lavoro)              */
      long k;                  /* indice (vbl di lavoro)              */
      long m;                  /* indice (vbl di lavoro)              */
      long z=10L;              /* vbl di lavoro                       */
      long f=1L;               /* cifre in base beta                  */
      long flag_dec = 0L;      /* 1=punto decimale attivo, 0=disatt.  */
      long cont_dec = 0L;      /* conta cifre dopo il punto decimale  */
/*                                                                    */
      int kbint = 0;           /* memorizza tasto premuto             */
      long rrp=0L, *p_rrp=&rrp;/* riporto e puntatore riporto         */
      long rrd=0L, *p_rrd=&rrd;/* resto   e puntatore resto           */
      MultPrec  w;             /* variabile multipla precisione       */
      MultPrec  *p_w =&w;      /* puntatore var. mult.prec.           */

      while (z<beta){f++;z=z*10L;}
      Init(beta,l,lpi,p_w,0L,p_rrp);
      while((kbint != 'q') && (kbint != 'Q')){       /*q,Q per uscire */
          _clrscr();
          for (i = l;i >= 1L;i--) {
              if ((*p_w).cifra[i] > 0L) {
                 if (w.segno == 0L) {w.segno = +1;}
              }
          }
          DisplayMultPrec(lv,lpi,p_w,nomevbl);
          printf("\n\n       (Q,q)= esci, (C,c)= cancella una cifra");
          printf(", (.)= punto decimale, (-) cambia segno\n\n");
          kbint = LeggiTasto();
          if (kbint == '.') {flag_dec = 1;}
          if (kbint == '-') {w.segno = -w.segno;}
          if (flag_dec == 0) {
             if ((kbint >= '0') && (kbint <= '9')
                              && (w.cifra[1] < (beta/10L)) ) {
                Mulz(beta,l,p_w,10,p_rrp);
                w.cifra[lpi] = w.cifra[lpi] + kbint - 48;
             }
             if ((kbint == 'c') || (kbint == 'C')) {
                Divz(beta,lpi,p_w,10,p_rrd); /* si parte da lpi */
             }
          } else {
             if ((kbint >= '0') && (kbint <= '9')
                              && (cont_dec < ((lv-lpi)*f)) ) {
                cont_dec++;
                z = (cont_dec-1) / f;
                k = lpi + z + 1;
                m = f - (cont_dec-1) + z*f;
                z=1;
                for (i = 1L;i<m;i++) {z=z*10L;}
                w.cifra[k] = w.cifra[k] + (kbint - 48)*z;
             }
             if ((kbint == 'c') || (kbint == 'C')) {
                z = (cont_dec-1) / f;
                k = lpi + z + 1;
                m = f - (cont_dec-1) + z*f;
                z=10;
                for (i = 1L;i<m;i++) {z=z*10L;}
                w.cifra[k] = w.cifra[k] - (w.cifra[k] % z);
                cont_dec--;
                if (cont_dec == 0){flag_dec=0;}
             }
          }
      }
      (*p_x) = w;
}
/*---------------------------------------------------------------------*/
/* Funzione realizzata :                                               */
/* restituisce all'esterno il codice ASCII di un tasto premuto         */
/*---------------------------------------------------------------------*/
int LeggiTasto(void)
{
      int c = 0;               /* carattere                           */
      while (_kbhit()) _getch(); /* finche' ci sono caratteri         */
                               /* li legge e li ignora                */
      c = _getch();            /* legge un tasto                      */
      return c;
}
/*---------------------------------------------------------------------*/
/* Funzione: moltiplicazione tra numeri in m.p. c = a * b              */
/*---------------------------------------------------------------------*/
void Mmpx(long beta,long l,long lpi,MultPrec *p_c
                          ,MultPrec *p_a,MultPrec *p_b,long *p_ovf)
{
      long cifra[(2*l)+1];     /* buffer cifre in base beta           */
      long i,j,k;              /* indici (vbl di lavoro)              */
      long rr;                 /* vbl riporto                         */
      long izer = 0L;          /* flag = 1 se cifra<>0, 0 altrimenti  */
      long long int ipro;      /* vbl lavoro                          */

      for (i=(l+1L);i<=(2L*l)+1;i++){cifra[i]=0L;}
      *p_ovf=0L;
      for (i = l;i >= 1L;i--) {
         rr=0L;
         if ((*p_b).cifra[i]>0L) {
            for (j = l;j >= 1L;j--) {
               k=i+j;
               ipro=(*p_b).cifra[i];              /*workaround PellesC*/
               ipro=rr+cifra[k]+((*p_a).cifra[j] * ipro);
               rr=ipro/beta;
               cifra[k]=ipro % beta;
            }
         }
         cifra[i]=rr;
      }
      for (i = 1L;i <= lpi;i++) {
         if (cifra[i]>0L) {
            *p_ovf=1L;
            break;
         }
      }
      for (i = 1L;i <= l;i++) {
         (*p_c).cifra[i]=cifra[i+lpi];
         if ((*p_c).cifra[i]>0L) { izer=1L; }
      }
      (*p_c).segno=(*p_a).segno * (*p_b).segno * izer;
}
```

Esplorando l'Amiga
Parte 7 - Inizializzazione della memoria

di Leonardo Giordani

La tabella completa dei vettori di Exec

Nel terzo articolo di questa serie ho mostrato la tabella dei vettori di Exec e il modo in cui `MakeFunctions` crea la jump table quando Exec viene installata nella memoria. In quell'articolo mi sono concentrato sui primi 4 vettori riservati, ma è utile avere la tabella completa mentre si legge il codice di Kickstart. Questa è quindi la tabella dei vettori di Exec 34.2 (28 Oct 1987).

La colonna **Relative offset** contiene l'offset della funzione nella jump table una volta che la libreria è stata installata, il che permette ai programmi Amiga di chiamare funzioni di Exec attraverso la sintassi `offset(a6)` (per esempio `OpenLibrary` può essere chiamata con `jsr -0x228(a6)`). **Vector position** è l'offset del vettore nella ROM di Kickstart 1.3 (il primo valore, `0x0001a7c`, è la posizione della tabella dei vettori stessa), **Relative address** è il valore esadecimale salvato nella tabella prima della rilocazione, **Absolute address** è la posizione della funzione dopo la rilocazione (cioè la somma ciclica tra l'indirizzo della tabella e l'indirizzo relativo), e **Function** è il nome della funzione secondo i file include e la documentazione Amiga.

Relative offset	Vector position	Relative address	Absolute address	Function
-0x00	00001a7c	08a0	0000231c	Open()
-0x06	00001a7e	08a8	00002324	Close()
-0x12	00001a80	08ac	00002328	Expunge()
-0x18	00001a82	08ac	00002328	Reserved for future use
-0x1e	00001a84	ee6a	000008e6	Supervisor()
-0x24	00001a86	f420	00000e9c	ExitIntr()
-0x2a	00001a88	f446	00000ec2	Schedule()
-0x30	00001a8a	04f8	00001f74	Reschedule()
-0x36	00001a8c	f4a0	00000f1c	Switch()
-0x3c	00001a8e	f4ea	00000f66	Dispatch()
-0x42	00001a90	f58e	0000100a	Exception()
-0x48	00001a92	f0b0	00000b2c	InitCode()
-0x4e	00001a94	f188	00000c04	InitStruct()
-0x54	00001a96	faac	00001528	MakeLibrary()
-0x5a	00001a98	fb36	000015b2	MakeFunctions()
-0x60	00001a9a	f080	00000afc	FindResident()
-0x66	00001a9c	f0e8	00000b64	InitResident()
-0x6c	00001a9e	1596	00003012	Alert()
-0x72	00001aa0	08ee	0000236a	Debug()
-0x78	00001aa2	f9ac	00001428	Disable()
-0x7e	00001aa4	f9ba	00001436	Enable()
-0x84	00001aa6	051a	00001f96	Forbid()
-0x8a	00001aa8	0520	00001f9c	Permit()
-0x90	00001aaa	f6e2	0000115e	SetSR()
-0x96	00001aac	f708	00001184	SuperState()
-0x9c	00001aae	f734	000011b0	UserState()
-0xa2	00001ab0	f74e	000011ca	SetIntVector()
-0xa8	00001ab2	f794	00001210	AddIntServer()
-0xae	00001ab4	f7d4	00001250	RemIntServer()
-0xb4	00001ab6	f8e0	0000135c	Cause()
-0xba	00001ab8	fc5c	000016d8	Allocate()
-0xc0	00001ac0	fcc4	00001740	Deallocate()

L'header della memory list

Prima di addentrarci nel codice di `AddMemList` per vedere come lo spazio di memoria viene aggiunto alla memoria libera, diamo un'occhiata allo stato della memory list stessa, visto che dobbiamo familiarizzare con la sua struttura per capire il resto del processo.

Ricorderete dal quinto articolo della serie che la struttura `MemList` structure viene creata `0x142` byte dopo `ExecBase`, e che la sua struttura è la seguente

```
0xe +-------+ 0x150
    |       |
0xd +-------+ 0x14f (LH_pad)
    | 0xa   |
0xc +-------+ 0x14e (LH_TYPE)
    | 0x142 |
0x8 +-------+ 0x14a (LH_TAILPRED)
    | 0x0   |
0x4 +-------+ 0x146 (LH_TAIL)
    | 0x146 |
0x0 +-------+ 0x142 (LH_HEAD)
```

Questo significa che possiamo accedere alla memory list con `lea 0x142(a6),a0`, dato che `0x142` è l'indirizzo relativo, cioè assumendo che `ExecBase` sia

-0xc6	00001abc	fd54	000017d0	AllocMem()
-0xcc	00001abe	fe00	0000187c	AllocAbs()
-0xd2	00001ac0	fdb0	0000182c	FreeMem()
-0xd8	00001ac2	fe90	0000190c	AvailMem()
-0xde	00001ac4	fede	0000195a	AllocEntry()
-0xe4	00001ac6	ff6c	000019e8	FreeEntry()
-0xea	00001ac8	fb6c	000015e8	Insert()
-0xf0	00001aca	fb98	00001614	AddHead()
-0xf6	00001acc	fba8	00001624	AddTail()
-0xfc	00001ace	fbc0	0000163c	Remove()
-0x102	00001ad0	fbce	0000164a	RemHead()
-0x108	00001ad2	fbde	0000165a	RemTail()
-0x10e	00001ad4	fbf4	00001670	Enqueue()
-0x114	00001ad6	fc1a	00001696	FindName()
-0x11a	00001ad8	0208	00001c84	AddTask()
-0x120	00001ada	02b4	00001d30	RemTask()
-0x126	00001adc	0334	00001db0	FindTask()
-0x12c	00001ade	0388	00001e04	SetTaskPri()
-0x132	00001ae0	03e2	00001e5e	SetSignal()
-0x138	00001ae2	03d8	00001e54	SetExcept()
-0x13e	00001ae4	0490	00001f0c	Wait()
-0x144	00001ae6	0408	00001e84	Signal()
-0x14a	00001ae8	0584	00002000	AllocSignal()
-0x150	00001aea	05bc	00002038	FreeSignal()
-0x156	00001aec	054e	00001fca	AllocTrap()
-0x15c	00001aee	0574	00001ff0	FreeTrap()
-0x162	00001af0	00d8	00001b54	AddPort()
-0x168	00001af2	00f0	00001b6c	RemPort()
-0x16e	00001af4	00f4	00001b70	PutMsg()
-0x174	00001af6	016e	00001bea	GetMsg()
-0x17a	00001af8	019c	00001c18	ReplyMsg()
-0x180	00001afa	01b6	00001c32	WaitPort()
-0x186	00001afc	01de	00001c5a	FindPort()
-0x18c	00001afe	f9cc	00001448	AddLibrary()
-0x192	00001b00	f9da	00001456	RemLibrary()
-0x198	00001b02	f9f0	0000146c	OldOpenLibrary()
-0x19e	00001b04	fa26	000014a2	CloseLibrary()
-0x1a4	00001b06	fa3a	000014b6	SetFunction()
-0x1aa	00001b08	fa58	000014d4	SumLibrary()
-0x1b0	00001b0a	ec14	00000690	AddDevice()
-0x1b6	00001b0c	ec22	0000069e	RemDevice()
-0x1bc	00001b0e	ec26	000006a2	OpenDevice()
-0x1c2	00001b10	ec74	000006f0	CloseDevice()
-0x1c8	00001b12	ec9c	00000718	DoIO()
-0x1ce	00001b14	ec8a	00000706	SendIO()
-0x1d4	00001b16	ed0e	0000078a	CheckIO()
-0x1da	00001b18	ecb2	0000072e	WaitIO()
-0x1e0	00001b1a	ed2a	000007a6	AbortIO()
-0x1e6	00001b1c	01e8	00001c64	AddResource()
-0x1ec	00001b1e	01f0	00001c6c	RemResource()
-0x1f2	00001b20	01f4	00001c70	OpenResource()
-0x1f8	00001b22	07b8	00002234	execPrivate7()
-0x1fe	00001b24	07c2	0000223e	execPrivate8()
-0x204	00001b26	07ee	0000226a	execPrivate9()
-0x20a	00001b28	06a8	00002124	RawDoFmt()
-0x210	00001b2a	f700	0000117c	GetCC()
-0x216	00001b2c	fdda	00001856	TypeOfMem()
-0x21c	00001b2e	131c	00002d98	Procure()
-0x222	00001b30	1332	00002dae	Vacate()
-0x228	00001b32	f9f8	00001474	OpenLibrary()
-0x22e	00001b34	1354	00002dd0	InitSemaphore()
-0x234	00001b36	1374	00002df0	ObtainSemaphore()
-0x23a	00001b38	13c4	00002e40	ReleaseSemaphore()
-0x240	00001b3a	1428	00002ea4	AttemptSemaphore()
-0x246	00001b3c	1458	00002ed4	ObtainSemaphoreList()
-0x24c	00001b3e	14ce	00002f4a	ReleaseSemaphoreList()
-0x252	00001b40	14f4	00002f70	FindSemaphore()
-0x258	00001b42	14e4	00002f60	AddSemaphore()
-0x25e	00001b44	14f0	00002f6c	RemSemaphore()
-0x264	00001b46	effc	00000a78	SumKickData()
-0x26a	00001b48	ffaa	00001a26	AddMemList()
-0x270	00001b4a	1504	00002f80	CopyMem()
-0x276	00001b4c	1500	00002f7c	CopyMemQuick()

0. Una volta che l'indirizzo assoluto è in uno dei registri di indirizzo possiamo accedere al primo nodo tramite (aX), con X il numero del registro. Questa è la versione Motorola Assembly dell'operatore di dereferenziazione di un puntatore in C (Address Register Indirect Mode). Con (aX) non usiamo il contenuto di aX, ma il contenuto della cella di memoria il cui indirizzo è contenuto in aX.
Quindi se per esempio a0 è 0x142 nella figura precedente, (a0) è 0x0 (0x142 contiene 0x146, 0x146 contiene 0x0). È quindi molto conveniente pensare ad aX come al puntatore e ad (aX) come al valore.

È interessante notare come la struttura all'indirizzo 0x146 sia molto simile alla struttura LN. Si ricordi che quest'ultima è definita in `include_i/exec/nodes.i` come

```
STRUCTURE    LN,0    ; List Node
    APTR    LN_SUCC ; Pointer to next (successor)
    APTR    LN_PRED ; Pointer to previous (predecessor)
    UBYTE   LN_TYPE
    BYTE    LN_PRI  ; Priority, for sorting
    APTR    LN_NAME ; ID string, null terminated
    LABEL   LN_SIZE ; Note: word aligned
```

e come si può vedere LH_TAIL, LH_TAILPRED, e LH_TYPE possono agire come un vero e proprio nodo di una linked list. Questo è stato fatto di proposito (ovviamente), in quanto la coda della lista deve venir elaborata dal codice che gestisce le linked lists.

Aggiungere memoria libera alla lista di sistema

Alla fine dell'articolo precedente abbiamo lasciato Exec subito dopo che il codice ha preparato i parametri della memoria libera che è disponibile nel sistema. Come abbiamo visto, questa operazione è sempre eseguita per la chip memory, e opzionalmente per la fast memory se l'espansione di memoria è installata.

La funzione `AddMemList` a 0x1a26 viene chiamata con il seguente prototipo

```
size = AddMemList(size, attributes, pri, base, name)
D0                D0    D1          D2   A0    A1
```

con lo scopo di aggiungere al pool di sistema l'ammontare `size` di memoria, iniziando dall'indirizzo `base`. Gli attributi della memoria contenuti in `attributes` verranno salvati nel blocco di memoria stesso e la priorità `pri` verrà usata per trovare il punto di inserimento. Anche il nome `name` viene aggiunto al blocco di memoria per poterlo facilmente identificare.

Possiamo dividere la procedura di aggiunta della memoria in due parti. `AddMemList` crea un nodo che contiene i parametri della regione di memoria, per poi chiamare `Enqueue` che aggiungerà il nodo alla linked list `MemList`. Quando Exec inizializza il sistema la memory list è vuota, ma queste funzioni devono poter essere eseguite in un caso generico. Effettivamente, quando il sistema ha un'espansione di memoria (fast memory) essa viene aggiunta per prima alla lista, ma è immediatamente seguita dalla memoria chip.

Per aiutare il lettore a seguire quello che accade nella funzione, questa è una descrizione dell'area di memoria che vogliamo aggiungere al pool di sistema alla fine di AddMemList, subito prima che Enqueue venga chiamata. L'area contiene un header composto da tre strutture: LN (linked list node), MH (memory header), e MC (memory chunk)

```
SIZE +---------------+
     | Free memory   |
     +---------------+
     | ...           |
     +---------------+
     | Free memory   |
0x2c +---------------+ <-+
     | MC_SIZE       |   |
0x28 +---------------+   |
     | MC_BYTES      |   | MC
0x24 +---------------+   |
     | MC_NEXT       |   |
0x20 +---------------+ <-+
     | MH_FREE       |   |
0x1c +---------------+   |
     | MH_UPPER      |   |
0x18 +---------------+   |
     | MH_LOWER      |   | MH
0x14 +---------------+   |
     | MH_FIRST      |   |
0x10 +---------------+   |
     | MH_ATTRIBUTES |   |
 0xe +---------------+ <-+
     | LN_NAME       |   |
 0xa +---------------+   |
     | LN_PRI        |   |
 0x9 +---------------+   |
     | LN_TYPE       |   | LN
 0x8 +---------------+   |
     | LN_PRED       |   |
 0x4 +---------------+   |
     | LN_SUCC       |   |
 0x0 +---------------+ <-+
```

È interessante notare che, visto che stiamo gestendo la memoria di sistema, non possiamo salvare informazioni su di essa in altro posto che nella memoria stessa. La presenza di strutture di gestione, quindi, riduce l'ammontare della memoria libera. Questo deve essere preso in considerazione quando si progetta un sistema di gestione della memoria, ma non verrà discusso in questo articolo.

Quando creiamo la struttura mostrata in figura dobbiamo assicurarci che la memoria libera sia un multiplo di una long word (8 byte), visto che questo è generalmente un caso desiderabile. Lo spazio di memoria che stiamo gestendo comincia a 0 nella figura, ma potrebbe essere effettivamente ovunque nella memoria di sistema, quindi non è garantito che l'inizio o la fine dello spazio libero siano naturalmente allineati ad una long word.

Il codice di AddMemList esegue l'allineamento in due passi. Prima di tutto allinea l'indirizzo di MC alla long word superiore più vicina (multiplo di 8), poi allinea la dimensione totale alla long word inferiore più vicina. Per esempio se il punto di partenza fosse 0 e la dimensione totale 64 byte lasceremmo i valori così come sono: LN_SUCC a 0x0, MC_NEXT a 0x20, e la dimensione totale del blocco (struttura MC + memoria libera) sarebbe di 32 byte.

```
0x40 +---------------+ <-+
     | Free memory   |   |
     +---------------+   | 32 bytes
     | MC            |   |
0x20 +---------------+ <-+
     | MH            |   |
     +---------------+   | 32 bytes
     | LN            |   |
 0x0 +---------------+ <-+
```

Se il punto di partenza fosse 1 e la dimensione totale 65 byte, invece, il risultato sarebbe il seguente: LN_SUCC a 0x1, MC_NEXT a 0x28 (long word superiore), e la dimensione totale sarebbe 56 byte (65-7 fa 58, arrotondato verso il basso ad un multiplo di 8), che significa un blocco di 24 byte

```
0x42 +---------------+ <-+
     | Ignored       |   | 2 bytes
0x40 +---------------+ <-+
     | Free memory   |   |
     +---------------+   | 24 bytes
     | MC            |   |
0x28 +---------------+ <-+
     | EMPTY         |   | 7 bytes
0x21 +---------------+ <-+
     | MH            |   |
     +---------------+   | 32 bytes
     | LN            |   |
 0x1 +---------------+ <-+
```

Il codice di AddMemList

Secondo la tabella dei vettori di Exec, AddMemList può esssere trovata a 00001a26, e termina a 00001a78. La funzione è immediatamente seguita da una padding word 0000 e dalla tabella dei vettori stessa

```
00001a26: 2149 000a              move.l  a1,0xa(a0)
00001a2a: 43e8 0020              lea     0x20(a0),a1
00001a2e: 117c 000a 0008         move.b  #0xa,0x8(a0)
00001a34: 1142 0009              move.b  d2,0x9(a0)
00001a38: 3141 000e              move.w  d1,0xe(a0)
00001a3c: 2209                   move.l  a1,d1
00001a3e: 5e81                   addq.l  #0x7,d1
00001a40: 0201 00f8              andi.b  #-0x8,d1
00001a44: c389                   exg     d1,a1
00001a46: 9289                   sub.l   a1,d1
00001a48: d081                   add.l   d1,d0
00001a4a: 0200 00f8              andi.b  #-0x8,d0
00001a4e: 0480 0000 0020         subi.l  #0x20,d0
00001a54: 2149 0010              move.l  a1,0x10(a0)
00001a58: 2149 0014              move.l  a1,0x14(a0)
00001a5c: 2209                   move.l  a1,d1
00001a5e: d280                   add.l   d0,d1
00001a60: 2141 0018              move.l  d1,0x18(a0)
00001a64: 2140 001c              move.l  d0,0x1c(a0)
00001a68: 2340 0004              move.l  d0,0x4(a1)
00001a6c: 4291                   clr.l   (a1)
00001a6e: 2248                   movea.l a0,a1
00001a70: 41ee 0142              lea     0x142(a6),a0
00001a74: 6100 fc48              bsr.w   0x16be
00001a78: 4e75                   rts
```

Commentiamo la funzione riga per riga. Mi riferirò ai campi mostrati in figura per nome, citando l'offset per chiarezza.

```
00001a26: 2149 000a                move.l  a1,0xa(a0)
```

Questo codice salva l'indirizzo dell'area di memoria in LN_NAME (0xa(a0)).

```
00001a2a: 43e8 0020                lea     0x20(a0),a1
```

Qui viene caricato l'indirizzo assoluto del blocco di memoria (struttura MC). Lo scopo di questo è di allineare il blocco di memoria ad una long word più avanti nel codice.

```
00001a2e: 117c 000a 0008           move.b  #0xa,0x8(a0)
00001a34: 1142 0009                move.b  d2,0x9(a0)
00001a38: 3141 000e                move.w  d1,0xe(a0)
```

Il valore 0xa viene salvato nel campo LN_TYPE (0x8(a0)). Questo corrisponde a NT_MEMORY (si veda include_i/exec/nodes.i). La priorità della lista viene copiata nel campo LN_PRI (0x9(a0)), e gli attributi di memoria vengono salvati nel campo MH_ATTRIBUTES (0xe(a0)).

```
00001a3c: 2209                     move.l  a1,d1
00001a3e: 5e81                     addq.l  #0x7,d1
00001a40: 0201 00f8                andi.b  #-0x8,d1
00001a4e: 0480 0000 0020           subi.l  #0x20,d0
```

Questo codice allinea l'indirizzo di MC (a1) a una long word, aggiungendo 7 e rimuovendo i 3bit meno significativi. Successivamente la dimensione degli header (0x20) viene rimossa dalla dimensione totale della regione di memoria che è stata passata come argomento.

```
00001a54: 2149 0010                move.l  a1,0x10(a0)
00001a58: 2149 0014                move.l  a1,0x14(a0)
```

La prima locazione di memoria (a1) viene salvata in MH_FIRST (0x10(a0)) e in MH_LOWER (0x14(a0)).

```
00001a5c: 2209                     move.l  a1,d1
00001a5e: d280                     add.l   d0,d1
00001a60: 2141 0018                move.l  d1,0x18(a0)
00001a64: 2140 001c                move.l  d0,0x1c(a0)
```

La dimensione della memoria libera viene poi aggiunta al primo indirizzo libero e salvata in MH_UPPER (0x18(a0)), mentre la dimensione stessa viene salvata in MH_FREE (0x1c(a0)).

```
00001a68: 2340 0004                move.l  d0,0x4(a1)
00001a6c: 4291                     clr.l   (a1)
```

Qui il blocco di memoria viene creato nell'area di memoria libera. La dimensione del blocco viene salvata nel campo MC_BYTES (0x4(a1)), e il campo MC_NEXT viene azzerato.

