UNIVERSITÀ DEGLI STUDI DI SIENA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

CORSO DI LAUREA IN MATEMATICA

IL PROGETTO MAME:
REVERSE ENGINEERING E MACCHINE DA GIOCO

Relatore:
Chiar.mo Prof. Alfio Andronico

Tesi di Laurea di:
Nicola Salmoria

Anno Accademico 2001/02

A Francesca

INDICE

INTRODUZIONE

> *Le persone "serie" hanno spesso sottovalutato l'importanza del gioco e frustrato sistematicamente la tendenza, sviluppatissima nei bambini ma presente anche negli adulti, a giocare. [...]*
> *Da bambino qualche volta mi hanno fatto sentire colpevole per aver "fatto i balocchi" anziché le cose "serie", ora so che le poche cose serie che ho fatto derivano semmai da quei balocchi.*
>
> Roberto Magari, *Presentazione.* Atti del convegno sui giochi creativi, Siena 11-14 giugno 1981.

MOTIVAZIONI

Questa tesi non è un punto d'arrivo, ma il resoconto del lavoro svolto finora in un progetto da me iniziato e seguito durante gli ultimi sei anni, denominato MAME[1], acronimo di *Multiple Arcade Machine Emulator*.

L'interesse per il problema trattato nell'ambito del progetto nacque in modo del tutto casuale, grazie ad un piccolo programma che avevo scaricato da Internet in un momento di noia. Si trattava di un *emulatore*, cioè di un programma che riproduceva su un PC il funzionamento di un altro *hardware*.

L'emulatore che avevo scaricato era speciale per vari motivi.

Anzitutto, si trattava dell'emulatore di un videogioco *arcade* (cioè da sala giochi).

[1] Pronunciato "mame", non "meim".

In secondo luogo, il gioco emulato era *Pac-Man*, uno dei più famosi di tutti i tempi, al quale sono associati vividi ricordi della mia infanzia.

Inoltre, insieme all'emulatore era fornito anche il suo codice sorgente, quindi si poteva capire com'era fatto e apportarvi modifiche.

Ma soprattutto... l'emulatore non funzionava! O meglio, funzionava, però era una versione preliminare con evidenti difetti: i colori erano sbagliati, non c'era sonoro e lo schermo era quadrato anziché rettangolare.

Molti credono che mi sia dedicato all'emulazione perché sono un appassionato di videogiochi; in realtà non è così. Sebbene sia affascinato dai videogiochi arcade, non sono mai stato un gran giocatore. La mia principale passione, da quando, quasi vent'anni fa, ricevetti in regalo il mio primo *home computer*, è sempre stata la programmazione. Quando scaricai quell'emulatore "fatale", mi trovavo in un periodo in cui non riuscivo ad esercitare questa passione, perché avevo iniziato da poco ad usare il PC, un'architettura che non conoscevo e che non mi offriva gli stessi stimoli del Commodore Amiga usato fino a poco tempo prima.

Adesso, inoltre, mi rendo conto che, pur senza saperlo, sono anche sempre stato interessato al *reverse engineering*. Infatti, molti dei programmi che avevo scritto per Amiga erano modifiche al sistema operativo che si potevano compiere solo conoscendone a fondo il funzionamento. Ad esempio, ero riuscito a cambiare la gestione delle icone, rendendole più colorate e riconoscibili; questa miglioria ebbe una tale diffusione che le versioni successive del sistema operativo ne inglobarono le funzionalità mantenendo la compatibilità con il formato da me utilizzato.

Per tutti questi motivi, uniti ad una mia innata tendenza a cercare di aggiustare quello che non funziona, dopo aver scaricato l'emulatore mi procurai gli strumenti necessari per ricompilarlo, mettendomi immediatamente all'opera per renderlo quanto più possibile simile all'originale.