```
00001a6e: 2248                     movea.l a0,a1
00001a70: 41ee 0142                lea     0x142(a6),a0
00001a74: 6100 fc48                bsr.w   0x16be
00001a78: 4e75                     rts
```

Il resto del codice prepara la chiamata alla versione protetta di Enqueue, copiando l'indirizzo del nodo in a1 , l'indirizzo assoluto di MemList in a0, e saltando alla subroutine. Quando Enqueue termina anche AddMemList termina.

Risorse

* Motorola M68000 Family Programmer's Reference Manual
https://www.nxp.com/docs/en/reference-manual/M68000PRM.pdf

* Amiga System Programmers Guide, Abacus
https://archive.org/details/Amiga_System_Programmers_Guide_1988_Abacus

Esplorando l'Amiga
Parte 8 - Branching e manipolazione delle liste

di Leonardo Giordani

Branching

Visto che l'operazione di branching (ramificazione) è uno dei modi più importanti per controllare il flusso del programma è utile discutere in dettaglio come funzioni nel processore Motorola 68000, ed esplorare le opzioni che abbiamo.

Contrariamente a quando accade in linguaggi di alto livello, in Assembly non abbiamo un modo per rappresentare espressioni logiche. In un linguaggio come C possiamo confrontare due interi con un'espressione come `a > b` che ci dice se il primo numero è maggiore del secondo. In Assembly, tuttavia, non esiste una simile sintassi, quindi dobbiamo appoggiarci alle flag del processore; queste sono dei singoli bit che il processore imposta a seconda del risultato di certe operazioni, siano esse un confronto diretto di due valori o qualsiasi altra operazione che gestisce numeri interi.

Le flag del processore Motorola 68000 sono salvate nei 5 bit meno significativi dello Status Register (SR). Quest'ultimo non è disponibile durante le normali operazioni in user mode, ma quei 5 bit, chiamati Condition Code Register (CCR) sono sempre accessibili. I nomi convenzionali di ciascun bit sono X, N, Z, V, e C, e gli ultimi 4 sono quelli che prendiamo in considerazione per il branching.

* N (Negative) viene impostato al valore del bit più significativo del risultato dell'istruzione, per mostrare che è stato prodotto un risultato negativo.
* Z (Zero) viene impostato se il risultato è zero.
* V (Overflow) viene impostato quando un'operazione aritmetica restituisce un numero che non può essere rappresentato con la dimensione dell'operando.
* C (Carry) viene impostato quando un'addizione o una sottrazione ha bisogno di riportare un bit.

Per dividere l'esecuzione secondo il valore di queste flag si usa la famiglia di istruzioni `Bcc`, dove `cc` è uno mnemonico a due lettere che rappresenta la condizione che viene testata. Per esempio la condizione `MI` sta per MInus e testa se la flag `N` è impostata. Quindi `bmi <address>` salterà all'indirizzo indicato se il risultato dell'operazione precedente è negativo, dato che questa operazione avrà impostato la flag N.

Ci sono varie famiglie di istruzioni che usano il CCR; `Bcc`, `DBcc`, `Scc`, e `TRAPcc`. La tabella seguente elenca tutte le possibili condizioni che possiamo testare sul Motorola 68000 e il loro significato.

La ragione per cui le condizioni `T` e `F` non sono disponibili per le istruzioni `Bcc` è semplice: `bt` significherebbe "salta sempre" e l'istruzione `bra` fa esattamente questo, mentre `bf` significherebbe "non saltare mai", il che è evidentemente inutile.

Vediamo un esempio di utilizzo assieme ad un'istruzione standard di confronto: `cmp`

```
cmp.w   d0,d1
beq.b   0x1522
```

Qui il processore confronta `d0` e `d1`, eseguendo la sottrazione `d1 - d0` e impostando le flag del CCR secondo la natura del risultato. Il manuale del processore (Programmer's Reference Manual, section 4-75, page 179) ci dice che `cmp` agisce sulle flag secondo le seguenti regole

Instruction	Full name	Tested condition	Notes
CC	Carry Clear	C == 0	
CS	Carry Set	C == 1	
EQ	EQual	Z == 1	
F	False	Always false	Not available for Bcc
GE	Greater or Equal	N == V	
GT	Greater Than	N == V and Z == 0	
HI	HIgher than	C == 0 and Z == 0	
LE	Less or Equal	Z == 1 or N != V	
LS	Lower or Same	C == 1 or Z == 1	
LT	Less Than	N != V	
MI	MInus	N == 1	
NE	Not Equal	Z == 0	
PL	PLus	N == 0	
T	True	Always true	Not available for Bcc
VC	V Clear	V == 0	
VS	V Set	V == 1	

* X — Not affected.
* N — Set if the result is negative; cleared otherwise.
* Z — Set if the result is zero; cleared otherwise.
* V — Set if an overflow occurs; cleared otherwise.
* C — Set if a borrow occurs; cleared otherwise.

L'istruzione beq alla seconda riga controlla il valore della flag Z, quindi il salto avviene se d0 e d1 hanno lo stesso valore.

Un esempio un po' più complesso utilizza l'operazione move

```
move.w  0x1c(a1),d0
beq.b   0x151e
```

La prima riga muove il valore all'indirizzo a1 + 0x1c in d0. La documentazione di move (Programmer's Reference Manual, section 4-116, page 220) ci dice che essa cambia solamente il valore delle flag N e Z

* X — Not affected.
* N — Set if the result is negative; cleared otherwise.
* Z — Set if the result is zero; cleared otherwise.
* V — Always cleared.
* C — Always cleared.

Il risultato che viene testato è il valore che viene spostato. Quindi la riga seguente salta solamente se l'indirizzo a1 + 0x1c (e quindi d0) contiene un valore nullo.

Si faccia attenzione che GE, GT, LE, e LT leggono N, quindi devono essere usate solamente con valori interi con segno (signed integer). Interi senza segno, invece, devono venir confrontati usando HI e LS.

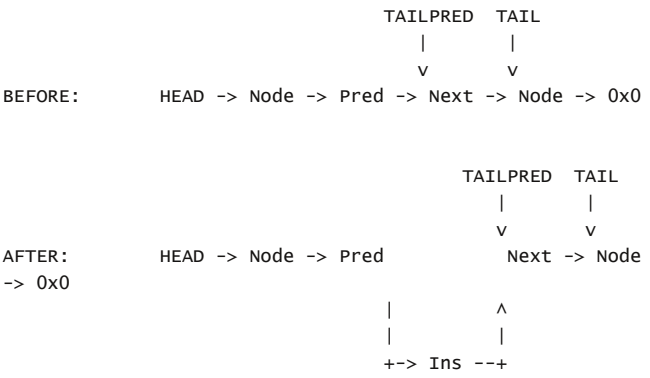
Enqueue

Nell'articolo precedente abbiamo discusso la struttura di un nodo di memoria creato da AddMemList ed ho brevemente citato Enqueue ed una versione protetta di essa. Enqueue è la funzione che Exec usa per aggiungere un nodo ad una linked list, seguendo un semplice schema basato sulle priorità. In effetti AddMemList non utilizza direttamente Enqueue, preferendo invece una versione contornata da due altre funzioni, Forbid e Permit, che sono connesse allo scheduling dei task. Per il momento, ad ogni modo, possiamo dimenticare le funzioni di contorno e saltare direttamente a Enqueue per capire come il sistema operativo dell'Amiga gestisce le linked lists.

Se la lista che stiamo gestendo è semplice (singly linked list), dove ogni nodo è collegato solo al successivo, siamo forzati a reperire il nodo dopo il quale vogliamo inserire il nuovo nodo. Questo risulta obbligatorio, in quanto dobbiamo cambiare il suo puntatore uscente, indirizzandolo al nuovo nodo. In una lista doppia come quelle usate dal sistema operativo dell'Amiga questo non è un problema, visto che da ogni punto della lista possiamo procedere sia verso la testa

che verso la coda.

La figura in basso è una rappresentazione semplice di quello che accade nel codice di Kickstart quando Enqueue viene eseguita. Ins è il nodo che vogliamo inserire, Pred e Next sono i nodi tra i quali vogliamo inserirlo.



Come si nota ci sono svariati puntatori che richiedono di essere modificati: LN_SUCC sia di Pred che di Ins, ma anche LN_PRED di Next e Ins.

Enqueue ha un prototipo molto semplice

```
Enqueue(list, node)
        a0    a1
```

dove a0 e a1 sono puntatori rispettivamente all'header della lista e al nodo che vogliamo inserire. Vale la pena di ricordare il contenuto dell'header della lista quando viene eseguito il primo inserimento, ovvero quando la memoria chip o la memoria di espansione vengono aggiunti al pool di sistema, gestito tramite MemList. Quest'ultimo è una struttura LH e i valori contenuti in essa sono i seguenti

```
    0xe +-------+ 0x150
        |       |
    0xd +-------+ 0x14f (LH_pad)
        | 0xa   |
    0xc +-------+ 0x14e (LH_TYPE)
        | 0x142 |
    0x8 +-------+ 0x14a (LH_TAILPRED)
        | 0x0   |
    0x4 +-------+ 0x146 (LH_TAIL)
        | 0x146 |
    0x0 +-------+ 0x142 (LH_HEAD)
```

Con questa struttura in mente, analizziamo il codice di Enqueue. La funzione è definita a 0xfc1670

```
00001670: 1229 0009               move.b  0x9(a1),d1
00001674: 2010                    move.l  (a0),d0
00001676: 2040                    movea.l d0,a0
00001678: 2010                    move.l  (a0),d0
0000167a: 6706                    beq.b   0x1682
0000167c: b228 0009               cmp.b   0x9(a0),d1
00001680: 6ff4                    ble.b   0x1676
00001682: 2028 0004               move.l  0x4(a0),d0
00001686: 2149 0004               move.l  a1,0x4(a0)
0000168a: 2288                    move.l  a0,(a1)
0000168c: 2340 0004               move.l  d0,0x4(a1)
00001690: 2040                    movea.l d0,a0
00001692: 2089                    move.l  a1,(a0)
00001694: 4e75                    rts
```

E può essere divisa in tre sezioni, secondo i salti interni

```
Init:
00001670: 1229 0009               move.b  0x9(a1),d1
00001674: 2010                    move.l  (a0),d0

FindPos:
00001676: 2040                    movea.l d0,a0
00001678: 2010                    move.l  (a0),d0
0000167a: 6706                    beq.b   InsertNode
0000167c: b228 0009               cmp.b   0x9(a0),d1
00001680: 6ff4                    ble.b   FindPos

InsertNode:
00001682: 2028 0004               move.l  0x4(a0),d0
00001686: 2149 0004               move.l  a1,0x4(a0)
0000168a: 2288                    move.l  a0,(a1)
0000168c: 2340 0004               move.l  d0,0x4(a1)
00001690: 2040                    movea.l d0,a0
00001692: 2089                    move.l  a1,(a0)
00001694: 4e75                    rts
```

Come ho fatto per le funzioni analizzate negli articoli precedenti, analizzerò il codice riga per riga. Partiamo dalla parte etichettata Init

```
Init:
00001670: 1229 0009               move.b  0x9(a1),d1
00001674: 2010                    move.l  (a0),d0
```

Visto che a1 punta al nodo da inserire, 0x9(a1) è il campo LN_PRI del nodo, quindi la sua priorità. La prima riga quindi salva questo valore in d1 perché esso verrà usato per cercare un punto di inserimento, visto che la lista è mantenuta in ordine di priorità.

La seconda riga è il nucleo dell'algoritmo di attraversamento dei nodi. Siccome a0 punta alla struttura LH è anche l'indirizzo del primo campo LH_HEAD. Quindi (a0) dereferenzia quell'indirizzo, per cui sarà l'indirizzo del primo nodo della lista. Dati i valori mostrati precedentemente a0 è 0x142, quindi move a0,d0 salverebbe 0x142 in d0. (a0), invece, è il contenuto di quell'indirizzo in memoria, quindi 0x146. L'operazione move (a0),d0 salva 0x146 (il contenuto di 0x142) in d0.

Lasciando da parte le complicazioni dei modi di indirizzamento, la seconda riga salva l'indirizzo del primo nodo della lista, che è la parte dell'header che inizia con LH_TAIL. L'header quindi agisce come il primo nodo della lista stessa.

```
FindPos:
00001676: 2040                    movea.l d0,a0
00001678: 2010                    move.l  (a0),d0
0000167a: 6706                    beq.b   InsertNode
0000167c: b228 0009               cmp.b   0x9(a0),d1
00001680: 6ff4                    ble.b   FindPos
```

Le prime due riche ripetono lo stesso algoritmo. L'indirizzo del nodo corrente (d0) viene copiato in a0 e l'operazione (a0) mette in d0 l'indirizzo del nodo

successivo. La ragione per cui questo viene fatto in due righe distinte è che la modalità di indirizzamento Address Register Indirect Mode può essere usata solamente con i registri An. A questo punto d0 contiene l'indirizzo del secondo nodo, il successore dell'header.

Se la lista contiene almeno un nodo, d0 a questo punto contiene il suo indirizzo. Ma se la lista è vuota d0 conterrà 0x0, e questa è la condizione testata dall'istruzione beq.b. Se d0 è nullo abbiamo raggiunto la fine della lista, il che significa che non è stato trovato un posto migliore dove inserire il nodo. Pertanto in questo caso saltiamo direttamente al codice di inserimento. Se il valore non è zero il nodo corrente ha un successore, quindi dobbiamo controllare la priorità di quest'ultimo per capire se dobbiamo procedere o se possiamo fermarci. Il codice confronta le priorità del nodo corrente e di Ins, e se quest'ultima è minore della prima torniamo a FindPos e procediamo. Si ricordi che le priorità sono espresse da numeri negativi, pertanto "minore" significa "con piorità più alta".

```
InsertNode:
00001682: 2028 0004                move.l  0x4(a0),d0
00001686: 2149 0004                move.l  a1,0x4(a0)
0000168a: 2288                     move.l  a0,(a1)
0000168c: 2340 0004                move.l  d0,0x4(a1)
00001690: 2040                     movea.l d0,a0
00001692: 2089                     move.l  a1,(a0)
00001694: 4e75                     rts
```

I entrambi i casi, sia quando raggiungiamo la coda o quando la priorità del nodo seguente è minore di quella del nuovo nodo, il processore esegue il codice di inserimento. A questo punto a0 punta a Next e possiamo accedere a Pred attraverso 0x4(a0) (ovvero LN_SUCC di Next).

```
00001682: 2028 0004                move.l  0x4(a0),d0
00001686: 2149 0004                move.l  a1,0x4(a0)
```

prima di tutto salviamo l'indirizzo di Pred in d0, per poi sostituirlo con il valore di a1. Il risultato è che il predecessore di Next diventa Ins.

```
0000168a: 2288                     move.l  a0,(a1)
0000168c: 2340 0004                move.l  d0,0x4(a1)
```

Questo copia a0, l'indirizzo di Next, nel primo campo di Ins, per cui Next diventa il successore di Ins. La seconda riga copia d0 (l'indirizzo di Pred) nel campo LN_PRED di Ins.

```
00001690: 2040                     movea.l d0,a0
00001692: 2089                     move.l  a1,(a0)
00001694: 4e75                     rts
```

Infine l'indirizzo di Ins diventa LN_SUCC di Pred, quindi copiamo d0 in a0 visto che, come ho già spiegato, l'indirizzamento Address Register Indirect Mode può essere usato solo con registri An. Dopo di questo la funzione termina e ritorna al chiamante.

Remove

Dato che abbiamo analizzato il codice di Enqueue vale la pena dare un'occhiata alla funzione opposta, Remove. Esiste una versione protetta anche di questa funzione, ma come fatto per Enqueue eviterò di mostrare per ora il codice di contorno.

Rimuovere un nodo è più semplice che aggiungerlo, visto che tutto quello che dobbiamo fare è di far puntare Pred a Next e viceversa, quindi la funzione è nel complesso molto più corta. Si noti che la funzione accetta l'indirizzo di un nodo in a1, ma che il valore in questo registro viene sovrascritto, quindi l'indirizzo originale deve venir conservato altrove quando si chiama la funzione.

```
0000163c: 2051                     movea.l (a1),a0
0000163e: 2269 0004                movea.l 0x4(a1),a1
00001642: 2288                     move.l  a0,(a1)
00001644: 2149 0004                move.l  a1,0x4(a0)
00001648: 4e75                     rts
```

Le prime due righe salvano l'indirizzo di Next in a0 e quello di Pred in a1 (sovrascrivendo il valore in input). La terza riga pone Next come successore di Pred e la quarta pone Pred come predecessore di Next. La funzione poi termina e ritorna al chiamante.

AddLibrary

Nella sesta parte di questa serie avevamo lasciato Kickstart subito prima che finisse di aggiungere la memoria fisica al pool di sistema. L'ultima istruzione che avevamo menzionato era una chiamata a AddLibrary, e questa è proprio la funzione che voglio esplorare qui.

Il codice della funzione si trova a 0xfc1448, e mi ha richiesto un po' di lavoro prima che potessi effettivamente leggerlo (si veda la sezione "Decompilazione manuale").

```
00001448: 41ee 017a                lea 0x17a(a6),a0
0000144c: 6100 0270                bsr.w 0x16be
00001450: 6100 0082                bsr.w 0x14d4
00001454: 4e75                     rts
```

La prima riga carica l'indirizzo assoluto di un oggetto che si trova 0x17a byte dopo ExecBase, e la tabella della struttura di Exec pubblicata precedentemente ci mostra che questo è l'indirizzo di LibList.

Le righe successive chiamano la versione protetta di Enqueue e SumLibrary, dopo di che la routine ritorna al chiamante. La chiamata ad Enqueue si comporta come spiegato precedentemente, a0 punta alla lista di sistema contenente le librerie e a1 punta all'indirizzo base di Exec, impostato appena prima della chiamata a AddLibrary. Quindi Exec stessa viene aggiunta alle

librerie di sistema attraverso questa routine.

`SumLibrary`, come si deduce dal nome, calcola il checksum di una libreria, o controlla il checksum già esistente.

Decompilazione manuale

Seguendo la chiamata a `AddLibrary` dal corpo principale di Kickstart sono stato sorpreso dal seguente codice

```
00001446: 0000 41ee                 ori.b   #-0x12,d0
0000144a: 017a 6100                 bchg    d0,0x754c(pc)
0000144e: 0270 6100 0082            andi.w  #0x6100,(-
0x7e,a0,d0.w)
00001454: 4e75                      rts
```

che ad una prima occhiata non sembra essere quello di una funzione di base, essendo troppo convoluto. Inoltre, il branch di partenza utilizza l'indirizzo 0x1448, che si suppone essere quindi l'inizio della funzione. Una veloce scorsa ai valore esadecimali rivela che l'indirizzo 0x1446 di Kickstart contiene una padding word 0000 che ha confuso il decompilatore. Per vedere il codice della funzione ho dovuto decompilare manualmente il codice macchina, e siccome il processo è molto interessante ho deciso di mostrarlo in dettaglio.

Quando si vuole decompilare manualmente il codice macchina servono due strumenti: un cheat sheet che riassuma il formato delle istruzioni e il manuale del processore (entrambi disponibili nelle risorse a fine articolo), che aiutino a tracciare il significato dei singoli bit. I valori del codice macchina a cui siamo interessati sono

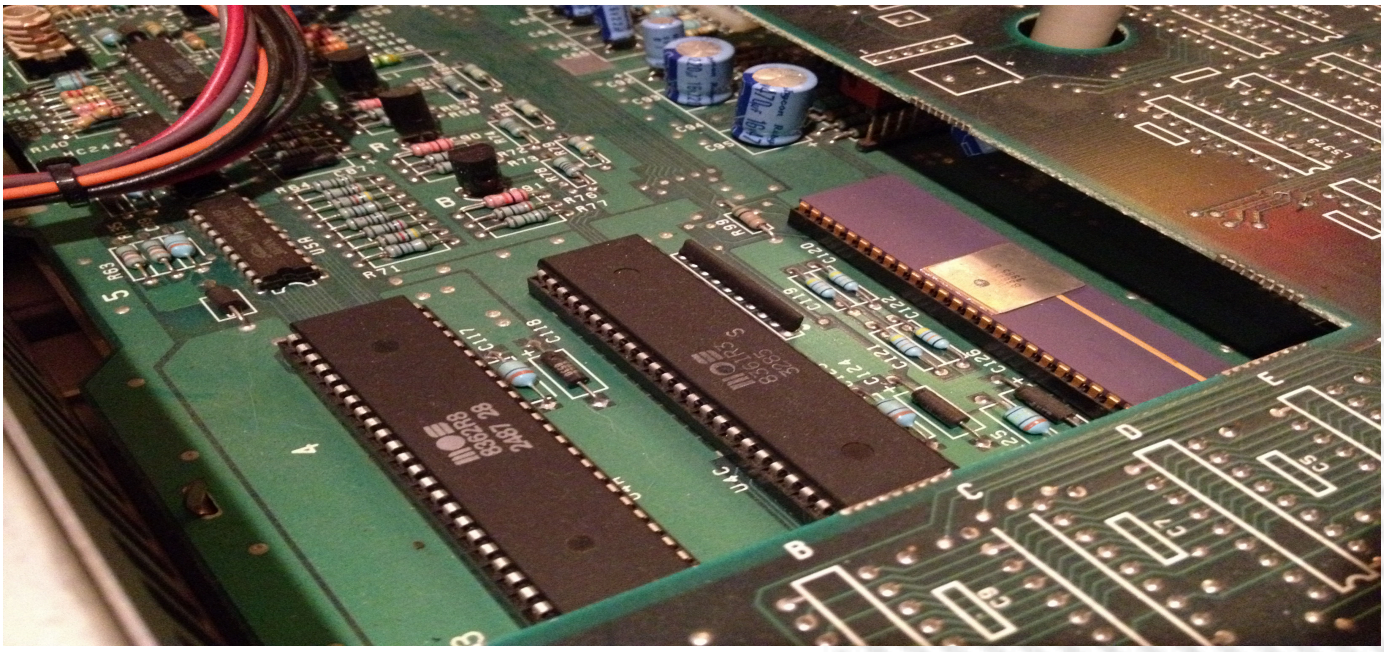
```
00001448: 41ee
0000144a: 017a
0000144c: 6100
0000144e: 0270
00001450: 6100
00001452: 0082
00001454: 4e75
```

dato che 0xfc1456 è presente nella tabella come indirizzo della funzione `RemLibrary`. La rappresentazione binaria di questi valori è la seguente

```
00001448: 0100 0001 1110 1110
0000144a: 0000 0001 0111 1010
0000144c: 0110 0001 0000 0000
0000144e: 0000 0010 0111 0000
00001450: 0110 0001 0000 0000
00001452: 0000 0000 1000 0010
00001454: 0100 1110 0111 0101
```

Ora, i primi 4 bit di ogni istruzione del Motorola 68k sono il codice dell'istruzione stessa. Quando istruzioni multiple condividono gli stessi bit iniziali (ad esempio `andi` e `subi`), tali bit non sono mai utilizzati per gestire indirizzamenti o modi specifici, quindi sono un ottimo punto di partenza.

L'istruzione a 0x1448 inizia con 01000, seguito da uno 0, e questo riduce il campo di possibilità a `lea` o `chk`. Entrambe le istruzioni usano i 3 bit successivi per specificare un registro (`An` per `lea`, `Dn` per `chk`), ma i tre bit successivi cambiano: nel primo caso abbiamo un gruppo fisso 111, mentre nel secondo abbiamo 110. Questo significa che stiamo guardando un'istruzione `lea`, e i 3 bit del registro sono 000, che indica a0 come destinazione. Gli ultimi 6 bit sono il modo di indirizzamento e il registro, e il cheat sheet mostra che 101 110 corrisponde ad Address Register Indirect with Displacement Mode su a6. 101 è etichettato come `(d16, An)`, mentre 110 è il numero 6. La parola seguente è quindi lo spostamento d16 da a6, quindi `41ee 017a` si traduce in `lea 0x17a(a6),a0`.



La seconda istruzione parte a 0x144c con 0110, che è il tratto distintivo di tutti i comandi di branch: `bra`, `bsr`, e tutti quelli basati su condizioni come `bgt`, `blt`, eccetera. I 4 bit successivi sono 0001, pertanto sappiamo che questa è una istruzione `bsr`, salto incondizionato ad una subroutine. In questa istruzione gli 8 bit meno significativi ci dicono che tipo di indirizzamento abbiamo e quindi il tipo dell'operando (Programmer's Manual, section 4-59, page 163). In questo caso tutti i bit sono a 0, che significa un indirizzamento a 16 bit. Si faccia attenzione che, come abbiamo discusso nel primo articolo della serie, il Program Counter contiene l'indirizzo della prima parola di spostamento. In questo caso lo spostamento è 0x270 all'indirizzo 0x144e, quindi l'indirizzo del salto è la somma dei due, quindi 0x16be.

L'ultima istruzione è ancora un `bsr`, e seguendo lo stesso procedimento troviamo che l'indirizzo del salto è la somma tra 0x82 e 0x1452, ovvero 0x14d4. L'istruzione finale è un `rts`, come ci attendevamo, e come già il decompilatore ci aveva mostrato.

La cosa più importante da tenere a mente quando si legge manualmente il codice macchina è la posizione del Program Counter. Come ho già ripetuto due volte, con `lea` e bsr` il PC si muove appena l'istruzione a 16 bit è stata letta, il che significa che lo spostamento dato va usato avendo come base l'indirizzo dello spostamento stesso.

Risorse

* *Motorola M68000 Family Programmer's Reference Manual*
https://www.nxp.com/docs/en/reference-manual/M68000PRM.pdf

* *Motorola 68000 Opcodes Cheat Sheet*
http://goldencrystal.free.fr/M68kOpcodes-v2.3.pdf

RetroMath: ma qui è tutto un "caos"!!!

di Giuseppe Fedele

Nell'ultimo mezzo secolo ha assunto un certo rilievo il concetto di caos deterministico che anche oggi è uno degli argomenti più interessanti della ricerca in matematica e fisica. Nello studio dei sistemi dinamici, con il termine "caos" si intende un movimento irregolare, non periodico, con talune caratteristiche apparentemente casuali, ma generato in realtà da un sistema perfettamente deterministico, anche se necessariamente non lineare.

Il primo articolo sul caos a firma di James A. Yorke e del suo studente di dottorato Tien- Yien Li dal titolo "Period three implies chaos" apparve sul numero di dicembre 1975 dell'American Mathematical Monthly [1].

In fisica è ben noto che, se un evento avviene a una determinata scala (spaziale o temporale), i fenomeni su scala inferiore sono quasi sempre trascurati. In tale contesto, una piccola variazione nello stato iniziale di un sistema, provoca una piccola variazione corrispondente, nel suo stato finale. Un modello che possiede le proprietà appena descritte è definito come modello newtoniano e come tale non può cambiare in modo drammatico per piccole differenze del dato iniziale. Laplace, alla fine del XVIII secolo, concepì il determinismo come la possibilità di conoscere esattamente lo stato futuro di un sistema, a partire dall'indicazione precisa del suo stato attuale, fino a quando Lorenz non introdusse il concetto di caos deterministico. Il termine caos deterministico indica il comportamento di sistemi fisici i quali, pur essendo regolati da leggi ben precise (deterministiche), risente notevolmente di eventuali e piccolissime imprecisioni delle condizioni iniziali, il che implica per lunghi periodi, l'imprevedibilità dello stato del sistema. In un sistema caotico due condizioni iniziali che, alla luce di misurazioni, sembrano identiche, possono portare a due evoluzioni completamente diverse. I sistemi che manifestano dinamiche caotiche sono di solito detti sistemi complessi, come le grandi masse d'aria che solcano il cielo e le dinamiche di popolazione su larga scala [2].

Attrattori nei sistemi non lineari

Consideriamo un sistema del tipo

$$\dot{x} = f(x)$$

e supponiamo che la sua soluzione sia:

$$x(t) = a(t, x_0), t>0$$

Definiamo attrattore per il sistema, un insieme chiuso e limitato $A \in R^n$ che gode di tre proprietà:

1. è invariante: ossia $\alpha(t, x_0) \in A$ per ogni $t \geq 0$; questo significa che la soluzione del sistema partendo da **A** rimane sempre in **A**;

2. è attrattivo: vale a dire che esiste un insieme **U** che contiene **A** per cui se la soluzione parte da essa tende, per tempi sufficientemente lunghi, ad **A**;

3. è minimo (o indecomponibile): non esiste nessun sottoinsieme proprio di **A** che soddisfa le condizioni 1. e 2.

Gli attrattori possono presentarsi in diverse forme, dalle più elementari come semplici punti, alle curve regolari (detti cicli limite), oppure, nel caso dei sistemi caotici, delle strutture ancor più insolite detti **attrattori strani**.

Il caos è "la scienza delle sorprese, dei fenomeni non lineari e imprevedibili". E' una scienza che ci insegna ad aspettarci l'inaspettabile.
In questo articolo spiegheremo cos'è il caos deterministico che, come vedremo, non è una contraddizione di termini.
Vedremo alcuni modelli matematici che generano traiettorie caotiche e proveremo ad implementarli su un Commodore 128!

Uno degli attrattori strani più interessanti fu scoperto da Michel Hénon,astronomo dell'Osservatorio di Nizza. Egli osservò che per determinati valori di energia le intersezioni tra le orbite degli oggetti celesti ed un piano immaginario davano luogo ad una forma geometrica abbastanza regolare, mentre per energie più elevate, tali orbite erano caotiche. Attraverso lo studio del seguente modello matematico ottenne una figura che somigliava ad una specie

$$\begin{cases} x(k+1)=1+y(k)-1.4x(k)^2 \\ y(k+1)=0.3x(k) \end{cases}$$

di banana costituita da più linee. La caratteristica più sorprendente tuttavia era rappresentata dal fatto che quelle linee che apparivano uniche, se ingrandite, erano in realtà costituite da due linee distinte, che a loro volta, ad ingrandimenti maggiori, diventavano quattro, otto e così via. Nonostante ciò, le infinite linee distinte l'una dall'altra giacevano in uno spazio ben confinato (Fig. 1).

Uno dei primi esempi di attrattore

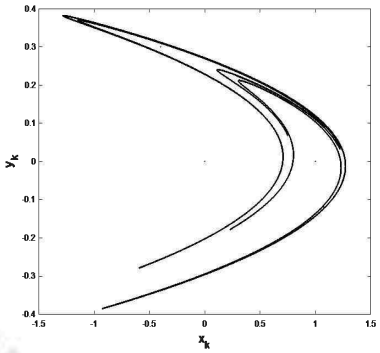


Figura 1 - Attrattore di Hènon

strano fu però quello ideato nel 1963 da Edward N. Lorenz, meteorologo statunitense del Massachusetts Institute of Technlogy. Il modello di Lorenz nasce dalla semplificazione delle equazioni di Navier-Stokes che descrivono il comportamento dinamico di uno strato di fluido che presenta moti convettivi a causa di una differenza di temperatura applicata fra la superficie inferiore e quella superiore. Esso è costituito da un sistema di tre equazioni differenziali

Nella pratica i due parametri **σ** e

$$\begin{cases} \dot{x}=\sigma(y-x) \\ \dot{y}=\rho x-xz-y \\ \dot{z}=xy-\beta z \end{cases}$$

β sono tenuti fissi ai valori rispettivi di **10** e **8/3** , mentre il parametro **ρ** viene lasciato libero ed è l'unico parametro da cui il sistema, di fatto, dipende. L'attrattore di Lorenz si ha per **ρ=28** e somiglia tanto alle ali di una farfalla (Fig. 2).

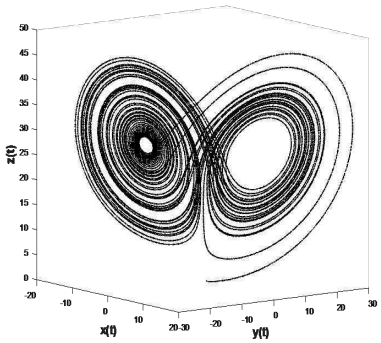


Figura 2 - Attrattore di Lorenz

Con il nome di "Effetto Farfalla", dovuto proprio a Lorenz, si indica il concetto molto più tecnico della sensibilità alle condizioni iniziali, questa frase deriva dall'idea che il battito di ali di una farfalla, ovvero un piccolo cambiamento nelle condizioni iniziali del sistema, può causare un piccolissimo cambiamento nell'atmosfera che può ripercuotersi su larga scala nell'insorgere o meno di un tornado.

La sensibilità alle condizioni iniziali rende impossibile prevedere il comportamento che un sistema caotico avrà dopo un intervallo di tempo anche piuttosto breve. Tale sensibilità indica che delle variazioni molto piccole nelle condizioni iniziali portano il sistema ad evolvere in maniera estremamente diversa. Ciò implica che, anche considerando due sistemi caotici identici, una piccola differenza sulle condizioni iniziali crescerà con un andamento esponenziale nel tempo dando origine a traiettorie completamente diverse.

Il circuito di Chua

Il circuito di Chua (Fig. 3) rappresenta il più semplice strumento in grado di provare l'esistenza del caos sperimentalmente, oltre che prevederne l'esistenza in maniera analitica.

Tali circuiti derivano dagli studi sul caos del prof. Leon O. Chua agli inizi degli anni '80. Si tratta di un circuito elettronico non lineare costituito da due condensatori, un induttore, un resistore e da un resistore non lineare chiamato diodo di Chua.

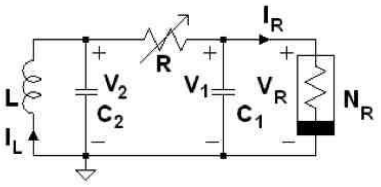


Figura 3 - Circuito di Chua

Il circuito ideato da Chua all'Università di Berkeley in California, può vantare un repertorio straordinariamente vasto di dinamiche non lineari, tanto da essere riconosciuto come paradigma universale per la generazione del caos. Come si può notare dallo schema elettrico, il circuito non presenta segnali d'ingresso ed è per questo che rientra nella categoria dei circuiti autonomi. Il resistore non lineare è un componente attivo, altrimenti il circuito non manifesterebbe nessun comportamento caotico. Una Tipica caratteristica tensione-corrente del resistore non lineare è mostrata in Fig. 4.

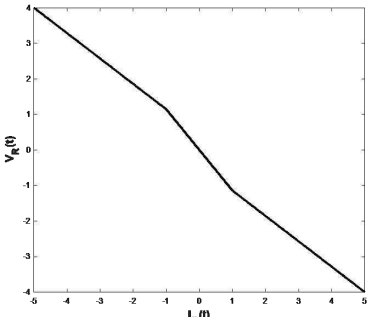


Fig. 4 - Tipica caratteristica v-i del diodo di Chua

Un'altra peculiarità del circuito è data dalla sensibile dipendenza dalle condizioni iniziali. Infatti, possono manifestarsi traiettorie che continuano a muoversi nello spazio di stato presentando oscillazioni non periodiche e non determinabili a priori. In un circuito autonomo, come quello in esame, questo comportamento non è dovuto a fattori forzanti esterni ma è una proprietà intrinseca del sistema stesso.

Senza entrare nei dettagli, è possibile dimostrare che il modello matematico normalizzato del circuito è il seguente:

$$\begin{cases} \dot{x}=\alpha\left(y-x-h(x)\right) \\ \dot{y}=x-y+z \\ \dot{z}=-\beta y \end{cases}$$

dove

$$h(x)=m_1 x+\frac{1}{2}(m_0-m_1)\{|x+1|-|x-1|\}$$

con

$$m_0 < -1 < m_1$$

Per simulare il comportamento caotico del sistema, utilizziamo il modello normalizzato del circuito di Chua, facendo variare il parametro **β** e fissando gli altri parametri come segue:

$$m_0=\frac{-8}{7}, m_1=\frac{-5}{7}, \alpha=15.6$$

Al variare di **β** si osservano gli attrattori mostrati in Fig. 5 e 6.

Una galleria di attrattori strani identificati sul circuito di Chua è descritta in [3]. Alcuni esempi sono mostrati in Fig. 7.

Un altro interessante attrattore fu scoperto da Chua nel 1985 [4]. Si tratta di una struttura macroscopia a doppio scorrimento, cioè due oggetti simili sono arricciati insieme a forma di spirale con un numero infinito di rotazioni. Il codice nel riquadro implementa il double-scroll su Commodre 128. La Fig. 8 mostra l'andamento x-y, z-x e y-x della traiettoria.

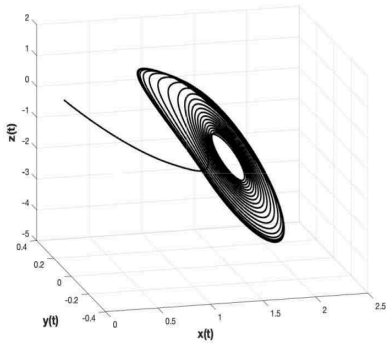


Figura 5 - Attrattore con β=50

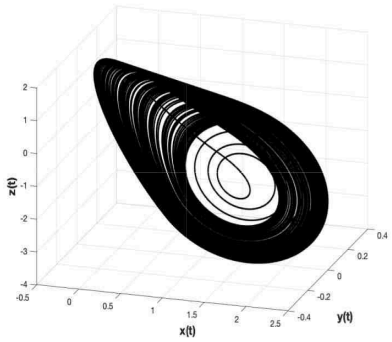


Figura 6 - Attrattore a spirale con β=33

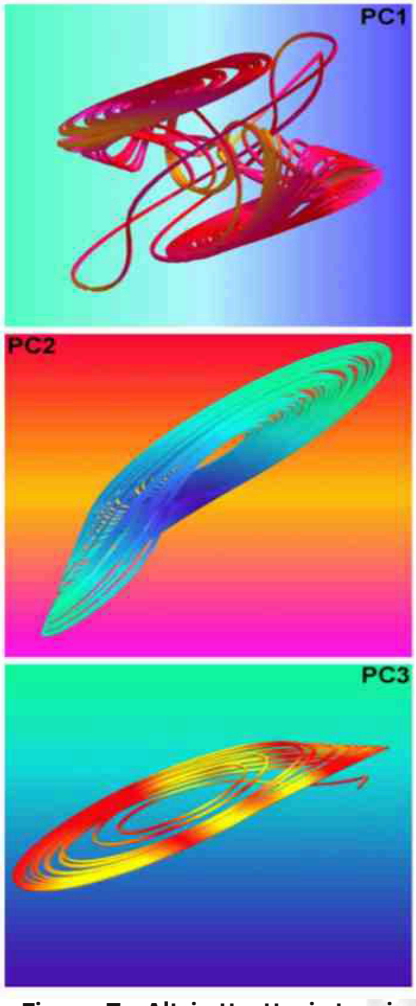


Figura 7 - Altri attrattori strani

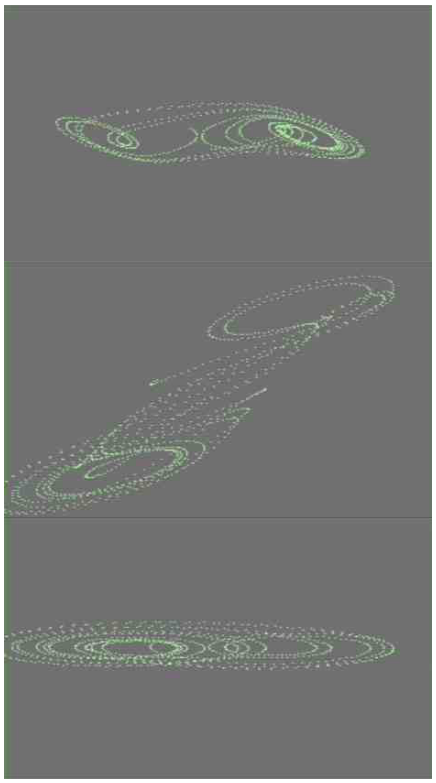


Figura 8 - Traiettorie doppio scroll

```
10 rem Circuito di Chua
20 rem Doppio Scroll
30 rem ----------------------
40 tt=20
50 n=1000
60 ts=tt/(n-1)
70 a=15.6 : b=28.6 : m0=-8/7 : m1=-5/7
90 dim x(n),y(n),z(n)
100 x(1)=0.1 : y(1)=0.2 : z(1)=0.1
110 for k=1 to n-1
120 : hx=m1*x(k)+(m0-m1)*(abs(x(k)+1)-abs(x(k)-1))/2
130 : x(k+1)=x(k)+ts*a*(y(k)-x(k)-hx)
140 : y(k+1)=y(k)+ts*(x(k)-y(k)+z(k))
150 : z(k+1)=z(k)-ts*b*y(k)
160 next k
170 graphic 1,1
180 for k=1 to n
190 : draw 1,50*x(k)+150,50*y(k)+100
200 next k
210 getkey a$
220 graphic 1,1
230 for k=1 to n
240 : draw 1,40*z(k)+150,40*x(k)+100
250 next k
260 getkey a$
270 graphic 1,1
280 for k=1 to n
290 : draw 1,40*z(k)+150,40*y(k)+100
300 next k
310 getkey a$
320 graphic 0
```

Bibliografia

[1] Li, Tien-Yien, and James A. Yorke. Period three implies chaos. The American Mathematical Monthly 82.10 (1975): 985-992.

[2] F. Bertacchini, E. Bilotta, P. Pantano. Il caos è semplice e tutti possono capirlo, GEM Ed., 2009.

[3] E. Bilotta, P. Pantano. A gallery of Chua attractors. World Scientific, Vol. 61, 2008.

[4] T. Matsumoto, L.O. Chua, M. Komuro, The Double Scroll, IEEE Trans. on Circuit and Systems, Vol. CAS-32, no. 8, 1985.

Un Sid engine in basic
Ovvero: come suonargliele (le voci) a colpi di DATA

di Francesco Clementoni a.k.a. Arturo Dente

1. Introduzione

Il Commodore 64 è una macchina meravigliosa, su questo saremo tutti d'accordo, ma il basic v2 è un po' ostico quando si voglia fare qualcosa di più che stampare "ciao" in un ciclo.

In particolare, l'accesso alle funzionalità grafiche e sonore è famoso per il ricorso massivo a pokes vari, nessuna scorciatoia, nessuna pietà!

Soffermandoci sul sonoro, quindi sul SID, l'integrato che svolge a meraviglia tale compito, ci si imbatte per lo più in listati che insegnano come suonare una nota, al più una canzoncina a singola voce, usando tipicamente l'istruzione for...next come metodo per separare una nota da un'altra.

Ma in un contesto videoludico l'ottimo è dato non solo dalla possibilità di suonare contemporaneamente le tre voci, ma dal rendere la stessa esecuzione del pezzo il meno bloccante possibile. Immaginiamo di usare tre voci e di impiegare il sopracitato for...next per ognuna. Nel caso migliore, tre note della stessa durata e con la stessa pausa rispetto alla nota successiva, basterà un solo ciclo di attesa, ma se le note durassero in modo diverso? Dovremmo gestire i tre for... next partendo dal più breve, e passando per i delta di differenza rispetto al primo per le altre note... non c'avete capito niente? Bene, tanto non ci serve, perché qui adotteremo una soluzione completamente diversa.

NB: Tengo a precisare che il programma al quale si perverrà alla fine è una mia espansione di un codice trovato in rete, all'indirizzo http://retro64.altervista.org/blog/commodore-64-sid-music-programming-with-basic-playing-a-simple-three-voices-tune/, al cui autore va la maggior parte dei credits! Il mio contributo consiste nel parametrizzare gli strumenti, normalizzare alcuni aspetti di durata degli stessi, e soprattutto l'implementazione della notazione anglosassone nei DATA piuttosto che l'inserimento a mano delle frequenze.

1.1. Di cosa parliamo e di cosa non parliamo

L'articolo presenta un codice basic (diciamo un "engine") che consente di suonare un pezzo inserendo le note e le relative durate in sequenza nella sezione Data, la quale è suddivisa idealmente in tre blocchi, uno per voce. Attraverso la personalizzazione di alcune (poche) variabili si possono poi scegliere gli strumenti per ogni voce. Non molti, a dire il vero, basso, batteria e piano, ma nulla vieta di estendere il codice con le forme d'onda e i parametri di proprio piacimento.

Si accennerà, inevitabilmente, ai registri del Sid, ma -volendo- il lettore non interessato potrà saltare direttamente alla sezione con il codice, copiarlo e adattarlo.

Quindi, in definitiva, quanto esposto non è un trattato sul Sid, né un Bignami di teoria musicale.

2. Sid for very dummies

Come accennato, non tratteremo estensivamente l'integrato 6581 deputato al suono, ma qualche accenno è necessario per capire come funziona l'engine che andiamo a costruire.

È prassi comune - in praticamente tutti gli esempi ricercabili in rete - di impostare un valore di base per il primo registro coinvolto nelle operazioni sonore, il 54272. Per cui spesso si trova l'inizializzazione della variabile s=54272, e a partire da questa si ricavano gli altri registri coinvolti incrementandola di opportune unità.

Ma nel nostro caso, dovendo gestire tre voci, dal punto di vista mnemonico questo sarebbe un approccio deleterio, perché i registri deputati alle stesse funzioni per ogni voce avrebbero ognuno il proprio offset rispetto ad s.

Ecco perché partiremo subito con tre variabili che svolgono la funzione della succitata "s", una per ogni voce, e le chiameremo L1, L2 ed L3. In immagine la loro rappresentazione e le altre variabili / registri coinvolti. (**Fig.1**)

In questo modo, se vogliamo riferirci ad esempio al registro per il valore dell'alta frequenza della nota, l'offset sarà sempre 1 rispetto al valore base. Quindi L1+1 per la prima voce, L2+1 per

REGISTRI USATI

BASE ->	L1=54272	L2=54279	L3=54286		
	VOCE_1	VOCE_2	VOCE_3	DESCRIZIONE	Variabili (i=1,2,3)
	L1+0	L2+0	L3+0	VALORI_PER_BASSA_FREQUENZA	**Li**
	L1+1	L2+1	L3+1	VALORI_PER_ALTA_FREQUENZA	**Hi=Li+1**
	L1+2	L2+2	L3+2	AMPIEZZA_BASSA_DELL'IMPULSO	
	L1+3	L2+3	L3+3	AMPIEZZA_ALTA_DELL'IMPULSO	
	L1+4	L2+4	L3+4	REGISTRO_DI_CONTROLLO	**Vi=Li+4**
	L1+5	L2+5	L3+5	SETTAGGI_ATTACCO/DECADIMENTO	*Vi+1*
	L1+6	L2+6	L3+6	SETTAGGI_SOSTEGNO/RILASCIO	*Vi+2*

Figura 1 - Registri usati

la seconda, L3+1 per la terza.

E' un modo elegante e mnemonico di riferirci agli indirizzi di nostro interesse.
Ma a cosa servono tali registri? Vediamoli brevemente uno ad uno, facendo riferimento alla tabella:

- **Valori per bassa frequenza e alta frequenza** (Li e Hi nella tabella): mettiamola così: la frequenza d'onda è un numero che può raggiungere valori troppo alti per 8 bit, soprattutto da una certa ottava musicale in poi, per cui è necessario utilizzare due registri ad 8 bit per memorizzare la parte a bassa frequenza e quella ad alta frequenza. Il codice basic che verrà presentato più avanti inizializza un array con otto ottave già pronte con le frequenze fisiche reali di ogni nota (Le frequenze delle note sono state prese da *https://www.edicolac64.com/public/GuidaUtenteC64.pdf*), per poi scomporle nelle due parti tramite le formule fh=int(ff/256):fl=ff-fh*256 , dove ff è la frequenza reale, fh l'alta, fl quella bassa.

- **Ampiezza bassa e alta dell'impulso** (Li+2 e Li+3 nella tabella): Quando si suona una forma d'onda ad impulsi, il segnale può assumere solo due valori: alto o basso. Questi due registri regolano esattamente l'ampiezza di tali estremi.

- **Registro di controllo** (Vi=Li+4 nella tabella): questo registro dice quale forma d'onda si è scelta e stabilisce l'inizio e la fine del ciclo di Attacco – Decadimento – Sostegno – Rilascio (vedi punto successivo). I bit che lo compongono funzionano come dettagliato in **fig. 2**

BIT N°	DESCRIZIONE
0	PORTA
1-3	NON USATO
4	FORMA D'ONDA TRIANGOLO
5	FORMA D'ONDA DENTE DI SEGA
6	FORMA D'ONDA AD IMPULSO
7	FORMA D'ONDA RUMORE

Figura 2 - Significato dei bit nel Registro di controllo

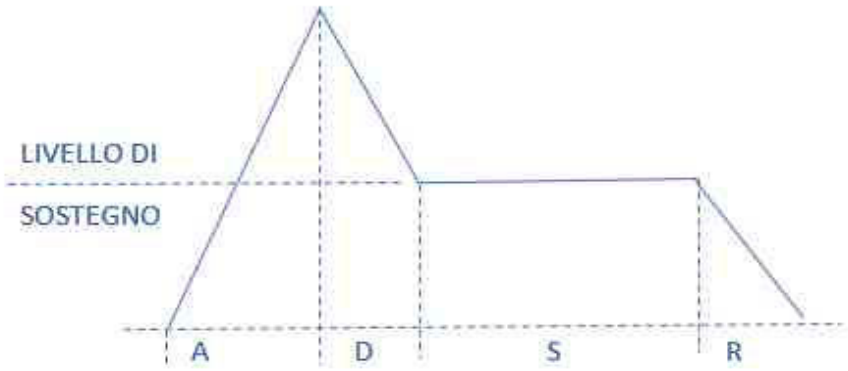


Figura 3 - Attacco, Sostegno, Decadimento, Rilascio (ADSR)

La forma d'onda deve essere una, in altre parole i valori possibili per tale registro non contemplano attivazione di bit di onde diverse, sebbene ciò sia tecnicamente possibile ma con risultati...strani. Una volta scelta la forma d'onda, il "bit di porta" stabilisce l'inizio e la fine del ciclo ADSR, per cui se ad esempio si è scelto il valore 16, la forma d'onda triangolare, lo strumento inizierà la sua nota con il registro di controllo impostato a 16+1=17 e inizierà il rilascio quando lo stesso registro verrà settato a 16.

- **Attacco / decadimento e sostegno/rilascio** (Li+5 e Li+6 – o Vi+1 e Vi+2, che è lo stesso - nella tabella)

Questi parametri rappresentano la nascita, la vita e la morte di una nota, come rappresentato in **fig. 3**

Il volume di un tono musicale cambia dal momento in cui si comincia a sentirlo fino a quando non scompare e non è più udibile. Quando una nota viene prodotta la prima volta, aumenta da un volume a zero fino al suo volume di picco. L'andamento in cui avviene si chiama **attacco**.
In seguito perde il picco e si assesta a un livello medio di volume. L'andamento in cui avviene questo assestamento si chiama **decadimento**.
Una volta raggiunto il volume medio, tutto il tempo in cui rimane la nota si chiama livello di **sostegno**.
Quando infine la nota smette di suonare, perde il livello di sostegno fino al volume a zero. L'andamento in cui cade si chiama **rilascio**.
Mentre per le frequenze delle note si è dovuto ricorrere a due registri per ogni nota – alto e basso – qui accade il contrario, sfruttando mezzo registro per ogni parametro.
Le impostazioni per l'attacco delle tre voci sono contenute nei nibble alti dei registri 5 (Li+5). Per esempio, gli andamenti d'attacco occupano i bit 2^7, 2^6, 2^5 e 2^4, per cui i valori saranno 128, 64, 32 e 16.
Le impostazioni di decadimento sono contenute nei nibble bassi degli stessi registri. Gli andamenti di decadimento usano i bit 2^3, 2^2, 2^1 e 2^0, per cui i valori saranno 8, 4, 2 e 1.
Sommando il valore di attacco e di decadimento scelti abbiamo il numero a 8 bit da inserire nel registro Li+5.
Similmente accade per Sostegno e Rilascio, con l'unica differenza che dei quattro parametri il sostegno è l'unico che riguarda un livello, gli altri sono andamenti. In altre parole, il sostegno è un volume, che inizia a subire il suo abbassamento nella fase di rilascio (ovvero, quando il bit di porta torna

a 0).

Bene, tutto ciò non è strettamente necessario da conoscere per il nostro scopo, sta dietro alle quinte della definizione degli strumenti suonabili, ma spero abbia soddisfatto la curiosità di qualcuno.

3. Let's rock (cit. Duke Nuke'm)

Veniamo alla parte che ci interessa di più: il codice. Di seguito lo copio incollo, per poi commentarlo. Ma prima di proseguire con la lettura invito a farlo eseguire, penso che a tutti voi ricorderà qualcosa…

NB: il codice è in formato CBM Prg Studio, sebbene non vi siano token particolari interpretabili esclusivamente da tale IDE. Tuttavia il copia incolla su Vice darebbe dei syntax error per alcune righe che sforano il numero massimo di caratteri, cosa che non accade se, appunto, si passa per CBM Prg Studio e si lancia il tutto da lì.

```
5 dim fq(96)
10 fq(1)=268:fq(2)=284:fq(3)=301:fq(4)=318:fq(5)=337:fq(6)=358:fq(7)=379:
20fq(8)=401:fq(9)=425:fq(10)=451:fq(11)=477:fq(12)=506:fq(13)=536:fq(14)=568:
30fq(15)=602:fq(16)=637:fq(17)=675:fq(18)=716:fq(19)=758:fq(20)=803:fq(21)=851:
40fq(22)=902:fq(23)=955:fq(24)=1012:fq(25)=1072:fq(26)=1136:fq(27)=1204:
50fq(28)=1275:fq(29)=1351:fq(30)=1432:fq(31)=1517:fq(32)=1607:fq(33)=1703:
60fq(34)=1804:fq(35)=1911:fq(36)=2025:fq(37)=2145:fq(38)=2273:fq(39)=2408:
70fq(40)=2551:fq(41)=2703:fq(42)=2864:fq(43)=3034:fq(44)=3215:fq(45)=3406:
80fq(46)=3608:fq(47)=3823:fq(48)=4050:fq(49)=4291:fq(50)=4547:fq(51)=4817:
90fq(52)=5103:fq(53)=5407:fq(54)=5728:fq(55)=6069:fq(56)=6430:fq(57)=6812:
100fq(58)=7217:fq(59)=7647:fq(60)=8101:fq(61)=8583:fq(62)=9094:fq(63)=9634:
110fq(64)=10207:fq(65)=10814:fq(66)=11457:fq(67)=12139:fq(68)=12860:fq(69)=13625
120fq(70)=14435:fq(71)=15294:fq(72)=16203:fq(73)=17167:fq(74)=18188:fq(75)=19269
130fq(76)=20415:fq(77)=21629:fq(78)=22915:fq(79)=24278:fq(80)=25721:fq(81)=27251
140fq(82)=28871:fq(83)=30588:fq(84)=32407:fq(85)=34334:fq(86)=36376:fq(87)=38539
150fq(88)=40830:fq(89)=43258:fq(90)=45830:fq(91)=48556:fq(92)=51443:fq(93)=54502
160fq(94)=57743:fq(95)=61176:fq(96)=64874
170 rem instruments defs:1=bass,2=drum,3=piano
180 dim ad(3):dim sr(3):dim il(3):dim ih(3):dim wi(3)
190 ad(1)=8:ad(2)=9:ad(3)=9
200 sr(1)=9:sr(2)=10:sr(3)=4*16+9
210 il(1)=10:il(2)=0:il(3)=100
220 ih(1)=30:ih(2)=0:ih(3)=90
225 wi(1)=64:wi(2)=128:wi(3)=64
230 s1=1:s2=2:s3=3:rem assegno le voci agli strumenti.
970 r$="000000"
980 dim f%(1000,2):dim tc(3)
990 tc(0)=4:tc(1)=1:tc(2)=1:rem MINIMUM DURATION
1000 ?"processing data..."
1010 vn=0:cn=0:rem reset counters v n #
1020 read f$,d: dr=d/tc(vn)
1030 if f$="-1" then f%(cn,vn)=-1:vn=vn+1:cn=0:if vn=3 then print"done.":goto1150
1040 if f$="-1" then 1020
1050 ac=-2:if f$="0" then ac=0
1060 gosub1380
1070 ff=f:fh=int(ff/256):fl=ff-fh*256
1080 f%(cn,vn)=fh:f%(cn+1,vn)=fl
1090 if dr =1 then 1140
1100 for t = 1 to dr-1
1110 cn=cn+2
1120 f%(cn,vn)=ac:f%(cn+1,vn)=ac
1130 next t
1140 cn=cn+2: goto 1020
1150 rem let's rock
1160 l1=54272:l2=54279:l3=54286
1170 forj=l1 to 54296:poke j,0:next j
1180 h1=l1+1:h2=l2+1:h3=l3+1
1190 v1=l1+4:v2=l2+4:v3=l3+4
1200 poke 54296,15
1210 poke v1+1,ad(s1):poke v1+2,sr(s1)
1220 pokel1+2,il(s1):pokel1+3,ih(s1)
1230 poke v2+1,ad(s2):poke v2+2,sr(s2)
1235 pokel2+2,il(s2):pokel2+3,ih(s2)
1240 poke v3+1,ad(s3):poke v3+2,sr(s3)
1250 pokel3+2,il(s3):pokel3+3,ih(s3)
1270 cn=0
1280 ti$=r$:t=ti
```

```
1290 x1=f%(cn,0):y1=f%(cn+1,0):x2=f%(cn,1):y2=f%(cn+1,1):x3=f%(cn,2):y3=f%(cn+1,2)
1300 ifx1 =-1 then 1370
1310 ifx1>-2 then pokev1,wi(s1):if x1 > 0 then poke h1,x1:poke l1,y1:poke v1,wi(s1)+1
1320 ifx2>-2 then pokev2,wi(s2):if x2 > 0 then poke h2,x2:poke l2,y2:poke v2,wi(s2)+1
1330 ifx3>-2 then pokev3,wi(s3):if x3 > 0 then poke h3,x3:poke l3,y3:poke v3,wi(s3)+1
1340 cn=cn+2:t=t+4:rem the biggest t, the slower timing
1350 ifti<t then 1350
1360 goto 1280
1370 poke54296,0:end
1380 rem desumo freq. dalla nota.Formato:n(n)o, il secondo n serve per i diesis
1390 f=0
1400 k1$=left$(f$,len(f$)-1):o1$=right$(f$,1)
1401 if f$="-1" then f=-1:return
1408 if f$="0" then f=0:return
1410 if k1$="c" then i1=1:goto1530
1420 if k1$="cs" then i1=2:goto1530
1430 if k1$="d" then i1=3:goto1530
1440 if k1$="ds" then i1=4:goto1530
1450 if k1$="e" then i1=5:goto1530
1460 if k1$="f" then i1=6:goto1530
1470 if k1$="fs" then i1=7:goto1530
1480 if k1$="g" then i1=8:goto1530
1490 if k1$="gs" then i1=9:goto1530
1500 if k1$="a" then i1=10:goto1530
1510 if k1$="as" then i1=11:goto1530
1520 if k1$="b" then i1=12:goto1530
1530 i1=12*val(o1$)+i1:f=fq(i1)
1540 return
1550 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1560 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1570 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1580 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1590 :data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1600 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1610 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1620 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1630 :data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1640 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1650 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1660 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1670 :data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1680 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1690 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1695 data "d2",2,"d2",2,"f2",2,"f2",2,"g2",2,"g2",2,"gs2",2,"a2",2
1700 data"-1",-1
1710 ::data "0",64
1718 :data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1728 data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1738 data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1748 data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1758 :data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1768 data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1778 data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1788 data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1798 :data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1808 data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1816 data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1820 data "c4",4,"b5",4,"c4",2,"c4",2,"b5",2,"b5",2
1825 data"-1",-1
1830 ::data "0",64,"0",64
1840 data"d5",10,"a4",2,"as4",2,"g4",2
1845 data"a4",10,"f4",2,"g4",2,"e4",2
1848 data"f4",10,"d4",2,"e4",2,"cs4",2
1849 data"d4",10,"0",6
1850 data"d5",10,"a4",2,"as4",2,"g4",2
1855 data"a4",10,"f4",2,"g4",2,"e4",2
1858 data"f4",10,"d4",2,"e4",2,"cs4",2
1859 data"d4",10,"0",6
1860 data"-1",-1
```

3.1. Righe 5-60: le frequenze in gioco

Le righe 5-60 non sono altro che l'array di tutte le frequenze per le note delle prime 8 ottave. Sebbene sia comune parlare di "sette note", in realtà sappiamo bene che dalle nostre parti, da diverse centinaia di anni, le note sono 12 perché abbiamo anche 5 semitoni (i tasti neri del pianoforte, per intenderci). Per cui l'array contiene in sequenza, a partire dal do più basso, 12*8=96 valori di frequenza.

3.2. Righe 170-990: inizializzazioni variabili e definizione strumenti in uso

Poiché il programma implementa tre possibili strumenti da assegnare alle voci, in queste righe si preparano gli array dim ad(3):dim sr(3):dim il(3):dim ih(3):dim wi(3) che conterranno rispettivamente: i tre valori scelti di attack-decay, i tre valori scelti di sustain-release, i tre valori scelti di ampiezza bassa e ampiezza alta dell'impulso, i tre valori scelti per la forma d'onda ovvero il registro di controllo.

L'unica parte di codice da personalizzare qui è la riga

```
230 s1=1:s2=2:s3=3:rem assegno le voci agli strumenti.
```

dove decidiamo, per ognuna delle tre voci del sid, quale strumento dovrà suonare, tenendo presente che, per come sono impostati i parametri, avremo 1=bass, 2=drum, 3=piano.

Per cui, nella configurazione attuale, la prima voce suona il basso (s1=1), la seconda la batteria e la terza il piano.

Facendo però uso di sostanze stupefacenti possiamo andare a reperire i parametri per altri strumenti e andare ad aggiungere ulteriori voci (dimensionando opportunamente gli array ad, sr, il, ih, wi visti sopra)

Il blocco di codice qui presentato si chiude con dim f%(1000,2) , un array di interi che conterrà nella prima dimensione le frequenze alte e basse (alta, bassa, alta, bassa…) delle note scelte per il pezzo da suonare, e tutto ciò per ogni voce (per questo è bidimensionale, la seconda dimensione va da 0 a 2, ovvero tre voci). Il suo popolamento sarà il risultato della fase di inizializzazione che andiamo subito qui a descrivere.

3.3. Righe 1000-1140: inizializzazione array delle frequenze del pezzo

Prima di accennare al ruolo di queste righe, c'è da fare una premessa. L'accesso ai DATA non è di tipo random, questo significa che in uno scenario di lettura ed esecuzione contemporanea avremmo dovuto mischiare le note dei tre track inserendole ad ogni colpo (oppure di tre in tre colpi) di read, con conseguente difficoltà di modifica e di lettura dello "spartito" memorizzato. Pertanto, un compromesso ragionevole è quello già accennato: perdiamo un po' di tempo in una fase di inizializzazione di un array bidimensionale (f%) con il vantaggio di scrivere lo spartito in maniera ordinata, suddividendo i DATA in tre blocchi, uno per ogni track.

I DATA presentano lo spartito attraverso le coppie "notaottava",durata_in_sedicesimi, ad esempio "d2",2 significa RE della seconda ottava, durata pari a due (sedicesimi). Perché "sedicesimi"? perchè la battuta dello spartito è qui suddivisa in 16 quanti, "semicrome" per chi mastica un minimo di musica.

Il lettore non uso alla notazione musicale non si spaventi, non lo sono neanche io! Semplicemente, quando la somma delle durate raggiunge 16 vuol dire che, in un ipotetico pentagramma, si sta passando alla battuta successiva, quello spazio di note tra due | | per intenderci.

Uniche eccezioni al formato nota-durata sono la coppia -1,-1 che sta a demarcare la fine del track, e il valore di nota "0" (senza ottava), che significa una pausa (la durata viene presa dal valore di durata, come per le altre note). Nello "spartito" di esempio, si vede che la seconda voce inizia con "0",64 perché deve subentrare dopo 64/16 = 4 battute.

Per i più musicalmente skillati: non è implementata nessuna facilitazione per il tempo. Se volete fare un tre quarti dovete regolarvi da soli con i sedicesimi.

Non ci soffermiamo sul codice più di tanto, perché esula dagli scopi di questo articolo. Del resto questa sezione non contiene alcunché di personalizzabile. Segnaliamo soltanto che il passaggio dalla notazione "amichevole" anglosassone al valore di frequenza avviene tramite la gosub 1380, la quale fa un parse di quanto arrivato tramite read e prende le dovute decisioni. In particolare, se trattasi di una nota, elabora a quale indice dell'array delle frequenze essa debba riferirsi, tramite un semplice calcolo basato posizione della nota nella generica ottava e il numero di ottava indicato (riga 1530).

3.4. Righe 1150-1360: il player

Diciamo subito che nel player c'è un solo – ma importante - parametro personalizzabile, la velocità di esecuzione.
A dire il vero non è proprio un parametro ma una costante (nulla vieta però di assegnargli una variabile), e precisamente ci riferiamo alla riga

```
1340 cn=cn+2:t=t+4:rem the biggest t, the slower timing
```

Quel "t=t+4" indica il timing, se invece che 4 si mette un valore più alto il pezzo andrà più lento. Maggiore è l'incremento di t, minore sarà la velocità di esecuzione.

Detto questo, è utile spendere qualche parola su come funziona il player.

La riga 1160 dovrebbe essere familiare se avete letto il paragrafo 2. l1=54272:l2=54279:l3=54286 sono le tre variabili di base per i vari registri di ognuna delle tre voci. Dopo la riga 1170, che azzera tutti i registri sonori, le due successive seguono la logica vista nella tabella degli offset per accedere ai registri delle tre voci, illustrata sempre al capitolo 2.

Le righe 1210-1250 tirano fuori gli strumenti dalle loro custodie e foderi, dopodichè inizia l'esecuzione vera e propria con la riga 1280.

Tale riga sarà richiamata nel loop come inizio del ciclo, e compie un'operazione importante: azzera la variabile di tempo ti$, assegnandogli la costante r$="000000". Questo è il modo di azzerare il timer, notare che ha effetto anche sulla variabile gemella e numerica "ti". L'istruzione successiva assegna il valore di timer attuale alla variabile t (quella di cui si accennava sopra, dal cui incremento dipende la lentezza di esecuzione).

Successivamente vengono letti, dall'array bidimensionale f% precedentemente inizializzato, i valori di frequenze alti e bassi per ogni voce. Qui si vede chiaramente come questa modalità non sarebbe stata possibile con un accesso non random quale quello dei read-data, dato che ci muoviamo agevolmente tra le tre voci:

```
1290 x1=f%(cn,0):y1=f%(cn+1,0):x2=f%
(cn,1):y2=f%(cn+1,1):x3=f%(cn,2):y3=f%
(cn+1,2)
```

Notare che la riga 1300 vede se x1 è pari a -1. In tal caso passa la palla alla linea di chiusura che azzera il volume e termina il programma. Siccome x1 è letto dal track numero 1, questo significa che non importa quanto lungo sia il track 2 o il track 3, ma la lunghezza del pezzo la definisce il track 1. Del resto, è comunque necessario che la somma dei sedicesimi di ogni track sia la stessa, eventualmente infarcendo il tutto con adeguate pause. In altre parole, i tre blocchi di DATA devono avere le durate che, sommate, sono uguali.

Le righe 1310-1330 suonano le note con frequenze alte-basse xi,yi, ma prima verificano se il valore trovato non sia diverso da una nota. In particolare, il valore -2 (che viene inserito opportunamente in fase di inizializzazione array) sta a indicare che la nota deve continuare a suonare durante questo passaggio di ciclo, per cui le istruzioni if xi>-2 ... qui indicate non fanno assolutamente nulla: il che vuol dire che la nota continua indisturbata a suonare. Notare che le note delle tre voci suonano contemporaneamente (o, meglio, così rapidissimamente vicine da sembrare contemporanee).

La riga 1350, l'unico vero momento di freeze del codice, mostra il motivo per cui abbiamo detto che maggiore è l'incremento di t, meno veloce è l'esecuzione. Infatti, 1350 if ti<t then 1350 attende che il contatore interno – azzerato ad ogni ciclo, come già detto – si incrementi fino al valore di t. Per cui , maggiore è t, maggiore sarà l'attesa.

4. Righe 1550-fine: lo spartito

Veniamo alla parte di maggior personalizzazione: lo "spartito" suddiviso nei tre track, uno per voce, uno strumento per ognuno.

Ricapitolando ed estendendo quanto detto sopra in maniera sparsa:

- le note di ogni track finiscono con la coppia -1,-1
- i DATA contengono coppie di dati del formato notaottava,durata, ad esempio "c7",4.
- le note sono stringhe, precisamente lettere secondo la notazione anglosassone, seguite da un numero compreso tra zero e 8 a indicare l'ottava. "c7" significa "do della settima ottava".
- Le note dei semitoni hanno una "s" dopo il nome della nota. Per cui "cs" sta per "c sharp", ovvero "Do #". Pertanto, "cs4" indica il do diesis della quarta ottava.
- Le pause sono "note" indicate con "0". La durata che le segue indica la durata della pausa.
- La durata esprime il numero di 16mi di durata della nota. In sostanza, misuriamo in semicrome.
- Tuttavia, per non incappare in un overflow, probabilmente per pezzi poco più lunghi di uno medio-corto è da evitare di utilizzare durate minori di 2, per motivi di dimensionamento dell'array.
- Le somme delle durate di ogni singolo track devono coincidere. Se, come nell'esempio qui trattato, la linea del basso (qui è la voce 1) ha una durata complessiva di 256, lo stesso deve valere per le altre voci, eventualmente facendo uso della pausa "0" di opportuna durata.

Lo "spartito" in esempio – che avrete riconosciuto essere un arrangiamento dell'iconico riff di basso + batteria + melodia di Ghost'n Goblins – mostra quanto detto sopra, riferitevi a quel set di data per comprendere quanto detto.

Vedrete che le durate sommate nelle tre voci sono uguali, noterete che il basso (primo track) parte subito, mentre la batteria (secondo track) attende 64 sedicesimi perché inizia con data "0",64 , dando così il tempo al riff di basso di concludere la sua introduzione (lunga sempre 64), e la melodia inizia dopo uno stesso tempo di attesa per l'introduzione della batteria, per cui 64+64=128 sedicesimi di attesa.

Insomma, prima di scrivere i DATA è bene aver chiaro in testa come deve suonare il pezzo. Se sapete

leggere un minimo la musica, consiglio di trovare un adattamento per piano del pezzo che volete riprodurre. È perfetto per i nostri scopi, perché si suppone che una persona abbia due sole mani, per cui potete prendere la linea della mano sinistra e quella della mano destra e assegnare le rispettive note ai primi due track, impostandoli come pianoforte. Con la terza voce libera ci fate invece quello che vi pare.

5. Conclusione

Essendo questo un engine a tre voci scritto in basic, c'è da gioire– intanto – che funzioni.

Teoricamente è possibile "immergere" l'esecuzione del pezzo dentro un codice più generale, un gioco ad esempio, basta ad esempio uscire momentaneamente dal loop che esegue le note tramite gosub a qualcosa per poi riprendere il controllo del tempo trascorso nell'esecuzione delle note, e eventualmente proseguire con la successiva.

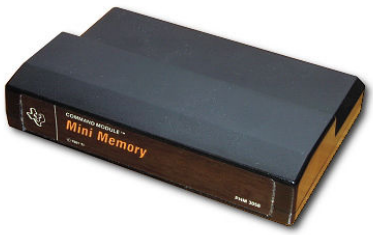
Tuttavia, dati i limiti del basic, tutto ciò resta teoria, a meno che non si voglia stampare un semplice "ciao mondo" a tempo di musica.

Ma per un title screen è perfetto: una get a$ per leggere la pressione di un tasto, oppure una peek sul joystick non sono così bloccanti.

Dark Side of 16 Bit - Ti99 software

di Ermanno Betori

Come scritto nel primo articolo di questa rubrica dedicata al TI99/4A, lo scopo finale è di dare la possibilità al lettore di conoscere discretamente bene il computer in modo da usare al meglio i vari emulatori dedicati ad esso.
Nei numeri scorsi abbiamo presentato quasi tutto l'hardware esistente da ora ci dedicheremo al software.
A differenza di oggi dove un computer viene venduto senza alcun sistema operativo che viene successivamente installato e grazie al quale si possono usare vari programmi applicativi tra cui dei linguaggi di programmazione, quasi tutti i vecchi Home Computer degli anni 70/80, erano venduti anche loro senza alcun sistema operativo ma come sistema di interfacciamento con l'utente usavano di default l'editor di un linguaggio di programmazione che quasi sempre era legato alla architettura hardware del computer e che comprendeva istruzioni in grado di gestire l'hardware della macchina, come i sistemi di memoria di massa o le stampanti.
Il linguaggio di programmazione che venne in pratica adottato da quasi tutti i produttori di computer fu il Basic ed a esclusione di alcune famose aziende come l'Atari, Texas Instruments, Acorn, Sinclair, Amstrad ed altre minori, tutti gli home computer dell'epoca avevano un Basic creato dalla Microsoft e adattato di volta in volta alla macchina destinataria. Basti pensare ad esempio al set di comandi presenti su computer come il Sega SC-3000, il Memotech MTX, la linea MSX ecc. che di fatto è quasi identico tra i vari microcomputer.



La Texas Instruments si fece creare nel 1979 da Bob Wallace che lavorava in Microsoft e da Bob Greenberg una versione Basic da usare essenzialmente sul computer TI99/4. Tale dialetto Basic ANSI non strutturato chiamato TI Basic, è rimasto unico in quanto oltre ad essere molto completo come set di istruzioni, disponeva di una estesa varietà comandi specifici per la gestione del filesystem e delle periferiche di I/O. A differenza di altri BASIC Microsoft, che implementavano le istruzioni LEFT$, MID$, RIGHT$ e INSTR per manipolare le stringhe, il TI BASIC disponeva solo delle istruzioni SEG$ e POS che erano conformi allo standard ANSI. Bob Wallace racconta che quando creò la sua implementazione del Basic, dovette lavorare molto sulla fase di editing dei programmi, per permettere all'utente di cambiare il contenuto del listato agendo direttamente sul numero di linea e non sull'intero testo, cosa oggi data per scontata. Questa implementazione Basic diventò il linguaggio di default del TI99/4, ma soffriva di alcune limitazioni in ambito grafico, poichè mancavano istruzioni per la gestione degli Sprite ed aveva una gestione del testo in parte limitata.
Inoltre la Texas nel creare la sua versione Basic dedicata al computer, voleva che tale linguaggio di programmazione fosse sia molto completo come set di comandi che a prova di errore, pertanto il codice scritto dall'utente veniva controllato e tradotto in un ulteriore macro linguaggio chiamato GPL (Graphic Program Language), eseguito a sua volta da un interprete in linguaggio macchina. Questa procedura però aveva un costo che si concretizzava nel fatto che i programmi basic erano particolarmente lenti, poichè di fatto venivano interpretati due volte durante la loro esecuzione. Per utilizzare il TI99/4 al meglio delle sue capacità era necessario, come per tutti i microcomputer del periodo, usare linguaggi di programmazione più adatti allo scopo come quelli simbolici (Assembly) molto simili al linguaggio macchina pertanto altamente performanti, oppure strutturati come il Pascal, o mix tipo il Fortran, il Forth o il C. Per usare questi linguaggi evoluti il TI99 aveva la necessità di avere una configurazione hardware costosa che all'epoca ben pochi si potevano permettere e questo creava un problema non da poco considerando che già la consolle stessa costava oltre i 1000$! Per ovviare a ciò e venire incontro alle richieste degli utenti, la TI procedette su due fronti: una reingegnerizzazione del computer in un nuovo modello meno costoso completamente compatibile con il predecessore chiamato TI99/4A e con la creazione di due cartucce che funzionando senza la necessità di acquistare ulteriori espansioni, diventarono de facto i primi acquisti obbligatori necessari ad usare il Ti99 in modo avanzato. Una conteneva la versione evoluta dello stesso TI-Basic chiamata TI Extended BASIC ed

una denominata Mini-Memory conteneva una espansione Ram di 4Kbyte, dei comandi aggiuntivi del TI-Basic e un programma assemblatore.

Entrambe le cartucce e il nuovo computer furono venduti nel 1981 dopo quasi due anni della messa in commercio del TI99/4.

Ci fu una attesa da parte degli utenti dell'epoca per questi moduli e specialmente per la versione estesa del basic? Per dare una risposta a questa domanda e per aiutare ad inquadrare i nuovi software nella giusta prospettiva, dovremmo prima di tutto esaminare il clamore mediatico che ci fu negli USA, basti pensare che nell'introduzione del manuale di riferimento veniva citato: "Texas Instruments Extended BASIC basa le sue caratteristiche su quelle che ci si aspetta da un linguaggio di alto livello, con in più comandi aggiuntivi non disponibili in molti altri linguaggi, compresi quelli progettati per l'utilizzo di computer grandi e costosi".

I comandi a cui faceva riferimento l'introduzione, come DISPLAY AT, ACCEPT AT, PRINT USING, IMAGE, ON ERROR, linee di programmazione multiple, operatori booleani AND, OR e NOT, e quelli per le chiamate di subroutine in linguaggio assembly erano attesi nel TI BASIC, poiché erano caratteristiche standard dei vari

```
>10 FOR I=1 TO 128
>20 READ X
>30 F$=F$&CHR$(X)
>40 NEXT I
>50 OPEN #1:F$
>100 DATA 0,0,0,0,0,0,0,0,0,0
 ,0,0,0,0,0,0,0,0,0,0
 ,0,0,0,0,0,0,0,0,0,6,2
 48
>110 DATA 55,215,158,200,0,25
 5,0,0,0,0,159,34,0,0,0,33,
 111,0,0,0,0,0,0
>120 DATA 32,0,33,224,42,93,3
 ,179,216,32,131,106,131,212,
 215,224,131,189,215,224,131,
 188
>130 DATA 215,224,131,191,215
 ,224,131,190,4,224,131,196,2
 16,32,131,187,140,0,4,91,0,0
 ,0,0,0,0,0
>140 DATA 226,208,133,15,71,1
 28,53,103,0,0,131,144,0,0,33
 ,111
>
```

Microsoft BASIC presenti in altri computer dell'epoca, perciò l'Extended BASIC dopo il suo rilascio divento il Basic di riferimento del TI99.

Ma il TI BASIC anche se vetusto non venne abbandonato poichè aveva (ed ha) la sua quota di assi nella manica tipicamente assenti in altri BASIC degli anni 70, come ad esempio comando CALL SAY per utilizzare il sintetizzatore vocale, RESEQUENCE, un set completo EDIT, TRACE e BREAK utilities, più la sua capacità di ridefinire facilmente i caratteri, l'assegnazione del colore agli stessi e i comandi per l'uso del registratore a cassette come memoria di massa o di ben tre disk drive, quando nel 1979 era già proibitivo acquistarne uno e, particolarità all'epoca quasi inesistente, l'utilizzo dei caratteri minuscoli!

Ma non era tutto. Il Ti-Basic aveva altre frecce al suo arco; usando una particolare serie di istruzioni, si poteva forzare il modo di lavoro del chip grafico TMS 9918, permettendo anche in TI-basic la ridefinizione e l'uso degli sprite, seppur con una serie di limitazioni superabili solo ricorrendo all'Extended Basic, il MiniMemory o altri linguaggi superiori.

A conferma di quanto appena detto, nelle ultime due pagine di questo articolo trovate tre listati (uno "di servizio") che mostrano proprio come sia possibile ottenere l'uso degli sprite, tramite il semplice Basic standard. Il programma "Loader" va eseguito prima di provare "Cavaliere che cammina" e "Parsec demo".

L'ultimo listato mostra come anche un altro limite del TI-Basic, ossia caricare in memoria e lanciare un programma in linguaggio macchina da un programma Basic, sia superabile con un'opportuna programmazione.

Come abbiamo già accennato, il Basic di questa macchina non aveva la capacità di eseguire codice macchina nè tantomeno la possibilità di caricarne uno in memoria. Sfruttando in maniera creativa un bug dell'interprete, in particolare del comando OPEN, un intelligente e purtroppo anonimo programmatore è riuscito proprio in questo scopo, utilizzando lo storage delle stringhe REM per caricare il programma in linguaggio macchina. Tutti i listati possono essere eseguiti sia su un computer "reale" che su un emulatore, come il Classic99, il che vi lascia aperte tutte le possibilità nel caso vogliate approfittare delle vacanze estive per divertirvi un po' con la retroprogrammazione. Una raccomandazione: nell'inserire le istruzioni dei listati, dovete seguire la formattazione richiesta dal TI-Basic e non limitarvi ad un copia&incolla, nel caso usiate l'emulatore. Per questioni legate a come il formato PDF formatta il testo, le righe mostrate dai listati includono dei ritorni a capo che debbono essere rimossi dal testo copiato. Siamo spiacenti per questo fastidio ma tant'è, i PDF comportano problemi come questo, ma se avete deciso di provare il codice qui presentato, di sicuro non vi mancano sufficiente dimestichezza con il TI-Basic e tenacia rimediare a questo antipatico "glitch" di formattazione.

Buone ferie dunque, ed arrivederci a settembre.

```
1 REM --------------
2 REM  Loader
3 REM --------------
10 FOR I=1 TO 128
20 READ X
30 F$=F$&CHR$(X)
40 NEXT I
50 OPEN #1:F$
100 DATA
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
,0,0,0,0,0,0,0,6,248
110 DATA
55,215,158,200,0,255,0,0,0,0,0,159,34,0,0,0,33,111,0,0,0,
0,0,0
120 DATA
32,0,33,224,42,93,3,179,216,32,131,186,131,212,215,224,13
1,189,215,224,131,188
130 DATA
215,224,131,191,215,224,131,190,4,224,131,196,216,32,131,
187,140,0,4,91,0,0,0,0,0,0,0,0
140 DATA
226,208,133,15,71,128,53,103,0,0,131,144,0,0,33,111
```

```
10 REM --------------
20 REM Cavaliere che cammina
30 REM --------------
110 CALL CHAR(96,"0F1F1C1C1E1F0F03")
111 CALL CHAR(97,"0303070702020203")
112 CALL CHAR(98,"84C4241424C4841F")
113 CALL CHAR(99,"C43C040480804070")
120 CALL CHAR(100,"0F1F1C1C1E1F0F03")
121 CALL CHAR(101,"0303070703030303")
122 CALL CHAR(102,"80C0201020C18204")
123 CALL CHAR(103,"A8502840000080E0")
130 CALL CHAR(104,"0F1F1C1C1E1F0F03")
131 CALL CHAR(105,"030307070305080E")
132 CALL CHAR(106,"80C0201020C08000")
133 CALL CHAR(107,"0040FF4080008070")
140 CALL CLEAR
150 CALL SCREEN(2)
160 CALL CHAR(33,"FF")
170 CALL HCHAR(1,1,40,256)
180 CALL HCHAR(8,1,33,32)
190 CALL COLOR(1,15,1)
200 CALL COLOR(2,15,7)
220 CALL CHAR(144,"2864C001D0")
230 CALL CHAR(40,"FF202020FF020202")
240 CALL CHAR(40,"FF404040FF040404")
250 CALL CHAR(144,"2864C401D0")
260 CALL CHAR(40,"FF808080FF080808")
270 CALL CHAR(40,"FF010101FF10101")
280 CALL CHAR(144,"2864C801D0")
290 CALL CHAR(40,"FF020202FF20202")
300 CALL CHAR(40,"FF040404FF40404")
310 CALL CHAR(144,"2864C401D0")
320 CALL CHAR(40,"FF080808FF80808")
330 CALL CHAR(40,"FF101010FF010101")
340 GOTO 220
```

```
10 REM --------------
20 REM Lines
30 REM --------------
100 FOR I=1 TO 128
110 READ J
120 A$=A$&CHR$(J)
130 NEXT I
140 FOR I=1 TO 102
150 READ J
160 Z$=Z$&CHR$(J)
170 NEXT I
180 DATA
0,0,2,224,131,160,198,12,4,217,194,58,194,90,162,213,215,
224,131,183,197,198,215,203,208,30,216,30
190 DATA
131,161,2,2,0,100,16,11,0,0,0,0,0,200,0,0,0,0,53,103,0,0,
0,0,192,59,192,155,96,128,2,11,131
200 DATA
0,192,75,2,12,220,94,215,224,131,161,5,194,215,192,16,0,4
,140,6,2,22,253,4,91,152,32,131,115,131
210 DATA
179,22,251,16,209,131,62,192,59,131,126,131,114,131,196,1
31,48,0,6,0,104,140,0,136,0,140,2,0,0,0
220 DATA
0,131,160,77,0,39,227,2,3,131,138,130,72,21,43,2,1,131,16
8,192,8,2,40,0,4,5,194,6,147,192,5,6
230 DATA
147,2,134,154,32,22,242,5,133,4,202,4,199,2,42,255,250,5,
133,192,5,6,1,4,198,7,2,6,147,2,38
240 DATA
255,223,23,228,192,10,10,6,225,198,2,138,255,250,19,239,7
,2,192,4,5,132,2,12,215,113,6,160
250 DATA
131,142,6,1,10,135,5,202,16,230,6,160,131,126,27,2,27,106
260 OPEN #1:A$
23296 REM U!!%Y)0'"/#$S!)#AV,)%I05V`)#AI.A&PT)-I0!
V`)1!.@SQ(,89)/D"9%7`.@"!A-9!.@SQ1O8119$&PX8Q2!!
V`)1!.>""A17
23371 REM `1;AAXY<;BO1Y)%#A!;#`Y-$B$;&%9@9D!"9`1"!9!
4$"=0"1Q8&Q15&R-'%"=9##1#!!AI!9!)(!.-##!#D"K#$@BO3'`9#!
(A!
23446 REM I=12!Q+(!1!2!A5%I=3C"B%$!IA!Q"%#"1;C"K*$%1-#C1%!
%1)&![*$II52!Q++!-!2!A5&II5#!9/O!A)!"A)-VX%'I)//!C!!
23521 REM #!+!?I!7VQ)!.`D)!).U"K#$@BPY('A#!9.UJ'!!%M!2!I!
[A"9%"K#$@BV'(<)#!9/O!A)!"A;AAYI%R!)&!!("B=$(#I/AS'(*
23596 REM &A%(""5#"15&"W)+&1)&"A5)Q+#$RB9(!A)>;[A#A]!
4`=#AA]!+QA)")!#!1B1"Q)()!I.W"K#$@BRK(.1#!9/ZQQI+8/-+!ET`
23671 REM "[-*Q!E#1!!(9Q$8U1),A!$8T!M,]N\!#>@2!GR!!.@-!
CTA!.>,#I$8U;!+V]Q#<%!!EJY7"A<-V]Q'T.@-VW#$>I$!&A1'I)._
23746 REM (.9>2-%%%1/C27%)%-WCBK%(%-I$!!!#!Q!!!!)!"1!#!Y/
K"-1#$!!E--!#$!!'"Q)W!A6#Y1,-QA)A`Q!9]Q)'A]A+'"=-!A?$
23821 REM RM!7&AE(&M!8%Q)%VR!%R?#$Q"!"".9#+O]!&A%%'M%
%&A1'I)._'`A=;.@AA^1%Y)07V_#$V2$-!A)!#A4%"A17`A9#&PQ#!9/G
23896 REM Q##$Q!)A(7M4`=#!%1-#1A!(%!)#9P`YT%)#A9/
O&P,)!)O!!A![?!)#!!9'I)/+!A!!
1!;AAVI#!!"A"K#$7A;AAXY<EBPWV]!'
23971 REM Q.@!"M!#!BA!VU!'!B<^"&M
```

```
1 REM  PARSEC "SPRITE" DEMO
2 CALL CLEAR
3 CALL SCREEN(2)
4 FOR X=1 TO 4
5 CALL COLOR(X,11,1)
6 NEXT X
7 CHARNUM=146
8 REM  UD$=UP/DOWN, LR$=LEFT/RIGHT, SD$=SPRITE DEFINITION
CHARACTER, CC$=COLOR CODE, END$=TURN OFF REST OF HIGHER
NUMBERED SPRITES.
9 UD$="49"
10 LR$="00"
11 SD$="B0"
12 CC$="02"
13 END$="D0"
14 CALL CHAR(33,"0103070F0F0F0F07")
15 CALL CHAR(34,"FFFF00FFFFFFFFFF")
16 CALL CHAR(35,"FEFF03FFFFFFFFFF")
17 CALL CHAR(36,"000080C0C0C0C080")
18 CALL CHAR(37,"0202060A0A0A0A12")
19 CALL CHAR(38,"E8A8ECAAEAAEEAA9")
20 CALL CHAR(39,"00000000070F3E7C")
21 CALL CHAR(40,"0000000080000000")
22 CALL CHAR(41,"1414141414141414")
23 CALL CHAR(42,"A0A0A0A0A0A0A0A0")
24 CALL CHAR(43,"4646495050606040")
25 CALL CHAR(44,"02020686462C1C38")
26 CALL CHAR(45,"0000000000000001")
27 CALL CHAR(46,"000000000040E0F0")
28 CALL CHAR(47,"0301000000000000")
29 CALL CHAR(48,"FFFF141414141414")
30 CALL CHAR(49,"FFFEA0A0A0A0A0A0")
31 CALL CHAR(50,"000000007F605049")
32 CALL CHAR(51,"00000000F8448202")
33 CALL CHAR(52,"0000000000000FFFF")
34 CALL CHAR(53,"12121E030200FFFF")
35 CALL CHAR(54,"E9A9EFB8A800FFFF")
36 CALL CHAR(55,"031F00000000FFFF")
37 CALL CHAR(56,"9CFC3E0F0700FFFF")
38 CALL CHAR(57,"000000008000FFFF")
39 CALL CHAR(58,"141414141414FFFF")
40 CALL CHAR(59,"A0A0A0A0A0A0FFFF")
41 CALL CHAR(60,"4041271F0000FFFF")
42 CALL CHAR(61,"78F8F898247EFFFF")
43 FOR X=1 TO 20
44 PRINT
45 NEXT X
46 PRINT "         ! #$               -.       /01
23            %& '(     )*  +,            "
47 RESTORE
48 FOR FILL=1 TO 33
49 READ Y,X,CP
50 CALL VCHAR(Y,X,CP)
51 NEXT FILL
52 FOR X=65 TO 79 STEP 2
53 CALL CHAR(X,"")
54 NEXT X
55 FOR X=64 TO 78 STEP 2
56 READ X$
57 CALL CHAR(X,X$)
58 NEXT X
59 FOR X=80 TO 83
60 READ X$
61 CALL CHAR(X,X$)
62 NEXT X
63 CALL CHAR(144,"5760A10F5770A50F")
64 CALL CHAR(145,"5780A90F5790AD0F")
65 GOTO 110
66 CALL KEY(0,K,S)
67 IF K=68 THEN 68 ELSE 72
68 TEMP$=LR$
69 Z=4
70 TEMP=2
71 GOTO 87
72 IF K=69 THEN 73 ELSE 77
73 TEMP$=UD$
74 Z=-4
75 TEMP=1
76 GOTO 87
77 IF K=83 THEN 78 ELSE 82
78 TEMP$=LR$
79 Z=-4
80 TEMP=2
81 GOTO 87
82 IF K=88 THEN 83 ELSE 66
83 TEMP$=UD$
84 Z=4
85 TEMP=1
86 GOTO 87
87 DECIMAL=0
88 FOR P=0 TO 1
89 DIGIT=POS("0123456789ABCDEF",SEG$(TEMP$,2-P,1),1)-1
90 DECIMAL=DECIMAL+DIGIT*(16^P)
91 NEXT P
92 DECIMAL=DECIMAL+Z
93 IF DECIMAL<0 THEN 94 ELSE 95
94 DECIMAL=0
95 IF DECIMAL>255 THEN 96 ELSE 97
96 DECIMAL=255
97 GOTO 98
98 DIGIT1=INT(DECIMAL/16)
99 DIGIT2=DECIMAL-(DIGIT1*16)
100 TEMP$=SEG$("0123456789ABCDEF",DIGIT1+1,1)&SEG$
("0123456789ABCDEF",DIGIT2+1,1)
101 ON TEMP GOTO 102,104,106,108
102 UD$=TEMP$
103 GOTO 110
104 LR$=TEMP$
105 GOTO 110
106 SD$=TEMP$
107 GOTO 110
108 CC$=TEMP$
109 GOTO 110
110 SHIP$=UD$&LR$&SD$&CC$&END$
111 CALL CHAR(CHARNUM,SHIP$)
112 GOTO 66
113 DATA
21,13,34,24,1,52,24,2,52,24,3,53,24,4,54,24,5,55,24,6,56,
24,7,57
114 DATA
24,8,52,24,9,52,24,10,52,24,11,52,24,12,52,24,13,58,24,14
,59,24,15,52
115 DATA
24,16,52,24,17,60,24,18,61,24,19,52,24,20,52,24,21,52,24,
22,52,24,23,52
116 DATA
24,24,52,24,25,52,24,26,52,24,27,52,24,28,52,24,29,52,24,
30,52,24,31,52
117 DATA 24,32,52
118 DATA 007C101010101010
119 DATA 0038101010101038
120 DATA 000000007C000000
121 DATA 003844443C040830
122 DATA 003844443C040830
123 DATA 0000040810204000
124 DATA 00081828487C0808
125 DATA 003844447C444444
126 DATA 0000003E1108FF80
127 DATA 47641820FF000000
128 DATA 000000000080F038
129 DATA 9C077C8000000000
```

GYRUSS

Publisher: Konami
Anno: 1983
Piattaforma: Arcade, Commodore 64
Genere: Sparatutto

E' possibile "sparare nel mucchio" e distruggere a distanza qualche navicella nemica...

Nemici in formazione che sparano contro la nostra navicella. Destrezza e buoni riflessi per evitare i colpi e distruggerli tutti, uno per uno!

Benvenuti su Urano :)

Siamo all'inizio degli anni 80, avvento dei primi arcade presenti nei bar e nelle sale giochi, disco dance di nuova generazione, macchine, moto sportive e vestiti firmati, ma anche degli intramontabili soprattutto ambientati nello spazio che tengono incollati al cabinato migliaia di ragazzi e bambini, me compreso.
Il titolone che andremo a ripescare in questo nuovo numero estivo è Gyruss!
Questo titolo fu prodotto da Konami nel 1983. Nacque quindi per i cabinet e fu convertito poi per diverse piattaforme, tra le quali il Commodore 64.
Ci troviamo nello spazio aperto, con decine di astronavi aliene che cercano di bombardarci e... facile a dirsi, un po' meno a farsi, dovremo evitare le bombe facendo un giro a 360 gradi.
Eh sì, una nota caratteristica che contraddistingue questo gioco è proprio il movimento a 360 gradi che compie l'astronave per muoversi ed evitare i nemici.
Una volta sconfitte tutte le astronavi si proseguirà per il livello successivo ed ogni due o tre livelli (warp) ci dirigeremo verso uno dei pianeti del Sistema Solare, partendo dai più lontani (Nettuno, Urano etc) fino ad arrivare alla Terra che sarà il pianeta conclusivo.
Il gioco non sarà provvisto di un finale vero e proprio, bensì si gioca a ciclo continuo; una volta arrivati sulla Terra riprenderemo il nostro viaggio verso la Terra, che potrebbe condurci al Game Over se ci dovesse andar male.
Gyruss non presenta enormi difficoltà.
Con un minimo di riflessi pronti e di studio del nemico, ed una volta presa la mano riuscirete a visitare tutto il Sistema Solare ed a ripulirlo dalle astronavi nemiche che lo infestano.
La musica di questo gioco, molto coinvolgente, non passava inosservata, anzi, esattamente il contrario!

Come vi dicevo qualche numero fa le musiche hanno sempre giocato un ruolo molto importante nel

mondo dei videogiochi in quanto vero e proprio intrattenimento a prescindere dal gioco in sè.
Forse non ci crederete e magari dopo mi odierete, ma è stato uno dei primi giochi che sono entrati nella mia vita e fino a qualche giorno fa non sapevo nemmeno il suo nome!
L' ho trovato per fortuna o per caso gironzolando in rete ed appena l'ho visto morivo dalla voglia di rigiocarlo e di recensirlo!
Per il resto, che dire?
E' un classico che molti di noi avranno visto in qualche bar e/o in sala giochi in quegli anni e che ognuno di noi dovrebbe giocare, almeno una volta nella propria vita.
Anche questa volta, sotto questo caldo afoso di Luglio con davanti il ventilatore, ho scelto con attenzione il titolo da trattare (spero di aver scelto bene), litigando un pò con l'editor installato sul mio computer, combattendo con errori ortografici, impaginazione, scelta e relativo inserimento delle immagini più salienti del gioco e tanto altro ancora.
Ma ormai siamo abituati e tutto questo fa parte del gioco (in tutti i sensi).
Spero che possiate godervi il nuovo numero della nostra rivista in santa pace sotto l'ombrellone e chissà, magari, grazie ad essa, potreste ottenere qualche nuova conquista, perchè no? :)
Buone vacanze a tutti, amici lettori!

di Daniele Brahimi

GIUDIZIO FINALE

» Giocabilità 80%

Comandi di facile utilizzo e divertimento degno di uno sparatutto che si rispetti

» Longevità 89%

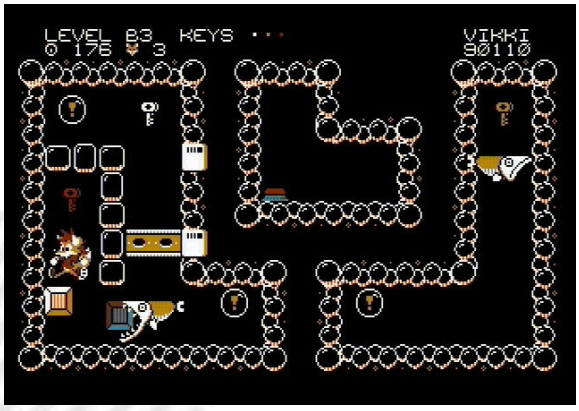
Una volta iniziato e presa la mano, la vedo dura, molto dura, staccarsi dal gioco

RIKKI & VIKKI

Publisher: Penguinet
Anno: 2018
Piattaforma: Atari 7800 / Windows
Genere: Platform

Il genere dei platform non è magari quello simbolo del retrogaming ma resta uno dei primi a fare capolino sulle console domestiche (dopo il suo esordio in sala giochi). L'epoca 8bit fu dominata dai giochi di piattaforme, che si trattasse di quelli a schermata fissa come Donkey Kong o quelli a scorrimento come Super Mario Bros,motivo per cui anche i nuovi homebrew per quelle piattaforme hanno ben ragione di guardare ad un genere che non è mai tramontato, fatto dimostrato anche da recenti titoli come Cuphead o Hollow Knight. Rikki & Vikki, come ormai avrete palesemente intuito, appartiene alla categoria appena annunciata, nello specifico al sottogenere "a schermata fissa".

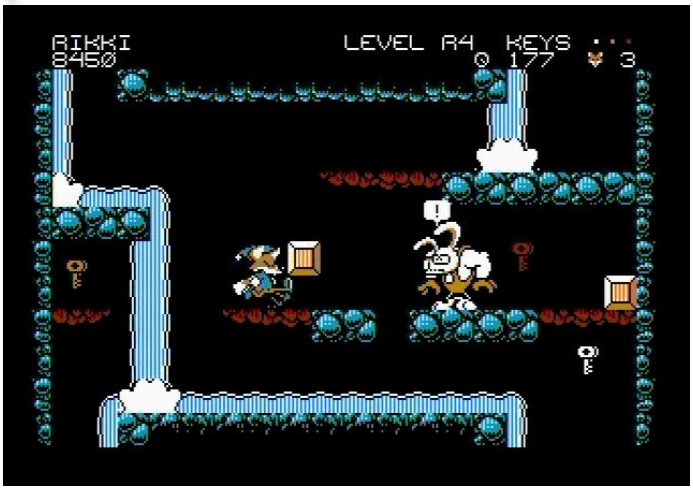
I protagonisti che danno il nome al titolo sono rispettivamente moglie e marito, entrambe volpi antropomorfe (come tali da me molto apprezzati anche nell'estetica) a cui il malvagio di turno ha rapito i figli.
Di fronte ad un tale affronto l'unica possibilità è seguire il marrano nelle Misery Land, una serie di livelli popolati da nemici vari ed enigmi da risolvere per poter passare allo stage successivo. Come riuscirci? Recuperando tutte le chiavi sparse all'interno della schermata ed aguzzando l'ingegno. Nei fatti la parte action è l'elemento più "hardcore" dell'esperienza ma non il principale. Una volta che avrete capito come affrontare determinati enigmi infatti si inizierà a compiere le mosse richieste e, da un certo punto del gioco in poi, queste si tradurranno in movimenti molto precisi e rapidi.
Difficilissimo? No, ma la sfida non è certo per debosciati alle prime armi, credetemi!
La parte che stupisce subito di



Copertina

Grafica Fuori scala

Il chip grafico è stato "pimpato" a dovere da un aggiunta esterna alla card ma il risultato è incredibile per le capacità del Pro System Atari. Vi sfido a trovare risultati migliori di quello ottenuto su Rikki & Vikki.



Rikki & Vikki è l'incredibile realizzazione tecnica. La grafica è dettagliata e pulita, gli effetti sonori e le musiche sono eccellenti (si nota molto bene il chip aggiuntivo) e la longevità è alle stelle: 100 livelli in totale.
Non dovrete affrontarli però tutti di fila poiché sono divisi a seconda di come deciderete di intraprendere l'impresa. Se sceglierete di giocare in single player potrete scegliere tra uno dei due coniugi da interpretare, ma i livelli non differiranno in base alla scelta, mentre se deciderete di giocare con un amico/a i livelli saranno tutti differenti e studiati per essere affrontati in coppia.
Quel che infatti possiamo fare in gioco si traduce nel muoverci, saltare e raccogliere/scagliare oggetti, ma in multiplayer potremo anche fare da trampolino per il nostro socio videoludico, così da permettergli di raggiungere altezze altrimenti precluse in single player.
Questa opzione rende l'esperienza ancora più varia ed appagante ma posso dirvi, da quello che l'ha giocato al 98% in single player, che già da soli si tratta di un gioco enorme e curatissimo.
Rikki & Vikki mi ha stupito per tanti motivi ed in conclusione penso sia l'esempio più calzante di quanto si possa fare bene oggi con le console di ieri. Il gioco è uscito in due versioni: quella fisica per atari 7800 e quella digitale per Windows, disponibile sulla piattaforma Steam.
A voi curiosi la scelta su come approcciarvici, io personalmente ho giocato a lungo sulla versione digitale e sto attendendo quella fisica dall'america. Sì, hanno avuto i miei soldi due volte, capite quanto sia serio nel dire che li meritano!

di Starfox Mulder

GIUDIZIO FINALE

» Giocabilità 95%

Imediato, spassoso e pronto al multiplayer. A meno che non siate nemici giurati dei platform questo gioco è un classico annunciato

» Longevità 90%

La difficoltà si presenta alta quasi da subito ma da un certo punto in poi si parla davvero di hardcore gaming. Per molti la cosa potrebbe risultare frustrante ma se saprete accettare la sfida, arrivare alla fine dei tantissimi livelli sarà un crescendo di divertimento.

NUCLEO 447

Anno: 2019
Autore: Leonardo Vettori
Piattaforma: C64
Genere: Sparatutto

In questo numero e nel prossimo, RetroMagazine ospiterà tra le sue pagine il diario di sviluppo di Nucleo 447, un gioco S.E.U.C.K. realizzato da Leonardo Vettori.
L'opera però non si potrebbe considerare conclusa se non fosse accompagnata da una recensione del gioco. Eccomi quindi, nella nuova veste di genitore, tra un cambio di pannolino ed un altro a scrivere la recensione di Nucleo 447.

Conosco Leonardo personalmente e so con quale passione ha concepito, disegnato e realizzato il suo gioco, quindi potrebbe non essere facile per me essere imparziale. Farò comunque del mio meglio per essere super partes, evidenziando i pregi del gioco e, su precisa richiesta di Leonardo, di elencarne anche i difetti.

Partiamo dal concept del gioco, Nucleo 447 è uno sparatutto a scorrimento verticale ambientato nello spazio profondo. Il nostro eroe, alla guida di un'astronave armata di raggi laser, deve districarsi tra una serie di mondi alieni infestati da nemici e passaggi tortuosi prima di arrivare alla fine della sua avventura e scoprire così il segreto che si nasconde dietro la mente aliena che controlla l'universo.

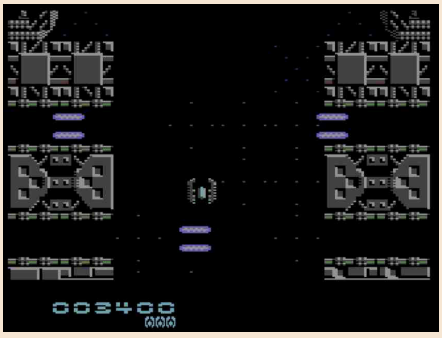
Come nella migliore tradizione dei giochi S.E.U.C.K., anche Nucleo 447 si compone di un lungo scrolling suddiviso in 4 livelli.
Ognuno di questi 4 livelli è però caratterizzato da una ben precisa ambientazione. Abbiamo quindi un livello ambientato in un pianeta ghiacciato, uno in un mondo ricoperto da verde vegetazione ed uno completamente roccioso.
Alt, ferma... ma non avevi detto che i livelli sono 4? Qual è l'ambientazione del quarto livello?
Beh, per quanto riguarda la natura

Il mondo di ghiaccio

Ecco il primo livello in tutto suo glaciale splendore!

Un passaggio impegnativo

Sempre piu' difficile



dell'ultimo livello dovrete scoprirlo da soli, altrimenti scopro troppo le carte e vi priverei del piacere dell'esplorazione.

La grafica e l'ambientazione sono tra le note positive del gioco. I vari livelli sono disegnati con passione e con dovizia di particolari. I tortuosi passaggi obbligati, presenti in ogni ambientazione, aiutano a mantenere alto il livello di attenzione per non rischiare di rimanere bloccati in un vicolo cieco o prematuramente perire nello scontro con una roccia od uno sperone ghiacciato.

Uno dei limiti del SEUCK è la quantità limitata di memoria a disposizione. Questo spesso si traduce in giochi graficamente monotoni o con pochi nemici a corredo. Oppure in alcuni casi i giochi risultano piuttosto semplici perchè non è possibile inserire una complessa AI per gestire i movimenti dei nemici.
Non è il caso di Nucleo 447, dove, come abbiamo appena appurato, il dettaglio grafico è assolutamente di prim'ordine, inoltre Leonardo ha abilmente giocato d'astuzia nella distribuzione dei nemici.
In questo caso le astronavi e le postazioni fisse sono state efficacemente disseminate all'interno di ogni livello, in modo tale da sfruttare le peculiarità dei passaggi obbligati per aumentare la difficoltà del gioco.

Come nella migliore tradizione dei giochi spaziali e Nucleo 447 non fa eccezione, al termine di ogni livello, dovrete affrontare e distruggere un nemico ben più potente delle semplici astronavi che cercheranno di sbarrarvi la strada durante il vostro percorso.
E qui ci troviamo di fronte al primo dei difetti di Nucleo 447: i boss di fine livello.
Intendiamoci, i boss di fine livello sono disegnati molto bene e sono sufficientemente cattivi da incutere rispetto, ma allo stesso tempo sono piuttosto semplici da distruggere. Onestamente come guardiani del passaggio al livello superiore mi sarei aspettato di trovare qualcosa di un po' più ostico.

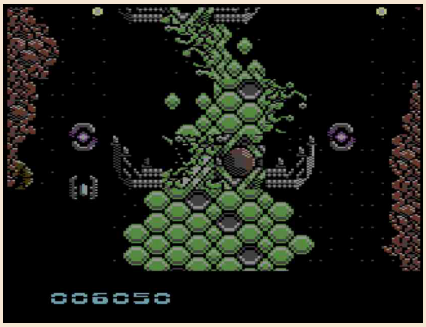
Appurato quindi che il comparto grafico di Nucleo 447 è di tutto rispetto non ci resta che parlare del sonoro, della giocabilità e della longevità di questo prodotto.

Sul sonoro c'è poco da dire. La mancanza della colonna sonora è un altro dei limiti dei giochi S.E.U.C.K. e Nucleo 447 non fa eccezione. Gli effetti speciali invece sono nella norma.

La giocabilità è sicuramente un punto a favore di Nucleo 447. Il gioco si presenta bene e si lascia giocare facilmente, invogliando il giocatore a cercare di migliorare se stesso fino al completamento dello stesso. La difficoltà, boss di fine livello a parte, è ben calibrata e, ad ogni partita, avanzerete quel tanto che basta per non rimanere vittime della frustrazione ed abbandonare il gioco.
Purtroppo però terminerete il gioco piuttosto velocemente e, complice il fatto che la logica che guida i nemici è sempre la solita, raramente lo rigiocherete.
La longevità quindi è un punto a sfavore di Nucleo 447.

Come avrete modo di leggere nel diario di Leonardo, il S.E.U.C.K. è stato spremuto al massimo per cercare di creare il miglior gioco possibile con questo strumento, cercando nel contempo di superarne i limiti.
Il giudizio finale di Nucleo 447 quindi non può che essere positivo. A mio modesto avviso Leonardo ha realizzato un gioco S.E.U.C.K. che, con tutte le limitazioni del caso, potrebbe tranquillamente competere con produzioni commerciali, magari con i famosi budget che tanto andavano negli anni '80.

di Francesco Fiorentini



Nonostante il gioco sia stato realizzzato con il S.E.U.C.K., Leonardo ha realizzato un prodotto che potrebbe essere tranquillamente commercializzato come budget. Ottimo lavoro!

GIUDIZIO FINALE

» Giocabilità 85%

Come scritto nella recensione, Nucleo 447 e' un gran bel gioco che invoglia il giocatore a proseguire l'avventura fino al completamento di tutti e 4 i mondi.

» Longevità 60%

Di contraltare la longevita' risente molto delle limitazioni imposte dal S.E.U.C.K.
Una volta terminato infatti e' difficile che tornerete a giocarlo almeno nell'immediato.
Magari passato qualche tempo potrete riprenderlo in mano per deliziarvi della bella grafica e dei piu' che decorosi fondali.

Un gran bel gioco

Zork I in italiano, a cura di Whovian e Ragfox di Old Games Italia

di Giorgio Balestrieri

CAPITOLO PRIMO

Il re bellicoso

Nel 659 GIS*, il Regno di Quendor era relativamente piccolo: era formato da sette province e mezza sulla costa occidentale del Grande Mare, una zona agricola la cui produzione principale era cordame e tende antizanzare. Era il 31° anno di regno di Zilbo III, discendente di una dinastia sorta più di sei secoli prima con Entharion il Saggio, il primo Re di Quendor. Tuttavia, tale dinastia si estinse con l'avvento di Duncanthrax, che salì al trono nell'ultimo giorno del 659.

Poco si sa delle sorti di Zilbo dopo il 659. Alcune testimonianze dell'epoca affermano che rimase ucciso nel corso di una rivolta all'interno del palazzo, o che morì per i troppi festeggiamenti in onore dell'Anno Nuovo. Ci sono tuttavia dei documenti che attestano che fu confinato in una villa dove inventò il gioco di carte noto come Doppio Fanucci.

Parimenti, gli storici hanno idee discordanti sulla vita di Duncanthrax prima del 659. Una petizione firmata dalle guardie del palazzo nel 657, in cui si chiedeva un aumento delle forniture di tende antizanzare, reca una firma decifrabile come "Duncanthrax". Alcuni storici insistono che Duncanthrax fosse un generale della Guardia Reale. Una leggenda afferma che fosse un demone che aveva assunto sembianze umane. Altri racconti popolari lo vogliono come un ex venditore di corde.

Quali che fossero le sue origini, Duncanthrax si guadagnò molto presto la fama di monarca crudele, aggressivo e assetato di sangue; fu per questo soprannominato "il re bellicoso". Egli formò un enorme esercito con il quale iniziò la conquista dei regni limitrofi. In soli tre anni, Duncanthrax regnava su un impero che virtualmente abbracciava ogni territorio compreso tra il Grande Mare e il Deserto del Kovalli.

*Aggiungere la dicitura "GIS" dopo l'anno diventò pratica comune solo dalla seconda metà dell'VIII secolo.

Un'antica villa nei sobborghi di Mithicus, simile a quella dove Zilbo pare abbia vissuto dopo la deposizione.

Nel numero 8 di RetroMagazine abbiamo già parlato di Zork e della Infocom, le cui genesi sono talmente legate da ricadere nel classico dilemma "è nato prima l'uovo o la gallina", e forse ai lettori appassionati di avventure testuali sarà già noto il lavoro di traduzione in italiano di questo gioco leggendario. In questo articolo abbiamo raccolto la testimonianza degli autori di questa notevole impresa, Whovian e Ragfox, entrambi collaboratori di Old Games Italia (OGI per gli amici), con cui RetroMagazine ha da tempo instaurato una felice collaborazione, in virtù dello spirito che accomuna i due progetti. Prima di passare la parola ai protagonisti, vogliamo darvi alcuni elementi per comprendere appieno l'enormità dello sforzo sostenuto dai due, così da apprezzarne al meglio il racconto dalla loro viva voce (ok, dal vivo scritto, non stiamo a sottolizzare).

Per la produzione dei suoi giochi, la Infocom creò un intero sistema specializzato nella creazione e fruizione di giochi di avventura testuali, inventando una macchina virtuale (detta interprete) capace di far girare programmi creati utilizzando un linguaggio di programmazione ad hoc (ZIL) e compilati nel formato in cui il processore dell'interprete avrebbe potuto eseguirli. Il sistema era inoltre fortemente dipendente dalla lingua inglese ed i testi codificati secondo un formato non standard che garantiva tra l'altro una leggera compressione delle dimensioni. Tutto ciò rende l'opera di traduzione particolarmente complessa e richiede notevoli sforzi ed incredibile determinazione per essere compiuta. Per nostra fortuna, Whovian e Ragfox dispongono di entrambe le qualità (e sospettiamo anche una buona dose di incoscienza) in quantità sufficiente da fargli accettare la sfida e portare a termine con successo questo eccezionale progetto, a beneficio di tutta la comunità degli avventurieri italiani. Ulteriori informazioni sulla Infocom, sul loro sistema e sugli strumenti moderni di sviluppo delle IF possono essere reperite nel numero 8 di RetroMagazine nell'articolo "Le avventure testuali: da Infocom a Inform".

In aggiunta alla difficoltà tecnica intrinseca nella traduzione di un software in generale, per i giochi Infocom c'è un livello di complicazione in più: i feelies. La Infocom distribuiva insieme ad ogni titolo, oltre al supporto fisico del software (il floppy disk in pratica), anche del materiale aggiuntivo che aveva lo scopo di ampliare l'esperienza di gioco e di permettere all'avventuriero di calarsi meglio nel mondo virtuale creato dal programma. I feelies potevano consistere in fumetti o libricini con storie introduttive e novelle di contorno alla storia narrata, fogli di indizi chiamati "invisiclues", che rivelavano suggerimenti se il giocatore avesse colorato certe aree con un pennarello speciale incluso nella confezione o anche accessori assolutamente folli, come il

"cartoncino degli odori" di "Leather Goddesses of Phobos". Siamo ragionevolmente sicuri che di fronte a questo aspetto, chiunque si fosse trovato nei panni di Whovian e Ragfox avrebbe deciso all'istante di lasciar perdere i feelies e tradurre solo il gioco che, ricordiamo, è già di per sè una rogna assurda. Ma per il nostro determinatissimo duo questo non rappresentava altro che l'ennesima sfida con cui confrontarsi perciò, non pago dell'immane lavoro con i testi del gioco, Ragfox decise di tradurre anche TUTTI i feelies a corredo di Zork, mettendoli poi a disposizione per il libero download insieme al gioco.
Ma come è stato possibile fare tutto ciò? Continuate a leggere e lo scoprirete, direttamente dalle parole di chi ha compiuto questa monumentale opera.

GB: Iniziamo dalle presentazioni, una breve descrizione per conoscere gli uomini dietro al nick: come e quando avete conosciuto le avventure testuali, su che sistema le avete giocate e come vi siete incontrati?

W: Ciao, sono Bruno e quando ho lavorato a Zork I avevo circa quindici anni. Conobbi Zork per via di alcuni riferimenti in tv (the big bang theory, Chuck). In Chuck sentii per la prima volta "Stai per essere divorato da un grue". Da lì partì l'idea di giocare e anche io uccisi il mio troll su emulatore Frotz. Solo che io mi stufo in fretta dei giochi, preferisco smontarli: cosa sta accadendo nel codice ora? Come si realizza questa funzionalità? Allora scoprii Inform 6 di Graham Nelson, linguaggio apposito per creare avventure testuali (1993!) e mi studiai il manuale (avevo già un po' di esperienza in programmazione). Per esercitarmi ricostruivo parti di Zork e quando ho avuto in mano qualcosa, mi sono iscritto a OGI e l'ho presentato (in realtà prima passai da Adventure's Planet, ma l'idea era ancora in gestazione e non riscosse molta attenzione). Su OGI invece, dopo un po' di miei passi falsi iniziali, Ragfox e The Ancient One si misero di impegno rispettivamente a tradurre e a supportare il progetto, e anche Gweneland e The Ruler mostrarono molta simpatia per la cosa. Senza il sostegno loro e di tutta la comunità di OGI non avrei mai avuto la determinazione di arrivare in fondo.

R: Le mie prime esperienze con le avventure testuali risalgono a un paio di pomeriggi afosi di qualche decennio fa, quando sul commodore 64 di un amico giocavamo in gruppo a un'avventura testuale italiana, una di quelle che si compravano in edicola (che tempi!). Si trattava di un gioco a tema horror. Potrebbe essere Il Mistero di Villa Parson del mitico Bonaventura di Bello, ma purtroppo non ricordo con precisione. Da traduttore amatoriale di videogiochi e appassionato di avventure testuali, non mi sono lasciato sfuggire l'occasione di partecipare al progetto di traduzione di Zork I, proposto da Whovian nel forum di OldGamesItalia, il portale che ospita diverso materiale relativo all'interactive fiction, tra cui l'archivio di IF Italia e le traduzioni ufficiali degli articoli apparsi su The Digital Antiquarian.

GB: Infocom, un leader indiscusso nel campo delle avventure testuali: quali giochi avete provato e quali avete amato di più (oltre a Zork ovviamente)? Cosa vi ha colpito maggiormente dei giochi di questa azienda?

W: Di avventure testuali sono alquanto digiuno (ops!), però ad esempio sul forum di OGI ho guardato lo svolgersi di "Violet" e da lì ho giocato i successivi "Vespers" e "Wishbringer". Ricordo che in "Vespers" alcune mie scelte portarono l'avventura a un finale tragico e in "Wishbringer" fui più accorto e diffidente: la Malvagia non riuscì a ingannarmi! Mi sono piaciuti i mondi che erano in grado di creare e gli enigmi da risolvere (il maze di zork è assolutamente non banale per gli standard moderni) e poi "Wishbringer" ha dei simpatici easter egg legati a Zork (puoi vedere un piccolo Grue alla luce di un frigorifero!).

R: Infocom ha avuto il merito di esser stato il primo editore a dare più peso e "dignità" alla narrazione all'interno delle avventure testuali. Fu da questa politica che nacquero la collaborazione con Douglas Adams, l'autore di "Guida Galattica per Autostoppisti" e la pubblicazione di un titolo come "A Mind Forever Voyaging", in cui gli enigmi erano una semplice cornice a un gioco basato sulla pura esplorazione di un mondo virtuale. Ma non fraintendetemi: se c'era da tirar fuori un puzzle particolarmente intrigante, il team guidato da Marc Blank e Dave Lebling non era certo secondo a nessuno. Inoltre, anche dal punto di vista tecnico la casa statunitense fu in grado di proporre al grande pubblico diverse innovazioni. Prendiamo per esempio "Border Zone", un'avventura che non si può definire la migliore della Infocom, ma che ho particolarmente apprezzato per la sua originalità dal punto di visto delle meccaniche, in quanto il gioco si svolge in tempo reale e non a turni come normalmente avviene negli altri titoli.

GB: Parliamo ora di Zork, quanto ci avete giocato e cosa avete amato della saga? Cosa invece proprio non vi è piaciuto?

W: Zork I ha un sacco di enigmi interessanti (uno su tutti la diga controllata dal bullone) e c'è molta sinergia fra le varie parti del gioco, ad esempio c'è una stanza vicino alla diga in cui non si può restare per via del rimbombo dell'acqua (a meno che la diga non sia vuota). E nonostante molti enigmi siano di natura "tecnica", del tipo un certo strumento, magari usato in modo creativo (cough dentifricio cough) sblocca la situazione, non mancano enigmi più fantasiosi come quelli con le candele. Avete mai provato a uccidervi in Zork, magari avendo prima posato lo sguardo su un misterioso libro nero? Il lato "negativo" è che, per quanto me la cavi in inglese, specialmente in seconda liceo l'inglese di Zork significava un uso smodato del dizionario. In un certo senso ho tradotto Zork anche per me stesso! Non temete, tutto il lavoro di traduzione vera e propria è stato dato in mano a Ragfox, che ha certamente molta (mooolta) più esperienza di me sul tema.

R: Zork entra a far parte delle mie esperienze con l'interactive fiction relativamente tardi, e ci arriva dopo titoli molto più "giovani" di lui, come le avventure della Lucas, le avventure ibride della Legend e persino dopo "The Pawn" della Magnetic Scrolls, uno dei suoi "cloni" più famosi. Il mio primo approccio con la serie fu in realtà con "Return to Zork", avventura grafica non particolarmente brillante che però ebbe il merito di farmi conoscere il Grande Impero Sotterraneo. Non posso in ogni caso definirmi un fan sfegatato della saga, tant'è vero che ho giocato seriamente a Zork I solo quando ho iniziato a partecipare alla traduzione. Se confrontato con titoli successivi (tra cui quelli scritti da Steve Meretzky o da Brian Moriarty, quest'ultimo autore di "Loom" per la Lucas), Zork appare ancora troppo legato all'idea del dungeon da esplorare derivata dal suo mitico progenitore "ADVENT" (o "Colossal Cave" che dir si voglia) e al piacere quasi sadico di veder fallire il giocatore di fronte agli enigmi, le cui soluzioni sono nascoste principalmente nella mente dei programmatori piuttosto che negli indizi presenti nelle varie location. Insomma, non certo il punto più alto dell'Interactive Fiction, ma senza dubbio uno dei suoi capostipiti; grezzo, ma degno di rispetto e ammirazione.

GB: Nel post (o più correttamente, thread) su Old Games Italia dichiarate che il lavoro di traduzione in realtà è stata una vera e propria riscrittura del gioco in Inform. Perché questa scelta?

W: Purtroppo non c'era davvero altro modo. Il problema centrale sta nel parser, cioè in quella parte del codice che deve smembrare la frase scritta dall'utente e agire di conseguenza. Inform 6 aveva già librerie appositamente create per questo e riadattate per l'italiano, grazie a Giovanni Riccardi. Esisteva anche inform 7, altro linguaggio di programmazione analogo negli scopi ma più vicino ad un linguaggio naturale che a uno di programmazione "standard"; paradossalmente questo tentativo di essere più naturale (esempio: frequente uso di preposizioni) mi risultò molto strano e mi allontanò. Questo è il primo motivo per cui scelsi Inform 6. Non ho nemmeno considerato di usare altro rispetto a Inform 6 e 7, perché voleva dire scrivere un parser da zero, impresa assolutamente non banale. Il motivo definitivo per scegliere Inform 6 però fu l'esistenza di reform.
NDR: giusto per dare un'idea della difficoltà insita nello scrivere un parser efficace per le avventure testuali, ricordiamo che Croshaw, autore della saga "Chzo Mythos" di cui abbiamo parlato su RetroMagazine n. 10, ne sviluppò uno per "Trilby's Notes" commentando al riguardo: "un culo assurdo da sviluppare e non tenterò mai più di farne un altro in vita mia".

GB: Cosa ha comportato l'analisi del gioco prima di iniziare a riscriverlo?

W: Insieme a Ragfox decidemmo che lui avrebbe tradotto, mentre io avrei riscritto il codice. Perciò, specie nelle fasi iniziali, quando non avevo alternative, ho giocato e testato tutti i vari puzzle del gioco originale. Tuttavia questo tipo di reverse engineering non può portare a una comprensione piena del gioco. Abbastanza presto, direi arrivato a dover implementare il troll, scoprii reform. Reform è un simpatico programmino che, dato in pasto il gioco di Zork, fu in grado di generare come output un gustosissimo "spaghetti code" del gioco ossia codice a stento leggibile che peraltro nemmeno si ricompilava, producendo una quantità industriale di errori se ci si provava. A questo problema non vi era in pratica rimedio, perchè gli errori erano tanti e diffusi ovunque nel codice che non era nemmeno facile da seguire: uno su tutti i cicli `for` *erano implementati con dei* `goto`*. Chi è programmatore sa che questo è il male assoluto. Per inciso, anche supponendo di aggiustare tutti i bug e riuscire a compilare, poi il codice va mantenuto: se qualcosa non va, il codice deve essere il più leggibile possibile per essere debuggato. Però erano problemi "tecnici", la logica c'era tutta. Se non sapevo come il gioco si comportasse in una certa situazione potevo seguire il codice e capirne tutti i dettagli. E c'erano già incorporati aspetti che onestamente non avrei pensato di testare, come il gommone che si buca se colpito da una spada o l'effetto del lavarsi i denti con il dentifricio Frobozz (provate, ma dopo aver salvato!). La strategia di solito era: prima si "gioca" il puzzle per avere un po' di comprensione generale di cosa accade, poi si studia il codice per avere tutti i dettagli.*

GB: Vi siete basati sui sorgenti originali di zork? Avete chiesto info agli autori, magari proprio a Mark e Lebling?

W: No, non avevamo gli originali (ora rilasciati pubblicamente su GitHub!), ci siamo accontentati degli spaghetti di cui sopra XD. Non abbiamo neanche chiesto agli autori originali. Chissà, forse se lo avessimo fatto avrebbero rilasciato i sorgenti con qualche anno di anticipo.

GB: L'analisi del gioco e dei sorgenti decompilati ha portato qualche sorpresa? Avete scoperto qualche aspetto nascosto in zork?

W: Sì. In Zork c'è più di quanto uno possa scoprire in una sola run di gioco. Davvero molto di più. Un aspetto interessante di Zork è peraltro che molti puzzle hanno due soluzioni, dunque spesso capita di indovinarne una sola e andare avanti. Esempi famosi SPOILER: >>> la camera rumorosa per via dell'acqua del fiume frigido: si può far calare la quiete sia agendo sulla diga che scrivendo "echo"; il ciclope può essere sia ingraziato con uno spuntino, sia intimorito e spinto alla fuga urlando il nome dell'omicida di suo padre.<<< Infine è molto interessante la parte di codice relativa a tutto ciò che può fare il ladro (ed è un ladro con molta iniziativa).

GB: Dal punto di vista tecnico, quali aspetti sono stati più difficili da ricostruire?

W: Come già accennavo tradurre tutto il codice spaghetti di reform in Inform 6 leggibile non è stato

banale, ha richiesto anzi di riscrivere da capo quasi tutto. L'inglese è stato gestito da Ragfox, che ha fatto un ottimo lavoro lavorando direttamente sul codice in parallelo con me: oltre che di tradurre le frasi inglesi, si è trattato anche di modificare i verbi riconosciuti dal parser (la routine che deve leggere e comprendere l'input del giocatore), cosa che ha richiesto anche un po' di competenze extra. Per quanto riguarda il codice ho due ricordi indelebili. Il primo riguarda la parte di codice che gestisce la navigazione in canotto sul fiume frigido, non perchè fosse particolarmente difficile ma perchè (non ricordo come mai) creai quel codice scrivendolo su una lavagna che ho in garage. Il ricordo è "indelebile" perché a distanza di anni le annotazioni non sono ancora state cancellate (sono tuttavia molto sbiadite!). Per me è come una fotografia, un ricordo di quei giorni in cui mi scervellavo su cose come la Loud Room, la camera rumorosa. Questa stanza è particolare, finchè non si risolve il puzzle, a qualunque input del giocatore risponderà l'eco delle sue parole, anche se l'input forma un comando apparentemente corretto. Il problema è che Inform è stato pensato per agevolare la scrittura di avventure testuali, in cui questa non è un'evenienza comune. In pratica fu un lavoro bestiale, dovetti studiare in dettaglio il manuale di inform, bypassare il parser e fare l'equivalente del leggere l'input un carattere alla volta e stamparlo a schermo. Ricordo che credetti di avere risolto ogni bug in quella camera per almeno due volte, solo per vederne segnalati altri. Decisamente vince il premio come stanza più difficile da programmare.

GB: Quali aspetti avete apprezzato di più?

W: Ok, non sono sadico, giuro. Però devo dire che mi è piaciuto come il gioco non si faccia problemi ad ucciderti, farti girare a vuoto o ancora consentirti di compiere una certa azione irreparabile che ti impedisce di finire il gioco. Quest'ultima è particolarmente cattiva, lo so, però di solito sono segnalate abbastanza chiaramente (come l'uovo decorato che può andare in mille pezzi) oppure non sono del tutto irreparabili. Una chicca notevole sono la merenda e la bottiglia d'acqua che si trovano in cucina all'inizio del gioco: l'avventuriero potrebbe pensare di mangiarsi tutto, solo per scoprire che due enigmi molto successivi necessitano di entrambe. Fortunatamente in questo caso la bottiglia si può riempire al fiume, mentre per la merenda la soluzione è più creativa (giocare per credere!). Ah, e posso assicurarvi che non esiste un machete con cui farsi strada ai bordi della foresta, al contrario di quanto si possa intuire.

GB: Personalmente, trovo il vostro lavoro davvero notevole, ma come ha reagito il popolo degli avventurieri italiani? Avete ricevuto apprezzamenti, critiche o suggerimenti?

W: Applausi! Leonardo Boselli ci ha anche dedicato lo speciale per i 100 video sul suo canale. Più qualche bug report, a conferma del fatto che ci si è giocato davvero. Per quanto forse possa sembrare strano, un bug report viene visto da un programmatore come un interesse verso il programma. Che qualche problema sia inizialmente sfuggito fa parte del naturale ciclo vitale di un programma.

R: Con mia grande sorpresa, trattandosi di un gioco di quasi 40 anni fa e appartenente a un genere non proprio conosciuto dal pubblico italiano, la traduzione di Zork è diventata quasi subito uno dei contenuti più visitati (e spero giocati) del sito di OldGamesItalia. Di certo il progetto non era nato per attirare click su OGI, ma è stato bello vedere l'ottima risposta ottenuta con la pubblicazione di quest'opera. Mi piace immaginare che qualcuno abbia scoperto l'intero genere dell'Interactive Fiction proprio provando a entrare nella famosa casa bianca di Zork, per la prima volta descritta da un testo in italiano.

```
"A ovest della casa - Sei in
un campo a ovest di una casa
bianca. La porta d'ingresso è
sbarrata."
```

GB: Zork II, mirabile seguito del primo capitolo su cui si sta lavorando alla traduzione. A che punto sono i lavori?

W: Non ci sto lavorando di persona, ogni tanto passo su OGI a guardare lo status dei lavori (btw: Frangifiamma è davvero stupendo, grandi ragazzi). A questo giro ci sono i sorgenti originali, quindi i problemi da affrontare sono abbastanza diversi dall'ultima volta. Maggiori dettagli possono darveli solo i programmatori che se ne occupano in prima persona.

R: La traduzione di Zork II è un progetto interno del gruppo di traduzione di OGI, con la preziosa collaborazione di Leonardo Boselli (un altro veterano nel campo dell'IF italiana). Whovian è stato contattato subito, ma non ha potuto partecipare a causa di altri impegni. Anch'io sto mantenendo un ruolo più marginale, pur seguendo con attenzione lo sviluppo dei lavori.

GB: Beyond Zork (letteralmente): prevedete di convertire altre avventure Infocom? Una traduzione di "Deadline" ad esempio, mi farebbe davvero felice (no, non ve lo sto chiedendo ma se vi andasse...)

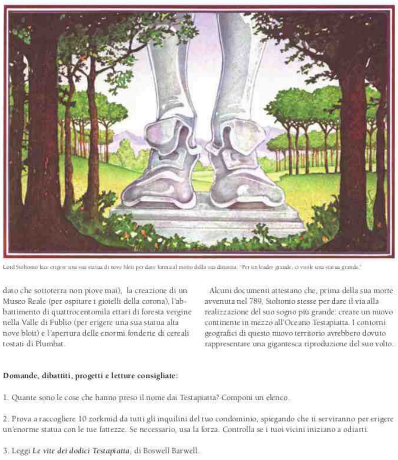
W: Personalmente, non nell'immediato futuro. Però se davvero si riuscisse a far compilare il codice originale di Zork II (ciò su cui si sta lavorando al momento) non vedo evidenti problemi nel compilare poi qualunque gioco. Certo, adattarlo per bene all'italiano potrebbe essere complicato e rappresenta una sfida, ma non poi così impensabile da vincere.

R: Il sogno sarebbe quello di tradurre tutte le avventure testuali della Infocom naturalmente, ma in ambito amatoriale i sogni spesso devono fare i conti con la dura realtà. Però... mai dire mai. Nel frattempo ho già personalmente tradotto i manuali e i materiali di contorno (i famosi "feelies") di Zork II e Zork III, giusto per rendere più piacevole (o tormentata!) l'attesa. Li trovate pubblicati su Old Games Italia.

GB: Consigli per gli aspiranti autori di avventure testuali: quali tool consigliereste e quali giochi dovrebbero ispirare un programmatore che si rispetti?

W: C'è in rete una versione di "Ruins" (gioco già carino di per sé) con tutto il codice Inform 6 commentato. É molto didattica e istruttiva, dà tutte le basi necessarie. L'inform manual può essere usato come riferimento, ma non ha senso leggerlo tutto se non si prova nel mentre di prima mano. Per quanto riguarda i giochi, chiaramente giocate Zork!

E noi di RetroMagazine non possiamo che essere d'accordo con Whovian. Chiudiamo dunque questo articolo ringraziando sia i due autori di Zork I in italiano per il tempo e la voglia di condividere con noi lo straordinario frutto del loro lavoro, che lo staff di Old Games Italia, Gwenelean in particolare, per averci messo in contatto e di fatto contribuito a realizzare questa intervista. Nella sezione dei link



Lord Stoltonio fece erigere una sua statua di nove bloit per dare forma al motto della sua dinastia: "Per un leader grande, ci vuole una statua grande."

dato che sottoterra non piove mai), la creazione di un Museo Reale (per ospitare i gioielli della corona), l'abbattimento di quattrocentomila ettari di foresta vergine nella Valle di Fublio (per erigere una sua statua alta nove bloit) e l'apertura delle enormi fonderie di cereali tostati di Plumbat.

Alcuni documenti attestano che, prima della sua morte avvenuta nel 789, Stoltonio stesse per dare il via alla realizzazione del suo sogno più grande: creare un nuovo continente in mezzo all'Oceano Testapiatta. I contorni geografici di questo nuovo territorio avrebbero dovuto rappresentare una gigantesca riproduzione del suo volto.

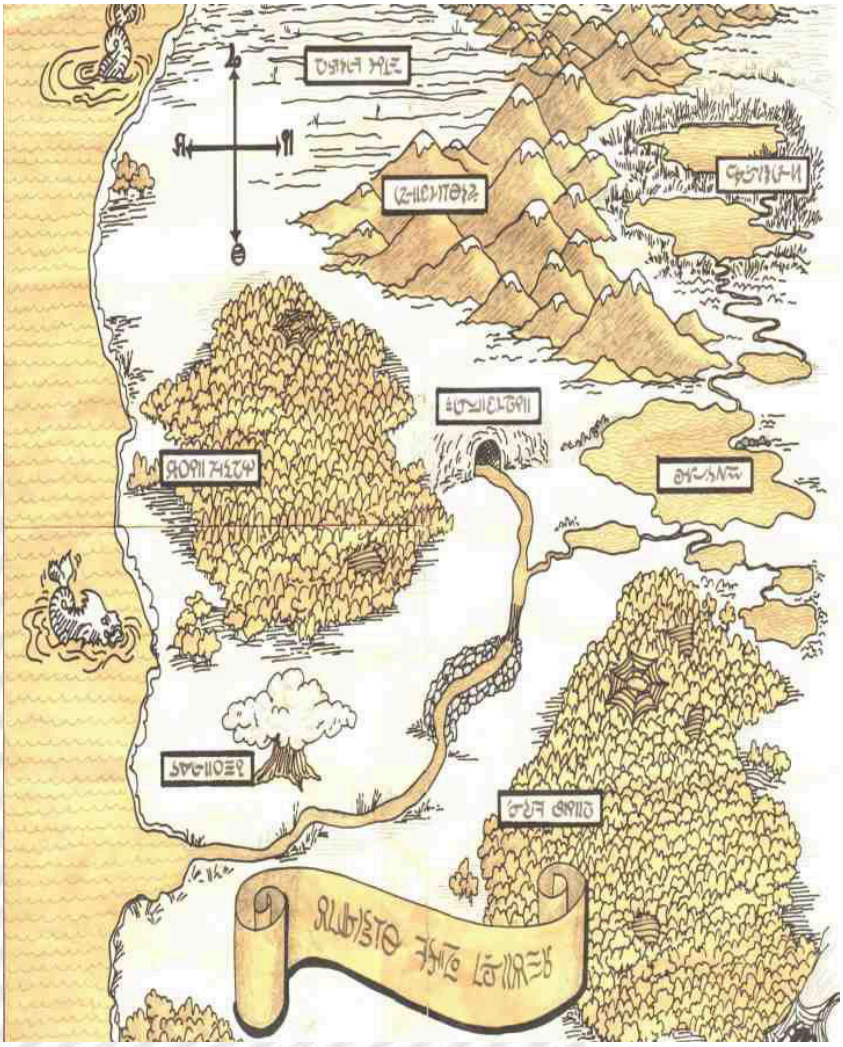
Domande, dibattiti, progetti e letture consigliate:

1. Quante sono le cose che hanno preso il nome dai Testapiatta? Componi un elenco.

2. Prova a raccogliere 10 zorkmid da tutti gli inquilini del tuo condominio, spiegando che ti serviranno per erigere un'enorme statua con le tue fattezze. Se necessario, usa la forza. Controlla se i tuoi vicini iniziano a odiarti.

3. Leggi *Le vite dei dodici Testapiatta*, di Boswell Barwell.

trovate tutte le indicazioni per scaricare il gioco e quale migliore occasione delle vacanze per goderlo in versione italiana? Divertitevi ed arrivederci a settembre.
Ah, e non dimenticate di seguire lo sviluppo della traduzione del secondo capitolo di Zork su Old Games Italia!



Link utili

Zork I in italiano:
http://www.oldgamesitalia.net/traduzioni/zork-i

Zork II in italiano:
http://www.oldgamesitalia.net/category/tag-giochi-citati/zork-ii

Da Infocom a Inform:
http://www.retromagazine.net/getrm.php?id=08

The Pawn
http://www.retromagazine.net/getrm.php?id=12

Giappone 4^puntata: Hard Off / Book Off

di Michele Ugolini

Come promesso, cari lettori, eccomi a voi per parlare di una interessante catena dell'usato nipponico: Hard Off.
Siete curiosi di ascoltare il jingle di questa mega ditta disseminata, attraverso i propri numerosissimi negozi, lungo tutto il Giappone?
Tutte le volte che entro ed ascolto questo jingle, provo una tremenda angoscia, spero che pure voi la proverete, per poi superarla.

https://www.youtube.com/watch?v=yFLYuKUKXoY

Ho forse detto angoscia? Eh si, ben presto, similmente all'esperimento del cane di Ivan Pavlov, proverete un riflesso condizionato ossessivo compulsivo, dal secondo ingresso in poi.
Cosa c'è dentro, di così pericoloso? La risposta è semplice, l'immediato paradiso impalpabile di tutto ciò che abbiamo desiderato durante anni di collezionismo, che tuttavia ci frustrerà all'atto dell'acquisto, anzi del non acquisto, rimuginando sui futuri problemi di spazio nella valigia e soprattutto supervisione doganale. E' bene dare una riletta all'articolo che ho scritto in RM15, riguardo la caccia al retrogaming. Gli accenni delle varie fobie, descritti nel precedente articolo, vi appariranno chiari, non durante la lettura ed aggiornamento di questo articolo e neppure al primo ingresso della sopracitata catena Hard Off: i problemi inizieranno dal vostro secondo ingresso.
Parliamone!
Siete arrivati in Giappone, probabilmente in Narita (Tokyo) oppure a Kix (Osaka). Avete già appreso che siete nella follia pura. Ritmi elevati, serrati, ultra precisi e sovra eccitati. Spese a non finire, prezzi pazzi, alti e bassi. Yen che escono vorticosamente dal portafoglio come accadrebbe con la materia intorno all'orizzonte degli eventi di un buco nero. E' la fine. Anzi no, peggio, questo è solo l'inizio. L'adrenalina e il cortisolo hanno già compromesso i vostri riflessi. Dopo aver assaggiato alcuni delicati spuntini descritti nel precedente articolo di RM15, finalmente finirete in pasto alla vorace e potente catena degli Hard Off, probabilmente anche dei Book Off.
Hard Off (cfr. figura 1) vende semplicemente l'usato nipponico in ottime condizioni, tranne i reparti "Junk" (cianfrusaglia), ottimi per l'acquisto di oggetti da cannibalizzare per i nostri progetti casalinghi.
Book Off (cfr. figura 2) vende l'usato di tutto ciò che è cartaceo e multimediale, quindi troverete fumetti, libri, collane di album, riviste di qualsiasi serie che la vostra fantasia non potrà mai immaginare. Qui potrete trovare scaffalature piene zeppe di vinili, audiocassette, cd, dvd, blu ray, cartucce, nonché qualsiasi videogioco per tutte le console del passato e presente nipponico, memorizzato su numerosissime tipologie di supporto ferromagnetico, addirittura a noi sconosciute, nonché prototipi rarissimi.
Ovviamente il Giappone è una delle culle più importanti dell'elettronica mondiale, quindi dentro Hard Off troverete numerose sottosezioni di prodotti, disseminati nei vari piani dell'edificio che analizzerete. Ogni negozio ha il proprio stile di vendita che tende e reimmettere nel mercato i prodotti più richiesti dalla popolazione della propria zona, fidelizzandola.
Troverete il piano dedicato alla musica con infiniti pianoforti e chitarre. Vedrete il piano degli elettrodomestici. Ci sarà soprattutto il piano del mondo multimediale. Mi spiace, mi è impossibile descrivervi tutto ciò che si trova qui dentro, anche in un solo Hard Off.
Un proverbio diceva che "per riconoscere bisogna prima conoscere" quindi è veramente difficile potervi indirizzare correttamente all'Hard Off giusto per la ricerca di una macchina rara di Sharp, Sony, Brother o altro. In questo caso è una vera caccia al tesoro, silenziosa e senza pubblicità dei prodotti interni, discorso soprattutto valido per lo sterminato assortimento retro gaming e macchine datate. Il pregio di questa caccia consiste nel fatto che, proprio nel negozio più sperduto, nonché meno setacciato da noi retro collezionisti, potrete trovare ciò che nelle grandi città era assente. Allo stesso tempo nelle grandi città potrete trovare una lunghissima fila dei medesimi prodotti più richiesti (per esempio Nintendo Famicom, funzionanti e pronti ad essere visionati ed analizzati), al fine di portare a casa l'oggetto con minor ingiallimento e maggior numero possibile di accessori in dotazione. In un Hard Off di una città minore infatti ho trovato una macchina da scrivere meccanica Brother in condizioni "mint" , perfetta, dell'immediato post guerra, alla cifra equivalente di soli dieci euro. Devo altresì ammettere che varie cartucce del Super Famicom le ho incredibilmente trovate al misero prezzo di 500Yen in qualche Hard Off di Tokyo, sigillate, per esempio F1-GrandPrix e SimCity2000 che, ahimè, la mia coscienza, dopo numerosi anni, ancora mi impedisce di liberarle dall'involucro originale, intatto, sigillato! Magari dentro potrebbe esserci un diamante, oppure, molto più probabilmente, la cartuccia smagnetizzata, chissà?
Vi consiglio di visitare anche Book Off, sappiate che contiene principalmente produzioni cartacee passate, in lingua nipponica, che a

noi occidentali potrebbero interessare come Ken il guerriero, Lamù , Dragon Ball, in condizioni che vanno dall'accettabile al "mint".
Possiedo questi tre titoli citati, li ho presi "near mint", quasi perfetti, preferisco non dirvi quali prezzi mi hanno chiesto per le condizioni "mint", sarà una sorpresa tutta vostra, l'importante è scegliere il prodotto che riterrete più attinente alla vostra caccia e ricordare il limite di peso della valigia: la carta è un grande nemico per la stiva, per questioni di peso e volume.
Più che nelle infinite vie del reparto cartaceo, vi consiglio di passeggiare lungo gli scaffali posti frontalmente alle traverse. Qui potrete trovare, sia appesi che dentro vari cestoni, migliaia di videogiochi del passato. Non sarà difficile trovare a prezzi ridicoli un ipotetico Dragon Quest, Super Mario o il mitico Takeshi's Castle: vi ricordate "Mai dire Banzai"? Lui! Proprio lui! Il folle gioco giapponese di coraggiosi e masochisti atleti in erba, pronti a farsi veramente male, nei modi più disparati e disperati, pur di raggiungere... la sfida finale al Castello. Il grande regista Takeshi Kitano, infatti, organizzava negli anni 80 una sorta di match tra alcune squadre di concorrenti giapponesi, presso un'area allestita (appunto "Il castello di Takeshi") per ospitare le strutture più fantasiose composte da trabocchetti, percorsi militari, pantani fangosi e rulli scivolosi, all'interno delle quali le squadre dovevano raccogliere più punti possibili, limitando nel percorso gli inevitabili e terrificanti traumi fisici, al fine di "assediare" il castello di Takeshi Kitano (cfr. figura 3). Se non ricordo male nessuno ha mai vinto la sfida finale contro Takeshi. Mi informerò. Questo gioco non si poteva proprio paragonare all'adorabile e colorato "Giochi senza frontiere" che si svolgeva nella nostra idealistica "Europa in fase di unione" degli anni 70, 80 e 90.

リユースのリーディングカンパニー
HARD·OFF®
GROUP

Figura 1

Mettiamo da parte i sentimentalismi. Torniamo ai nostri Yen. Dosateli bene mi raccomando. Potrebbe trovarsi dietro l'angolo, proprio nel prossimo negozio, ciò che avete cercato per vent'anni e chissà quando potrete tornare in Giappone!
Attraverso Google Map organizzate il vostro "Safari di caccia" considerando il miglior tragitto possibile di andata e soprattutto di ritorno. Attenetevi ad una tabella di marcia rigorosamente cadenzata nei tempi, al fine di potervi fiondare nel prossimo Hard Off o Book Off. Controllate gli orari di apertura e chiusura dei vari negozi, considerate che in Giappone le aperture del mattino sono posticipate rispetto ai nostri standard. Tenete a mente che, nelle città minori, potrebbero esserci orari più limitati rispetto ad una megalopoli come Tokyo, quindi meglio iniziare direttamente in questi negozi più lontani dal centro, per poi ritornare indietro. Riguardo alla tabella di marcia dei mezzi pubblici, state tranquilli, siete nel paese dell'ultra precisione, i treni Shinkansen nell'arco dei 365 giorni hanno un ritardo medio di circa dodici secondi (si esatto, dodici secondi!). Vi abituerete molto presto a questo genere di servizi, lo stupore svanirà entro pochi

Figura 2

Figura 3

spostamenti, purtroppo "l'effetto boomerang" avverrà inevitabilmente al rientro in Italia.
Un altro consiglio, valido su tutto il suolo nipponico e soprattutto per questa tipologia di negozi dell'usato: non recatevi alla cassa con l'idea di contrattare il prezzo. I dipendenti hanno sostenuto estenuanti colloqui ed una ferrea formazione prima di trovarsi lì davanti ai vostri occhi. I loro modi garbati non sono sinonimo di arrendevolezza, contrattare i prezzi, o addirittura lasciare la mancia, genera un "bug" nel sistema nipponico: è una pratica scorretta e maleducata ai loro occhi, se voi trovate quel prezzo, significa che tutta la ditta ha deciso che quei prodotti/servizi meritano proprio quel prezzo, né più né meno, al fine di offrire simbioticamente la miglior soluzione possibile, sia per l'acquirente che per il venditore. Oltretutto, la decisione del prezzo avviene solo dopo aver analizzato, pulito, supervisionato nel funzionamento e ricercato il vero prezzo di mercato, su tutto il loro territorio, attraverso valutazioni rilevate dal WEB, nazionale ed internazionale. A testimoniare questa approfondita ricerca di un prodotto sarà la monumentale quantità di oggetti dietro i loro banconi/laboratorio, che ancora non sono stati sottoposti a questo rigido processo di immissione sugli scaffali per la vendita al pubblico!
Mi sento in anche in dovere di sottolineare le rigide regole riguardo il peso dei bagagli. Solitamente scelgo una compagnia di volo che mi permetta di viaggiare con due valigie, oltre al bagaglio per la cabina. Inoltre parto sempre dall'Italia con una valigia da stiva poco più grande, che conterrà una valigia poco più piccola: dentro quest'ultima metto tutti i miei oggetti. Invece, al ritorno, vestirò tanti indumenti come uno "Yeti" per permettere, ad entrambe le valigie, di contenere il massimo dei Kg di prodotti elettronici acquistati. E' sempre un dramma. Non il caldo provato come uno "Yeti super vestito", non i troppi Kg che la mia schiena deve far piroettare per Km di aeroporto, non la mancanza di due braccia aggiuntive per sospingere un trolley, un borsone, due pesantissime valigie per la stiva.. bensì la paura per il varco doganale da superare!
Purtroppo ora mi soffermo su questo discorso per aiutarvi a non vanificare, l'intero safari nipponico, in un momentaneo sequestro preventivo doganale, con il rischio di perdere l'aereo.. o peggio: dar adito, ai doganieri, ad effettuare un controllo molto, molto, molto più preciso ed approfondito.
Avete letto l'articolo che ho scritto su RM15? Ve la state cavando bene riguardo la manipolazione mentale tipica degli Jedi? Bene! Sfoderatela nel caso verrete sorteggiati a campione per il controllo delle valigie.
Non mi assumo alcuna responsabilità riguardo ai consigli che sto per darvi, prendeteli con filosofia, sono frutto della mia esperienza personale, probabilmente distorta, quindi...
Se sarete sorteggiati a campione, oppure avete un andamento impacciato o troppo serioso, che salta all'occhio vigile della sala di videosorveglianza, ebbene si, potreste essere sottoposti ad un controllo. Si, si, certo, nessuno trasporta alcunché di illecito, ci mancherebbe. Oltretutto, la rigida legislazione nipponica ha prodotto un timore così elevato, nella propria popolazione, che l'individuo giapponese medio non ha alcuna intenzione di trasgredire la legge, discorso soprattutto valido riguardo i prodotti illegali. Però noi occidentali proveniamo da un sistema più elastico, discorsivo e filosofico riguardo ai concetti di giusto, non giusto, legale ed illegale. Perciò in una nazione così rigida sentiremo la vocina della coscienza che, similmente a Pikachu nei momenti drammatici, chiederà aiuto! Avvertiremo un senso di colpevolezza e nervosismo all'atto delle dichiarazioni doganali, nonché all'eventuale controllo di cotanto materiale dentro quelle numerose, ingombranti, pesanti valigie, che faticheremo a sollevare ed aprire sui loro banconi così alti. Secondo me li hanno posizionati così alti, troppo alti, per rendere più onerosi i nostri sforzi nel sollevare tutti quei kg, con il fine ultimo di minare la nostra serenità e buona fede. Oppure, forse, è solo una mia fisima.
Ad ogni modo, secondo me, i doganieri giapponesi, alla visione di un passaporto italiano, forse complice l'alleanza nell'ultima Guerra Mondiale, forse complice lo stereotipo di "Italia paese di pace e

prosperità", forse complice che davanti ai loro occhi sono presenti "giovani ragazzi" e "ragazzi adulti" che amano svisceratamente l'informatica nipponica... chiudono spesso un occhio al controllo, effettuandolo in maniera apparentemente sommaria. Apparentemente (cfr. figure 4, 5).
Tanto poi le valigie dovranno superare il controllo effettuato per mezzo di cani antidroga nonchè metal detector, apparecchi densitometrici, a raggi X, etc.. Alla fine quindi potranno visionare in maniera tridimensionale e millimetrica i contenuti di ciascuna valigia senza aprire alcunché.
Vi consiglio pertanto di collaborare con loro in maniera completamente trasparente, con oculata e parsimoniosa comunicazione verbale, dato che se alcune frasi che direte non coincideranno con ciò che appare ai loro occhi, vi sposteranno in una sala a parte, in fila dietro ad altre persone da controllare, per ulteriori approfondimenti.
Si certo, mi è capitato più volte, purtroppo.
Massima cortesia, rapidità ed efficienza. Dovrete essere altrettanto carini. Quindi spiegate tranquillamente che tutta l'Italia e soprattutto voi siete animati da un amore incondizionato verso il Giappone, più di ogni altra cosa e soprattutto spiegategli con un sorriso smagliante che siete ultra felici di un controllo così approfondito... rapido... efficiente... che garantisce la vostra incolumità, nonché quella dell'intero aeroplano. Capito? In effetti è tutto vero, i giapponesi applicano le leggi senza cattiveria, unicamente alla lettera, senza filosofia, nel miglior modo possibile, con la massima educazione e discrezione. Ne sarete sorpresi, è una cosa sconvolgente.
Una volta finito il viaggio probabilmente sarete d'accordo con me che la raffinata e delicata compostezza mite del giapponese medio non appartiene ai metodi di approccio occidentale. Naturalmente questa è la mia

Figura 4

visione del loro punto di vista.
Pertanto, una prossemica goffa ed impacciata in aeroporto (e soprattutto in dogana) salterà ancor più all'occhio dell'allenato fiuto di un doganiere nipponico. Ricordatevi che state trasportando ciò che amate, non qualcosa di proibito. I doganieri, durante l'apertura delle valigie, esclameranno stupore osservando la quantità di materiale che avete catturato? Non dovete temere nulla, mostrate gli scontrini che avete scrupolosamente conservato, spiegate tutto in maniera serena e composta. I miei primi errori di tanti anni fa, da impacciato e borioso ragazzino, al quale durante un controllo, gli era stata usurpata la propria libertà...
nei miei viaggi successivi si sono trasformati in una gradevole chiacchierata con i deliziosi doganieri giapponesi, curiosi della nostra nazione così lontana ed esotica ai loro occhi, sogno del giapponese medio per una futura vacanza o viaggio di nozze.
Con questo articolo ho concluso alcuni accenni per organizzare una caccia proficua nel loro territorio. Ovviamente ho coperto solo una minima parte della loro civiltà.

Figura 5

Sicuramente ci sono altri metodi per una caccia più finalizzata ad oggetti di culto: per esempio loro utilizzano molto spesso il WEB, aste online, Yahoo, Rakuten, Amazon, etc..

In questo caso però, l'handicap di non conoscere la lingua giapponese vanifica al nascere la ricerca tramite il WEB nipponico. Non preoccupatevi.

Il materiale che troverete nei negozi sarà comunque infinitamente superiore alle vostre aspettative poiché, ricordatevi, il collezionismo occidentale non è esigente come quello del loro sistema interno ed infine, il 90% del materiale multimediale giapponese è dato in pasto alla loro nazione: l'occidente non lo conoscerà mai!

Ecco qua ragazzi. Il prossimo articolo sarà pubblicato su RM17. Creerò prossimamente un post nella pagina ufficiale di RM, su Facebook, dove mi potrete chiedere quale argomento vorreste venisse trattato nel prossimo numero.

Buone ferie a tutti!

Dragon's Lair (Ti99/4A)

Di Ermanno Betori

Oggi scriviamo di un gioco che è stato una pietra miliare nel mondo video ludico e precisamente della sua trasposizione per il TI99/4A. Dragon's Lair è IL videogioco arcade creato su laserdisc (l'antenato del DVD). La sua particolarità, che fu poi il fulcro del suo successo, consisteva nell'utilizzo di spezzoni di cartone animato combinati insieme per creare tutte le fasi di gioco, realizzati dall'ex disegnatore Disney Don Bluth, fresco del successo del suo "Brisby e il segreto di NIMH". Grazie all'uso dei filmati animati, la cui sequenza di proiezione era gestita tramite il laserdisc, Dragon's Lair poteva sfoggiare una grafica strabiliante per l'epoca, dove lo standard era costituito da una manciata di colori e definizione "a scalini", causa la dimensione dei pixel di quegli anni. Venne pubblicato nel 1983 e pur non essendo il primo titolo su laserdisc, fu quello che portò al successo questa categoria di giochi.

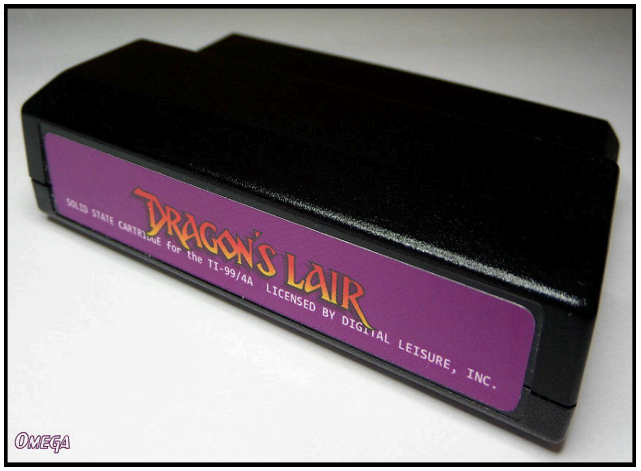
Il protagonista del videogioco è il cavaliere Dirk the Daring (Dirk l'Ardito) che ha il compito di salvare la bella Daphne dal crudele drago Singe che si nasconde in un castello stregato.

Come al solito l'eroe dovrà superare indenne tutte le insidie, giungere nella tana del drago e liberare la principessa Daphne tenuta prigioniera in una sfera di cristallo. Insomma una trama scontatissima.

Ora dobbiamo metterci nei panni di un ragazzino nell'Italia del 1983. I giochi da bar/sala giochi dell'epoca erano Pacman, Scramble, Invader, Donkey Kong e tenere presente che chi giocava con i videogame era considerato un perdigiorno se non addirittura un alienato sociale; entri per fare la tua partitella (facendo la fila, ricordate?) e trovi una moltitudine di tuoi coetanei che sono impazziti peggio che in un concerto dei Led Zeppelin. A spintoni superi parte della calca e vedi LUI: il GIOCO! Dove la bella da salvare è una super topona sexy che manco nei tuoi più perversi sogni avevi osato immaginare!

(nda: Internet non esisteva ancora, il sesso era ancora più tabù di oggi e le riviste osè erano moooolto difficile da procurarsi, direi quasi impossibile per un adolescente)

L'effetto è traumatico: azzeri completamente gli altri video game... hai una sbornia ludica di circa 2 mesi, spendi una enormità di denaro, e se sei bravo finalmente riesci a vincere la partita. Finalmente la parte rettile del tuo cervello si mette in stand-by, e cominci a capire che il gioco è in realtà un cartone animato di estremo impatto audiovisivo con un grosso difetto di giocabilità che è la scarsa libertà d'azione. Infatti il gioco consiste nel fare movimenti istantanei del joystick o premere il bottone corrispondente alla spada, correttamente e nei tempi giusti.



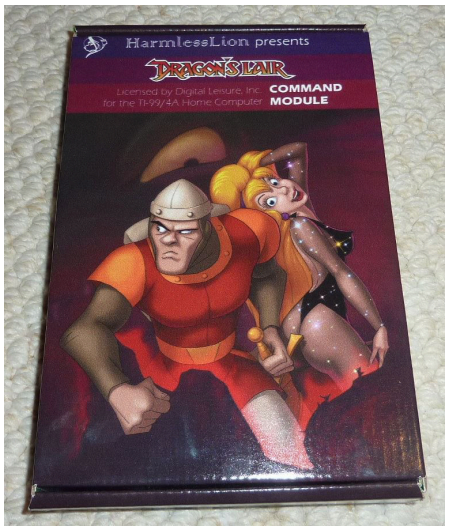
Se il tempismo e la mossa sono corrette il laserdisc procede con il filmato mostrando al videogiocatore i progressi compiuti da Dirk, altrimenti il videogioco propone diversi filmati che ci mostrano la sua morte in funzione dell'errata azione intrapresa. In pratica è necessario memorizzare le mosse da fare in tutti gli scenari presenti nel gioco, perché spesso non è evidente quale mossa salverà Dirk e bisogna pertanto andare per tentativi. Uno Schiacciapensieri offre un'interazione più ampia.

Nonostante questo, fu un titolo che lasciò il segno e all'epoca il sogno di molti videogiocatori era avere Dragon's Lair in casa e vi furono molte conversioni di questo gioco per tutti i più famosi 8 bit come il Coleco Adam, Amstrad CPC, Commodore 64 e ZX Spectrum. Tuttavia, l'hardware dei sistemi casalinghi dell'epoca non rendeva possibile una conversione diretta, e ci si ritrovava con titoli che riprendevano a grandi linee i temi trattati nell'originale,rimescolandoli in un gioco "standard" affrontabile da un hardware senza il supporto di un laserdisc. In pratica di uguale vi era solo il titolo. Una versione piuttosto fedele all'originale fatta in computer grafica, anche se suddivisa in due titoli, Dragon's Lair del 1989-1990 e Dragon's Lair: Escape from Singe's Castle del 1990-1991, è stata quella per i personal computer a 16-bit Amiga, Atari ST, Mac OS e DOS. Lo schema di gioco è quasi identico a quello utilizzato dal laserdisc e l'animazione si avvicina a quella del cartone animato ma con tanti distinguo.

Assolutamente fedeli all'originale furono le versioni successive basate su veri e propri filmati, uscite su 3DO, CD-i e molto dopo sui computer con in dotazione i lettori ottici CD-ROM e DVD, i quali potevano riprodurre alla perfezione quanto era stato visibile in sala giochi. A queste conversioni in tempi recenti si è aggiunta quella per il TI99. Cosa ha di speciale? Tutto e nulla.

Tutto perché è il sogno mistico di un adolescente del 1983 diventato realtà, nulla perché ora l'adolescente ha intorno ai 50 anni (per eccesso o difetto) e magari è un po' tardi per averla...

Negli screenshot del gioco in testa ed in fondo a questo articolo, potete notare che la grafica anche se in formato 192x256 pixel rende abbastanza bene la trasposizione del disegno e (anche se questo non potete vederlo dalle immagini) viene utilizzato un full motion video di ottima qualità, cosa irrealizzabile fino a pochi anni fa, il tutto senza supporto ottico. Il gioco funziona tramite una cartuccia sul TI99 usando la sola consolle senza espansioni che possiede come RAM della CPU solo 256 byte e 16Kbyte



di VideoRam presenti nel chip video TMS9929 (Pal) o TSM9918(NTSC).

Tecnicamente parlando il creatore di questa conversione di Dragon's lair (Mike Brent alias Tursi) ha di fatto inventato una cartuccia per il TI99 che arriva a contenere circa 128Mbyte di dati necessari per contenere tutti i frame dei filmati delle animazioni. Non appena avuto tra le mani, la cartuccia, noto che Brent ha mantenuto tutte le promesse; abbiamo davvero Dragon's Lair funzionante su un computer progettato nel 1978! Oggi dico che è un pezzo di alta progettazione sia hardware che software, ieri avrei gridato al miracolo dell'informatica ludica. I tools e software usati da Mike per costruire questa cartuccia sono stati molteplici, come testimoniato da questo lungo elenco:

- Classic99 (Emulatore)
- Visual Studio (C++ per tools)
- GCC (C per altri tools)
- SOX (per conversione audio)
- ffmpeg (per video conversione e estrazione)
- ImageMagick (per resizing immagini)
- Convert9918 (per conversione immagini)
- dircmd (per procedure batching)
- KiCAD (per schemi elettrici e PCB layout)
- xdt99 (per assembler)
- Photoshop (per manipolazione immagini)
- Krita (per altra manipolazione immagini)
- cmd (sempre per procedure

batching)
• ispLEVER (per sviluppo CPLD, programmazione e test)
• Rigol (oscilloscopio)
• Aoyue (stazione di saldatura)
• Hakko (saldatore)
• AmScope (microscopio)
• Extech (Alimentatore)
• TI-99/4A (per test finali e programmazione)
• F18A (per test GPU code),
• MiniPEB (per compact flash interface)
• OSHPark (per prototipi)
• PCBExpress (per prototipi)
• pcbcart (per i prototipi PCBs)
• PCB4U (per i PCB definitivi)
• Digikey (componenti)
• Mouser (componenti)

Qui sopra abbiamo la postazione di Dragon's Lair presentata da Farmer Potato nel mese di giugno al Midwest Gaming Classic, Wisconsin Center, Milwaukee.

Che altro dire? Se avete un TI-99, procuratevene una copia e buon gioco a tutti!

LINK UTILI

La presentazione di Dragon's Lair dal blog di Mike Brent
http://tursilion.blogspot.com/2018/10/dragons-lair-for-ti-994a.html

La genesi di Dragon lair sul ti99, raccontata da Mike Brent
https://docs.google.com/presentation/d/1u7SOusjQDInq95GrEH2tmIphUAjRg-9YbhR_X25D62A

Il gioco in esecuzione su hardware "reale"
https://www.youtube.com/watch?v=Bd6ZbWKPe4M&feature=youtu.be

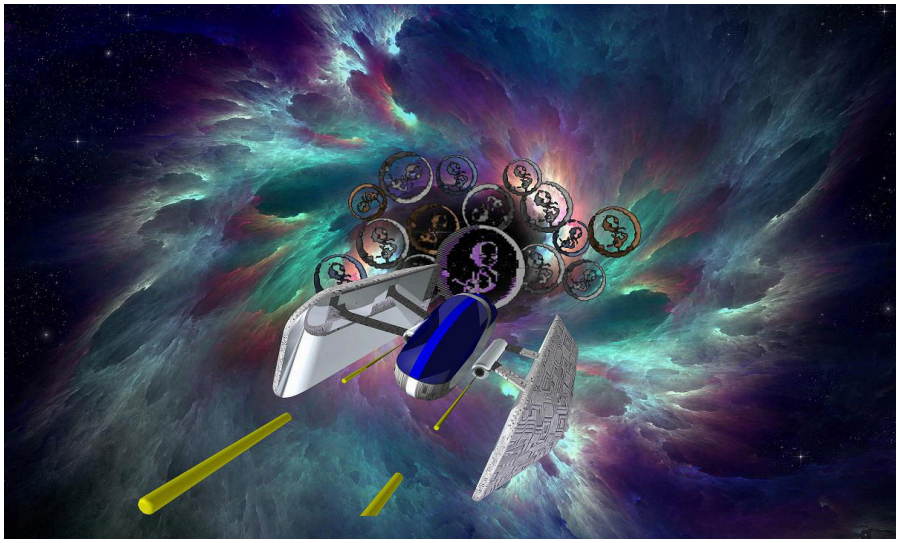
Emulatore classic99
http://www.harmlesslion.com/software/Classic99

Demo del gioco da lanciare sull'emulatore Classic99
https://atariage.com/forums/applications/core/interface/file/attachment.php?id=610370

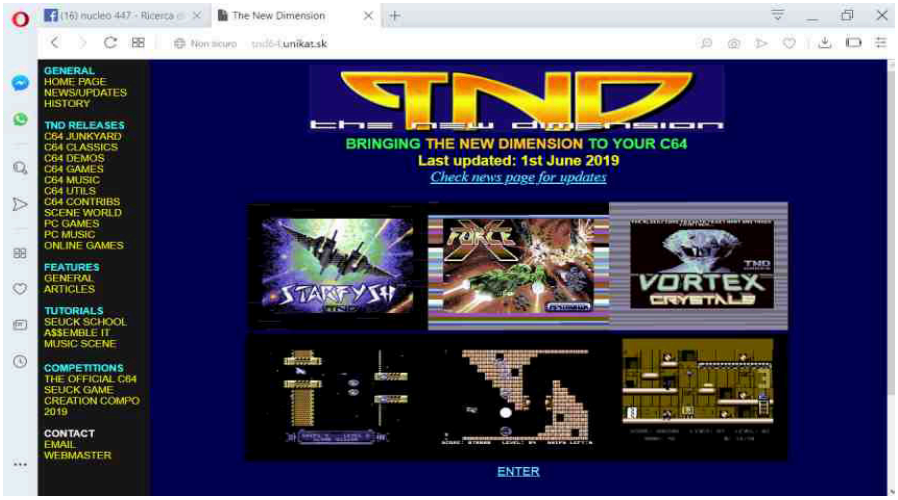
NUCLEO 447 – DIARIO DI SVILUPPO PARTE 1 DI 2

a cura di Associazione Firenze Vintage Bit Onlus (Leonardo Vettori)

20/05/2018 – Guns 'n Roses



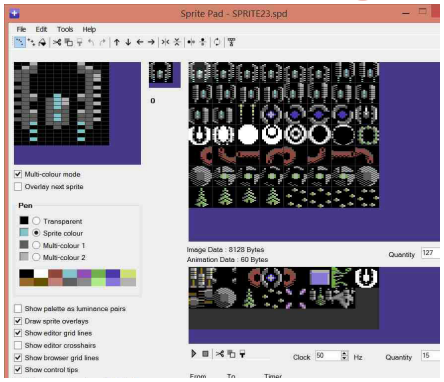
Tutto è iniziato qualche mese fa, quando stavo ripulendo la soffitta. Dietro quelle buste dei vecchi libri di informatica trovo un vecchia scatola di scarpe dove ci sono una quarantina di cassette per il mio Commodore 64. Una bellissima sorpresa per un retrocomputeraio come me, è come riportare alla luce un piccolo tesoro di un valore nostalgico inestimabile. Questo tesoro ha una data precisa, 1988. Ne sono sicuro perché è l'anno in cui ho messo da parte il C64 per passare al PC. Apro le cassette una ad una e leggo la lista dei vari giochi che avevo copiato sulle cassette vergini da 2500 lire. Sono tantissimi e molti hanno nomi tradotti in italiano ma alcuni di questi sono dei propri e veri capolavori che non ammettono nessuna traduzione. Impossibile mission, Bruce Lee, Pit Stop 2, Ghost'n goblins, Nucleo 477, Lode Runner... ehi, aspetta un attimo, Nucleo 477? Proprio lui. Me ne ero completamente dimenticato di Nucleo 477. Uno shoot'em up a scorrimento verticale semplice ma con una giocabilità solida ed una delle grafiche più belle e interessanti che fossero mai state realizzate su un Commodore 64. Ricordo perfettamente che la mappa di gioco era divisa in quattro aree: l'area della vegetazione selvaggia con piante carnivore alla "Io" (capolavoro grafico ingiocabile della Firebird), l'area delle rocce con piccoli vulcani che sparavano enormi palle infuocate, l'area nello spazio profondo e infine l'area supertecnologica con basi spaziali alla Uridium. I nemici erano ben disegnati ed avevano un senso di profondità notevole dato il fatto che faceva un magistrale uso delle ombre e le animazioni erano di una fluidità fuori dal comune. Su ZZap si sarebbe sicuramente preso un voto sopra il 90 ma su ZZap, Nucleo 477, non c'è mai arrivato. Se state leggendo questo articolo avrete probabilmente capito che quel gioco lo avevo disegnato io e che forse sto pure esagerando ma Nucleo 477, a distanza di 30 anni, io me lo ricordo così, bellissimo. Accendo il C64, infilo la cassetta nel registratore, Press Play on tape, ma non accade niente. Maledetta testina del registratore, serve immediatamente un cacciavite per spostare l'azimut del Datassette. Riprovo. Niente. Smuovo la testina, riprovo, niente. Rismuovo la testina, ancora nulla. Poi vengo assalito da un terribile presentimento. Mi maledico per la mia tirchieria da adolescente, prendo la cassetta, la infilo in un registratore "vero" e ne esce "Take me down to the Paradise City" dei mitici Guns N' Roses. Nucleo 477 è perso per sempre e ormai vive solo nei miei ricordi di giovane nerd incompreso degli anni 80. Può darsi che in futuro trovi altre cassette con i salvataggi del gioco ma le speranze sono scarse poiché dovrei prima caricare il Seuck e poi rovistare nelle cassette che non hanno nome ma non trovo nemmeno più la cassetta originale del Seuck. Passa qualche giorno e non mi tolgo dalla mente quell'unico gioco che avevo portato a termine nella mia vita e che avevo spedito alla redazione italiana di Zzap. ZZap non l'ha mai recensito o forse è andato perso nei meandri delle poste italiane, non lo so, ma ad ogni modo mi dispiace davvero averci registrato sopra e di non ritrovare gli altri salvataggi. Da quando ho riscoperto Nucleo 477 vado spesso a visitare il sito www.thd64.unikat.sk dove ogni anno viene indetto un contest come miglior gioco dell'anno e dove il premio finale è la gloria. Ad un certo punto, penso che sto perdendo tempo a rimembrare i tempi passati e che devo prendere una decisione: o continuo a piangerci sopra o mi metto al computer e lo ridisegno per come me lo ricordo. Scelgo la seconda possibilità, al momento l'unica cosa certa è il nome del gioco: Nucleo 477 rev 2.0.



30/07/2018 – Seuck 1987

Sono passati 2 mesi da quando ho iniziato a ridisegnare il gioco. Ricominciare con il Seuck è stato come ricominciare ad andare in bicicletta, una volta che hai imparato non te ne dimentichi più. Dopo qualche decina di minuti mi muovevo tra i vari menù senza problemi ed ero confidente che ce l'avrei fatta a ridisegnare Nucleo 477, ma il vero problema è che dopo 30 anni non è facile lavorare senza il mouse e con solo una finestra alla volta. Il Seuck, per quanto sia un ottimo tool, porta tutto il peso dei suoi 31 anni. Mi sono dato come scopo la partecipazione al contest Seuck 2019. In questi giorni ho conosciuto Pinov Vox che sforna giochi uno dietro l'altro come se non ci fosse un domani. Ma quando dorme? Quando trova il tempo libero per farli? Boh? Io tra lavoro, impegni familiari, imprevisti vari, vacanze, riesco a dedicarmici si e no solo un paio di ore la settimana ed effettivamente non sono andato molto lontano. Sono partito dall'unica cosa che so di saper fare: la grafica. Ho risperimentato il disegno dei 256 caratteri, dei 128 blocchi di 5x5 caratteri e ho verificato l'estensione della mappa. Ho riscoperto tutti i limiti dei colori utilizzabili nel gioco e ho deciso che il colore principale sarà il nero che funzionerà sia per le ombre che per lo sfondo. Il gioco sarà pulito. Tutti i caratteri che non sono neri saranno caratteri "mortali". Il giocatore saprà sempre cosa fare, cioè evitare schiantarsi sulle rocce, sulle piante e sulle strutture della base spaziale. Forse il gioco risulterà un po' troppo spartano anche per il 1988 ma amo la pulizia e voglio che il giocatore non abbia mai dubbi sul da farsi. A maggior ragione la grafica dovrà essere bellissima. Questa è la stessa decisione che presi 30 anni fa. Per quanto ami il C64 devo ammettere che i giochi a scorrimento verticale non sono il suo forte. La ragione è molto semplice e consiste nella dimensione delle schermate. Uno schermo di 160 pixel (320 in modalità monocromatica) per 200 fa sembrare qualsiasi sprite e

fondale "grasso" e "slargato" che stona completamente con lo scorrimento verticale. Gli sparatutto a scorrimento orizzontale invece sono l'ideale perché l'ambientazione sembra più "slanciata" e naturale. Non a caso i migliori sparatutto (per me) sul C64 sono tutti a scorrimento orizzontale, infatti sono quasi tentato nell'usare il Sideways Seuck. Cos'è il Sideways Seuck? Non lo sapete? È presto detto. Negli anni passati bravissimi programmatori, come JonWell, hanno modificato il codice di Seuck migliorandolo e aggiungendo la possibilità di creare shoot'em up a scorrimento orizzontale, ma solo da destra verso sinistra. Le modifiche apportate al codice hanno migliorato un po' la qualità dei giochi prodotti ma non abbastanza da non farli sembrare giochi Seuck, l'impronta del tool originale è ancora ben presente. Il Seuck è proprio un gran bel programma ma in pratica ci puoi fare solo un tipo di gioco, uno sparatutto a scorrimento verticale dove puoi usare solo un tipo di sparo e dove i nemici non hanno un minimo di intelligenza artificiale. Dunque, Sideways Seuck del 2008 o Seuck classico del 1987? Alla fine decido di lavorare



solo con il secondo, quello del 1987. Voglio essere un "retrocomputeraio" fedele alla tradizione, il Seuck non si deve toccare. Se non mi piace così com'è tanto vale che non lo usi e che mi metta a programmare per conto mio una volta per tutte, cosa di cui non sento assolutamente il bisogno.

28/08/2018 – CHARPAD

In questo mese Pinov Vox mi ha dato 3 dritte preziosissime.

1) CharPad, programma fondamentale per disegnare mappe di gioco sul c64 e che gestisce benissimo quelle per il Seuck. Mi ha fatto risparmiare un sacco di tempo e soprattutto mi ha permesso di vedere e di gestire la grandezza della mappa. Senza questo tool mi sarei arreso quasi subito alla rigidità del Seuck.

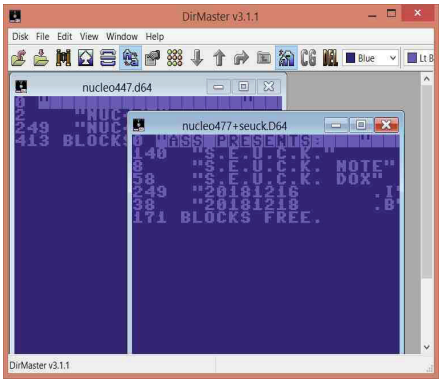
2) SpritePad, fratello di CharPad, anche lui un programma utilissimo anche se lo sto usando poco. La gestione degli sprite del Seuck è buona e quindi riesco a lavorarci bene anche senza SpritePad. SpritePad mi è servito per disegnare i primi Sprite e per farmi un'idea di ciò che volevo ottenere.

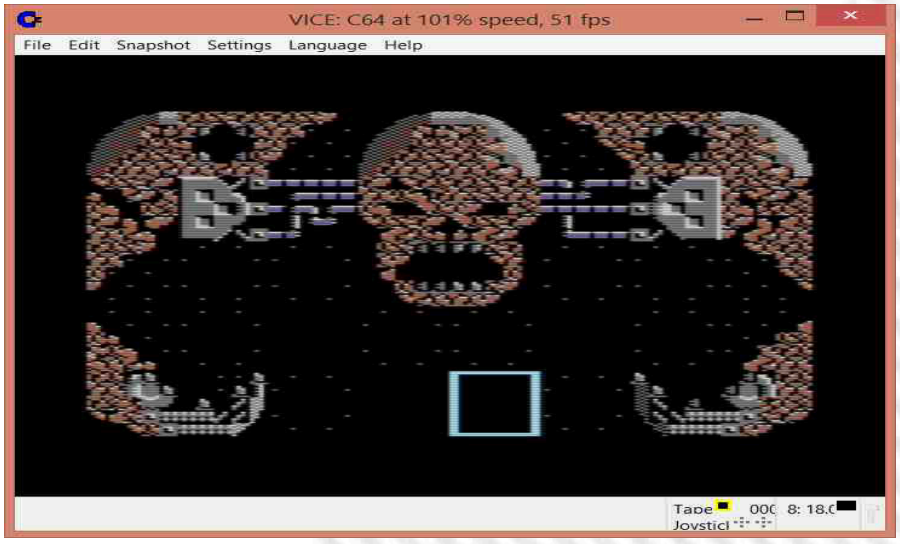
3) DirMaster. Senza questa applicazione non potete mettere niente dentro un disco virtuale per C64, quindi si, questa è importante.

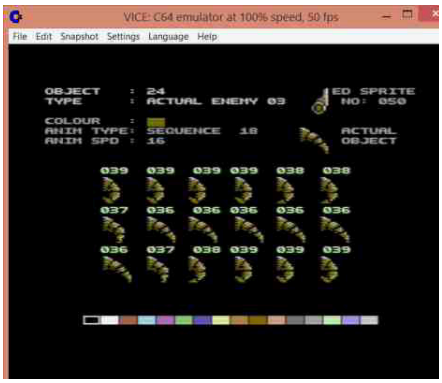
Prima di disegnare la mappa vera e propria ho preso coraggio ed ho disegnato subito un mostro di fine livello. Il mostro di fine livello è importante, senza di quello il gioco non ha senso, almeno per quanto riguarda gli sparatutto anni 80 e 90. Sono contento perché mi è venuto meglio di come l'avevo pensato. In pratica è un mega gorillone fatto di roccia che sputa palle di fuoco. L'ho disegnato realizzando i fondali delle rocce e questo mi fa ben sperare per la poca memoria che avrò a disposizione. È abbastanza minaccioso e tamarro stile anni '80 ma nonostante mi piaccia non si merita di essere il mostro di fine gioco e quindi decido di dargli il terzo posto, anche perché non credo che il mostro 1 e 2 mi verranno meglio di questo. Ce ne saranno 4, perché 4 sono gli stage da affrontare. Ma come disegnare gli altri? Sicuramente continuando a usare i blocchi della mappa e non l'opzione Joint del Seuck. Siccome l'opzione Joint del Seuck mi ha sempre fatto schifo non starò nemmeno a spiegarvela.

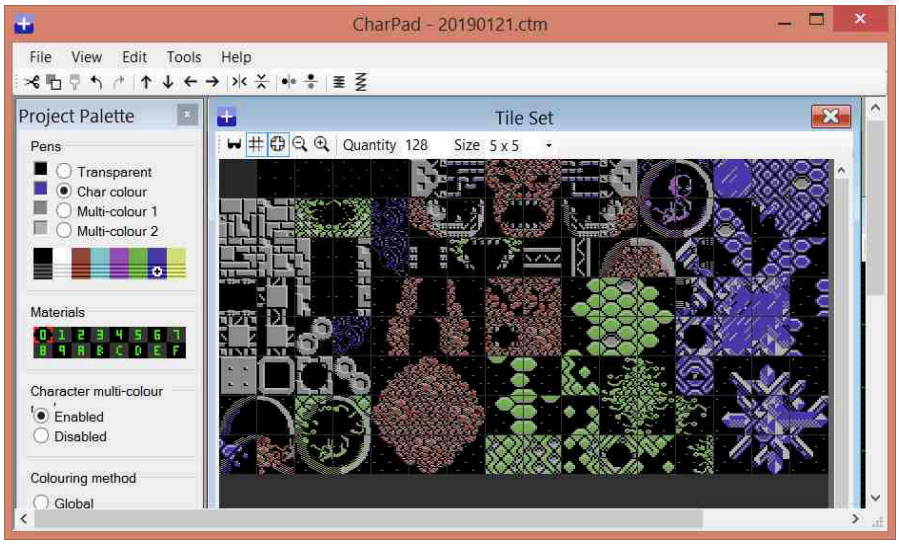
20/09/2018 – L'IMPORTANZA DEL VIAGGIO

Ho terminato la mappa del gioco utilizzando solo Charpad ma non sono per niente soddisfatto. Poiché i miei colori base sono il nero, il grigio scuro e il grigio chiaro, il gioco sembra quasi monocromatico, non ai livelli dello ZX Spectrum ma nemmeno abbastanza variegato da sembrare multicolore. Inoltre i livelli 1 e 4 mi hanno un po' deluso, speravo di fare meglio. Il primo livello è composto da cristalli di ghiaccio con spuntoni "mortali" ma non mi sembrano disegnati molto bene. Un mio amico mi ha giurato e









spergiurato che "si capisce cosa sono" , ma io non voglio che si capisca, voglio che si veda. Il quarto livello invece è composto da una struttura meccanico-tecnologica molto bella ma per quanto i singoli blocchi siano ben disegnati, quando li metto insieme il tutto non funziona e non danno l'idea di un'astronave madre supertecnologica e compatta che contiene il mostro più cattivo di tutti. Per quanto riguarda i livelli 2 e 3 sono abbastanza contento. Il livello 2 è composto da piante e da una struttura esagonale che mi ricorda molto gli alveari di Exed Exes della Capcom. Il livello 3 invece ha delle belle rocce e poi contiene il gorillone finale. Un altro problema della mappa è che questi 4 mondi non sembrano avere nessun nesso nel gioco. Sono potenzialmente belli ma sembrano slegati l'uno dall'altro. Ricontrollando l'intera mappa con Charpad mi domando "che ci stanno a fare tutti insieme? Che relazione c'è tra loro?" Qualcuno potrà pensare che io sia esagerato a chiedermi una cosa del genere. In fondo è solo un gioco fatto con il Seuck, lo scopo del gioco è "uccidi loro prima che loro uccidano te!!". Lo so che il genere sparatutto non necessita di nessuna spiegazione filosofica, ma io me la chiedo lo stesso perché vorrei anche raccontare un viaggio. Se da qui alla fine del gioco non trovo nessuna spiegazione, pazienza, ho 47 anni e ci sono cose peggiori nella vita, me ne farò una ragione prima o poi. Un'altra cosa che mi preoccupa è che ancora non ho trovato un mostro finale migliore del gorillone, degno di rappresentare il "male assoluto" del gioco. Non vorrei terminare il gioco distruggendo il "classico" megacervello spaziale alla Metroid. Vorrei trovare qualcosa di alternativo e disturbante ed in effetti qualcosa di disturbante e politicamente scorretto l'ho trovato e l'ho disegnato pure bene, ma non so per quale motivo dovrebbe essere proprio lui il mostro finale (naturalmente non vi dico cos'è), non c'è niente durante il percorso che suggerisce un finale del genere. Lo so che è solo uno sparatutto a scorrimento verticale e non un'opera filosofica ma questo gioco io lo sto facendo più per me che per i giocatori. Se non piace a me non c'è ragione di condividerlo con voi.

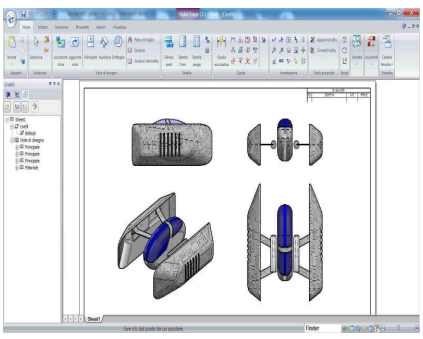
10/10/2018 – Le dimensioni della mappa

La memoria a disposizione per la mappa totale è di 8 blocchi orizzontali per 512 blocchi verticali. Naturalmente l'esperienza di gioco può essere allungata con l'editor di livelli riutilizzando più volte le porzioni di mappe disegnate, ma io volevo fare una mappa tutta diversa, senza riciclare nulla, non so se riesco a spiegarmi. Ad ogni modo a ogni porzione di mappa ho dedicato circa 100 blocchi in verticale. La mappa di gioco è composta come segue:

1) 125 blocchi del mondo di cristallo e mostro finale di primo livello.
2) 125 blocchi del mondo vegetale e mostro finale di secondo livello.
3) 125 blocchi del mondo di rocce e mostro finale di terzo livello.
4) 125 di mondo tecnologico con richiami agli altri 3 mondi e il mostro finale.

Mi piace come ho composto la mappa ma nonostante tutto non mi convince e non capisco perché. Forse dovrei passare a lavorare

direttamente sul Seuck per avere una visione diversa del gioco ma è troppo comodo abbandonare Charpad per i menù del Seuck. Poi arriva una gradita sorpresa. Un amico di una mia amica mi regala il suo Commodore 64 a patto che non lo rivenda (e ci mancherebbe altro!!!) e sento il bisogno di farci girare Nucleo 477. Fino ad ora il gioco ha girato solo sul Vice e adesso lo voglio veder girare sulla macchina vera, soprattutto se è un regalo. Prendo il Seuck e salvo il gioco come gioco definitivo. Lo metto su un disco virtuale con DirMaster. Poi lo metto sulla SD che infilo nella Princess del Colombari e carico il gioco sul C64 vero. Non succede niente. Riprovo altre 3 o 4 volte e alla fine decido per la prova del nove, lo faccio partire sul Vice e scopro che il gioco nel formato finale non me lo legge nemmeno l'emulatore. Non so cosa non vada e onestamente non me ne importa nemmeno. Riparto da zero, prendo il Seuck e lo metto su un disco virtuale nella SD insieme ai salvataggi "puri" del gioco. Poi infilo la SD nella Princess del Colombari, carico il Seuck sul C64 e da lì carico tutti i salvataggi e voilà, per la prima volta vedo girare Nucleo 447 su un C64 vero. La prima cosa che noto è che perdo 16 pixel in altezza rispetto alla mappa pensata con Charpad, cavolo!!! Mi sono completamente dimenticato che nella parte bassa dello schermo di un gioco Seuck c'è il punteggio con le vite a disposizione e quindi mi viene tagliata la testa del gorillone, però è bello lo stesso e mi da soddisfazione. Mi viene voglia di iniziare a lavorare sugli Sprite, ma prima faccio una prova. Lascio girare il gioco dall'inizio alla fine senza interazioni, solo per vedere quanto dura. I 4 mondi con lo scorrimento a velocità 1, cioè quella più lenta, più gli schermi a tempo per i mostri finali impiegano in totale 15 minuti per essere visti tutti. Tanto lavoro per 15 minuti di gioco? Ne vale la pena? Si, ne vale la pena, va bene così. Lo faccio girare un'altra volta, i 4 mondi mi sembrano sempre un po' slegati. Sono abbastanza soddisfatto ma qualcosa deve cambiare.



17/10/2018 – Dominator

Dominator è un gioco del 1989 della System 3 che io non conoscevo perché in quegli anni avevo abbandonato il Commodore 64 per un mediocre clone PC, un Amstrad con Hard Disk da 20 Mega. L'ho scoperto qualche giorno fa per caso. Non ricordo nemmeno cosa stavo cercando su Google per quando è saltato fuori, ma fatto sta che ha attirato la mia attenzione. Zzap inglese n. 53 gli dette un voto abbastanza misero: presentazione 71%, grafica 72%, musica 70%... il voto finale 75%. In poche parole diceva che era semplicemente un altro shooter a scorrimento orizzontale inferiore a Io, Delta, Armalyte e Salamander. Ho scaricato da Youtube il video completo del gioco e sono rimasto colpito dalla grafica. Non è la più bella mai apparsa sul C64 ma ha qualcosa di accattivante e a tratti addirittura di perverso. Mi piace. Dominator in scorrimento verticale. Dominator in scorrimento orizzontale. Mi studio il video un paio di volte e poi capisco che man mano che si va avanti, dentro questo mondo "malsano", lo sfondo cambia poco a poco fino ad arrivare al mostro finale. Questa variazione abbastanza costante nella grafica è il sistema migliore per raccontare il viaggio che la nostra astronave deve affrontare per distruggere questo mondo maledetto. Nota stonata di questo gioco è il passaggio da gioco a scorrimento verticale a quello orizzontale. Non ne capisco il senso e ad ogni modo questo problema non si pone con il Seuck. Per quanto riguarda la grafica posso tentare di fare qualcosa del genere. Inoltre ho apprezzato gli aculei che escono dalla vegetazione e decido di adottarli anche io e li metterò nel mondo vegetale.

Per ora è tutto, il resto del racconto vi aspetta nel prossimo numero di RetroMagazine, nel frattempo vi auguro buone vacanze e se volete, divertitevi con nucleo 447 scaricandolo da:

http://tnd64.unikat.sk/Seuck_Compo_2019.html#Nucleo447

OASI ESTIVE

di Querino Ialongo

L'estate probabilmente è un periodo difficile per organizzare fiere e mercatini poichè c'è il rischio concreto che le città si svuotino per le classiche vacanze. Ciò nonostante vi vogliamo segnalare due importanti eventi che sono delle vere e proprie oasi in questa calda estate.

Il primo evento è il **San Marino Comics** che si terrà il 23, 24 e 25 agosto.
Organizzato dall'Associazione Culturale San Marino Comics, la manifestazione ha lo scopo di promuovere e divulgare, all'interno della Serenissima Repubblica di San Marino, tutto quello che riguarda fumetti e libri (nazionali e internazionali), inoltre cinema d'animazione (sempre nazionale e internazionale), giochi da tavolo, giochi di ruolo, sigle TV e colonne sonore di cartoni animati, film e telefilm nonchè un'importante area dedicata al fenomeno del cosplay. La divulgazione di tali voci potrà svilupparsi anche attraverso collaborazioni strette con altre associazioni culturali che operano in settori compatibili sia all'interno che all'esterno della Repubblica stessa. Infatti questa collaborazione è testimoniata da un'importante area retro che è stata interamente affidata a Retroacademy.
Si tratta di un gruppo di persone unite dalla passione per il retrocomputing e il retrogaming, per la programmazione, la ricerca e lo sviluppo di idee indipendenti.
Retroacademy è un insieme di professionisti ed esperti che vogliono mettere a disposizione delle nuove generazioni la passione e la cultura per il passato informatico. Il gruppo nasce principalmente per diffondere a 360 gradi la cultura informatica, dal videogioco all'esposizione di classici di un tempo, fino alla programmazione su macchine oggi purtroppo ritenute erroneamente obsolete.

Il secondo appuntamento che vi segnaliamo è il **Monfalcomics** che si terrà il 24 e 25 Agosto ed è un evento creato dalla Pro Loco di Monfalcone.
All'interno della manifestazione potrete trovare un'importante gara cosplay, un'area dedicata agli spettacoli, una ai workshop, una al mercatino e alle mostre ma ci sarà soprattutto un'importante area retrogames che sarà interamente curata dai Retro Corner.
Si tratta di un gruppo di grandi appassionati ed esperti di videogiochi, provenienti dal Friuli Venezia-Giulia che ha scelto di mettersi in gioco al festival monfalconese dedicato alla Geek Culture. Sabato 24 e domenica 25 agosto, presso l'oratorio San Michele di via Mazzini, i Retro Corner vogliono mettere a disposizione dei visitatori, in maniera libera e gratuita, le loro numerose console provenienti direttamente dagli anni 80 fino ad arrivare ai primi 2000, insieme ai giochi che le hanno rese famose; l'intento è quello di far rivivere le stesse emozioni di chi quegli anni li ha vissuti, e di far provare sensazioni a chi è troppo giovane e purtroppo in quelli anni ancora non c'era.
Dodici saranno le postazioni con le quali i partecipanti potranno giocare, partendo dal vecchio Nintendo Nes, fino ad arrivare alla PlayStation 2. Non mancherà un angolo dedicato al retrocomputing, il tutto sotto la supervisione di un personale qualificato che saprà spiegare la concezione, i meccanismi e l'evoluzione di un videogioco, fino ad approfondire tematiche quali l'immancabile crisi dei videogiochi, e la famigerata console-war che vide protagoniste SEGA e Nintendo soprattutto negli anni novanta.

Eventi Agosto 2019

SAN MARINO COMICS
23-24-25 AGOSTO

MONFALCOMICS
24-25 AGOSTO

L' innovazione al momento sbagliato.
IL COMMODORE CDTV

a cura di Associazione Firenze Vintage Bit Onlus (Luca Cusani)

Commodore è stata da sempre sinonimo di innovazione mai al passo con i tempi ma proponendo prodotti che erano avanti rispetto alla concorrenza. Esemplare fu il Vic-20 costruito e distribuito per essere il primo computer accessibile a tutti. Il Commodore 64 primo computer al mondo ad avere infranto il record di vendite, primato che ancora oggi detiene; per Amiga invece non avvenne, nonostante proponessero un sistema operativo a finestre quando i Personal Computer compatibili IBM ancora utilizzavano un sistema basato sulla linea di comando come il DOS.
Nel 1990 Commodore fa un ennesimo grande passo annunciando un nuovo prodotto destinato alle famiglie, che possa stare in tutti i salotti di casa che possieda un design accattivante e che intrattenga il pubblico con tecnologia multimediale.

Nasce il CDTV: l' acronimo significa Commodore Dynamic Total Vision e viene presentato al CES (Consumer Electronic Show) di Las Vegas nel Gennaio del 1991.Tanta è la tecnologia che contiene, tra cui: lettore CD-ROM, telecomando a infrarossi e un esteso corredo di accessori che lascia perplessi. Una tastiera, un mouse, lettore floppy disk e una originalissima Trackball che ingloba due porte standard a 9 pin che consentono di poter collegare due Joystick. Tutto questo accorpato in un design modernissimo per l' epoca.

Perché questa perplessità ? Perché invero trattasi di un Amiga 500 espanso a 1 Mbyte confezionato all' interno di un case accattivante di stampo metallico, dotato di una alimentazione interna con ventola, schermo digitale e un imponente lettore CD-ROM.
Ma andiamo con ordine!
Il progetto si chiamava CD1000 e potevate trovare il nome nella targa descrittiva apposta sotto la macchina, un po' per dare una sorta di continuità al successo dell' Amiga 1000 che fu capostipite dell' era a 16Bit per la Commodore.
Il primordiale lettore CD-ROM non era simile a quelli cui siamo abituati a vedere oggi, ma presentava il famoso "CADDY": ovvero una custodia apribile dove si inseriva il Compact Disc per poi inserirlo nella fessura del posto sul frontale del CDTV, certamente non il massimo della comodità.

Veniamo alle caratteristiche tecniche:
CPU Motorola 68000 (Bus indirizzi 32 Bit, dati a 16 Bit).
Chipset OCS.
Memoria di 1MByte di Chip RAM (Espandibile).
ROM 512 KB di cui una metà era destinata al Kickstart e la restante per il Firmware CDTV.

Come possiamo leggere dalla descrizione tecnica, tranne per il Firmware dedicato, tutto era

analogo alle caratteristiche dell' Amiga 500 plus che conserva il primato di essere il primo computer al mondo equipaggiato con lettore CD-ROM.

Il software disponibile era molteplice e vario ma tutto legato sempre al concetto di "Multimedialità" quindi, fatto di enciclopedie elettroniche, programmi didattici anche per bambini e potevano essere ascoltati i CD-ROM musicali. Infatti il CDTV era anche un ottimo lettore musicale con una completa interfaccia e invitante schermata di scelta delle tracce musicali, il tutto gestibile dall' avveniristico telecomando a infrarossi che poteva essere azionato stando comodamente seduti sul proprio divano preferito.
Non per ultime è bene soffermarsi sulla presenza delle porte "MIDI" che davano l' opportunità di poter utilizzare la macchina anche a un pubblico di musicisti. Queste importanti porte sono state sempre assenti ingiustificate che hanno in parte penalizzato i computer della serie Amiga ove da questo punto di vista su Atari (ST ed altri) erano presenti.
Quindi possedere un " SET TOP BOX " completo del CDTV permetteva un utilizzo ben più ampio rispetto al solo lettore multimediale per la quale era nato, e il problema della sua scarsa diffusione fu proprio tutto su questa sua natività.
Commodore a differenza degli altri

prodotti presentati in passato, dove la pubblicità e la distribuzione capillare di vendita anche in quei negozi non propriamente di Computer (Negozi di Giocattoli, Elettronica, e anche, si... Supermercati), scelse d'imporre e vendere il prodotto come un qualcosa destinato all' Home Entertainment quindi, lo potevamo trovare solo ed esclusivamente nei negozi muniti di reparti HI-FI o videoregistratori.

Questo perché Commodore non voleva che il prodotto risultasse un Home Computer, ma qualcosa di completamente diverso anche se alla fine proprio di un Home Computer si trattava.
I potenziali acquirenti quindi non avrebbero mai conosciuto le vere potenzialità della macchina che: se invece avessero avuto il modo di provare e conoscere di che prodotto si trattasse, avrebbero sicuramente acquistato. Tuttavia Commodore aveva paura che il CDTV andasse a intaccare le vendite dei prodotti Amiga. Il costo del CDTV non era alla portata di tutti, ben 750.000 Lire per la versione "base" ovvero il corpo macchina e il telecomando: questo dava la possibilità di essere utilizzato come selettore dei menù, mouse o pad da gioco. L' infrarosso che stava prendendo sempre più piede diede un altro primato a questa macchina, ovvero di avere un vero e proprio pad da gioco e un mouse dedicato con questa tecnologia.
La "Grolier" nota casa editrice che produceva enciclopedie sfruttò molto questa macchina in Italia quando usava ancora il porta a porta. Il rappresentante della Grolier che facevamo accomodare ci presentava due opzioni per vendere la sopracitata enciclopedia.
O quella cartacea ma sconsigliata perché ingombrante.
O quella multimediale dove incluso nel prezzo c'era appunto in dotazione il CDTV.
Quest' ultima opzione veniva data in ulteriori tre scelte:
Il CDTV in versione base; " Macchina più telecomando ".
il CDTV in versione top ; " Macchina, telecomando, tastiera, lettore floppy, mouse e addirittura il monitor 1084S , tutto elegantemente abbinato in colore nero, dove il sedicente informatore

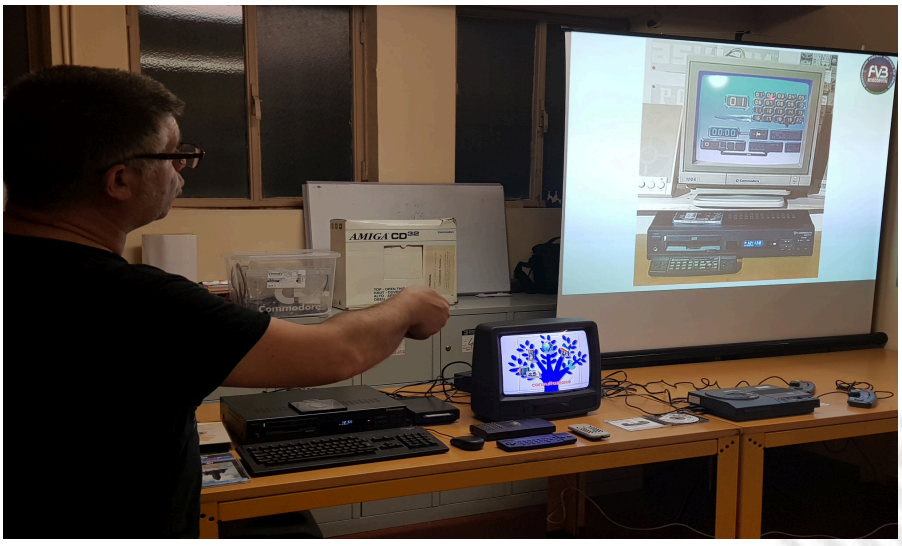
ci avrebbe convinto all'acquisto di quest' ultima, dicendoci che avremmo avuto un vero e proprio computer in casa.

Un CDTV con questa configurazione era, come ho già accennato un Amiga 500 a tutti gli effetti e come tale poteva essere usato tranquillamente anche per giocare con tutti i titoli presenti all'epoca alle applicazioni o i software destinati alle macchine Amiga.

Infine.... Come terza scelta :" Avete in casa già un Amiga 500? Non ci sono problemi! Sempre il sedicente rappresentante Grolier. L'enciclopedia veniva fornita insieme a un lettore CD-ROM: la periferica A570 che collegata alla porta laterale di un 500 trasformava letteralmente la macchina in un CDTV con tanto di bella schermata iniziale dedicata.

Il CDTV fu prodotto fino a 1993 e purtroppo fu una macchina che contribuì alla fine di Commodore che sarebbe sopraggiunta l' anno successivo, proprio per le enormi risorse spese per aver reso reale questo ambizioso progetto. Purtroppo il mercato non rispose con l'entusiasmo sperato In modo speciale con l'avvento dei primi PC dotati di Windows e la diffusione di Internet, l'interesse di queste enciclopedie nonché dei sistemi multimediali decadde in tempi piuttosto repentini.

Oggi è diventata molto ambita e ricercata tra i collezionisti in quanto è protagonista di un grande interesse proprio per la modernità e l'unicità dell'oggetto, dove un nutrito ecosistema di persone compreso il sottoscritto adorano questo bel pezzo di Hardware e le cifre per avere un esemplare in ottime condizioni richiede talvolta anche somme in euro abbastanza importanti; queste oscillano tra i 150 e i 250 per la versione " base ", ma non è detto che alcune volte la fortuna possa strizzarvi l'occhio per trovarne una accatastata tra i videoregistratori o lettori DVD nei vari mercatini dell' usato.



I problemi stanno nel reperire le periferiche dove la tastiera, lettore Floppy Disk e Mouse non si sono così diffusi, non parliamo poi se vi viene in testa di trovare il Trackball, il Monitor o il mouse a infrarossi. Dove il costo della periferica supera anche il valore di un CDTV se perfetto e con scatola.

E pensare che questa bellissima e

tecnologica macchina di colore nero che oggi ogni collezionista agogna, all' epoca non la voleva più nessuno.
Già... il colore nero.

Prima di lui questa scelta di colore già fece vacillare Commodore con la serie 264 (C16- Plus/4 e 116) quindi non molto fortunato...

Dopo il CDTV nel 1994 fu presentato il CD32... Anch'esso caratterizzato dalla stessa sfortuna e anch'esso di colore nero....

Caratteristiche del CDTV32

◊ **CPU Motorola 68000 (Bus indirizzi 32 Bit, dati a 16 Bit);**

◊ **Chipset OCS;**

◊ **Memoria di 1MByte di Chip RAM (Espandibile);**

◊ **ROM 512 KB di cui una metà era destinata al Kickstart e la restante per il Firmware CDTV.**

Bilancio di una notte di mezza estate

La stagione calda per me resta sempre il momento in cui sospendere ogni attività e dedicarsi ad altro, forse perchè ho una visione “scolastica” del calendario. Magari si finisce per lavorare di più ed uscirne stressatissimi ma di fatto è questo il periodo in cui tutti i progetti giungono ad una sorta di pausa temporanea per poi ripartire da Settembre con nuova linfa vitale.

Il bilancio della stagione 2018-2019 di RetroMagazine è stato decisamente positivo.

E' strano per me pensare che sia passato solo un anno da quel numero 8 di cui scrissi l'editoriale. Mi sento parte della famiglia da parecchio tempo e condivido con i ragazzi l'entusiasmo per tutti i download che vengono conteggiati ogni mese. Sì, la rivista va decisamente bene e per questo non posso che essere fiero sia dei miei colleghi che dell'affetto dimostratoci dal pubblico nel credere, ogni giorno di più, al sogno di **Francesco** (a proposito, auguri papà).

In quest'ultimo anno siamo maturati sotto diversi aspetti, affrontando i primi piccoli problemi ed i dilemmi tipici di chi non è più parte di un mini-progetto nel quale sono coinvolte poche persone. Ogni numero ormai consta di un numero di pagine costantemente sopra le sessanta unità. Sono arrivati tanti nuovi redattori, qualcuno ha lasciato e qualcun altro (come accennato sopra) ha condiviso con noi una tra le gioie più belle che la vita riesce a donarci.

Ci siamo incontrati, come narrato nell'editoriale del numero scorso, portando il rapporto ad un livello successivo rispetto ai semplici collaboratori di penna e sì, insomma: "siamo una squadra fortissimi" (cit.).

Il bilancio non si può fare però considerando solo il seguito ottenuto da RetroMagazine, ma anche valutando gli oneri che tale seguito comporta.

Sono convinto che le oltre duemila persone che ogni mese scaricano la rivista meritino di ricevere sempre il meglio che il nostro entusiasmo sappia trasmettere.

Più ancora che una costanza nella pubblicazione o un numero di pagine in crescendo, RetroMagazine deve poter garantire articoli capaci di far trasparire l'amore che ognuno di noi prova per questa passione comune. Secondo me non è mai mancato fino ad ora, ma voi non esitate a tirarci le orecchie se pensate sia giusto farlo!

Prima di salutarci, vorremmo ringraziare tutti i gruppi Fb che ci permettono di pubblicizzare ogni uscita della fanzine. Altri doverosi ringraziamenti vanno al gruppo "***8 Bit RetroProgramming Italia 8 bit ed oltre - Associazione culturale***", nostro partner consolidato la cui collaborazione si dimostra sempre più preziosa e proficua, ed alla community ***"Old Games Italia"***, che pubblicizza ormai da tempo ogni nostro numero, seguendoci con simpatia.

Auguriamo a tutti voi buone ferie e... non temete, RetroMagazine tornerà presto!

Starfox Mulder

Disclaimers

RetroMagazine
Anno 3 - Numero 16

Direttore Responsabile
Francesco Fiorentini

Vice Direttore
Marco Pistorio

Luglio 2019