È stato un bene che il mio interesse per l'emulazione fosse principalmente tecnico: se fosse stato sostenuto solo dai giochi per se stessi, probabilmente si sarebbe esaurito una volta replicati i miei preferiti; invece l'interesse risiedeva nella tecnologia, e in essa ha trovato una fonte inesauribile di stimoli. Questo punto di vista ha anche determinato un approccio diverso da quello della maggior parte degli altri emulatori, mettendo l'atto del giocare in secondo piano rispetto alla correttezza dell'emulazione e alla completezza della documentazione.

OBIETTIVI

I videogiochi arcade costituiscono un'importante parte della nostra cultura popolare che rischia di andare perduta a causa dell'obsolescenza dell'hardware. Nell'ottica della conservazione e della trasmissione alle generazioni future di un prodotto della creatività umana, è ampiamente motivata un'opera di recupero, archiviazione e documentazione. Con pertinente analogia, chi si dedica a tale opera è una sorta di archeologo, un "archeologo digitale", per citare l'appropriata definizione, alla quale sono piuttosto affezionato, coniata tempo fa da un giornalista [29].

Il modo più ovvio per preservare i giochi è collezionarne e mantenerne in efficienza le schede originali, nonché i manuali, le decorazioni dei cabinati, gli opuscoli pubblicitari, e tutto ciò che è attinente. Questo viene già fatto da molti, si veda ad esempio <3>, <4> e <5>.

Un'altra procedura molto interessante da seguire è documentare l'hardware utilizzato da ogni gioco, studiando così dall'interno la storia di questo campo dell'informatica.
Quando iniziai ad occuparmi di emulatori, mi resi subito conto che la documentazione esistente sull'hardware dei videogiochi arcade era pressoché nulla. L'obiettivo che mi prefissai iniziando MAME fu quello di colmare questa lacuna, usando l'emulazione come mezzo di ricerca e di verifica delle informazioni raccolte. La documentazione sull'hardware è dunque rappresentata dallo stesso codice sorgente di MAME.

L'obiettivo dichiarato è quello di emulare *tutti* i videogiochi arcade prodotti nella storia. Si tratta di un obiettivo molto ambizioso, che probabilmente non sarà mai raggiunto; per questo motivo MAME è un *work in progress*, un progetto in continuo sviluppo fino a quando ci saranno persone interessate a proseguirlo.

In questi sei anni di intenso lavoro è stato fatto molto: quasi tutti i giochi più importanti sono stati emulati, insieme ad un gran numero di giochi meno conosciuti e addirittura a prototipi esistenti in pochissimi esemplari. A dimostrazione dell'importanza di agire il più in fretta possibile, si osservi che per molti giochi la ricerca di schede ancora funzionanti ha richiesto lungo tempo ed in alcuni casi è tuttora in corso.
Allo stato attuale, il numero totale di giochi emulati ha superato quota 2000: questo significa che in media è stato aggiunto circa un gioco al giorno!

MAME è qualcosa di unico nel suo genere. Ci sono molti altri emulatori di videogiochi, ma nessuno di essi è paragonabile a MAME per la quantità di giochi supportati, per l'attenzione posta nella correttezza dell'emulazione, per la continuità e la durata dell'impegno profuso e per il numero di collaboratori, che sono centinaia sparsi in tutto il mondo.

PROBLEMATICHE

Durante la stesura di questa tesi ho incontrato difficoltà per due ordini di motivi.
Il primo riguarda la vastità dell'argomento: la diversificazione dell'hardware in questo campo è elevatissima; oltre a ciò, MAME copre più di 25 anni di storia dei videogiochi, dagli inizi fino ai giorni nostri. Ho scelto quindi di non entrare nei dettagli di alcun hardware specifico; mi sono limitato ad una descrizione generale delle caratteristiche comuni alla maggior parte dei giochi, inserendo ogni tanto qualche aneddoto o cenno storico per inquadrare meglio gli argomenti. L'unica eccezione

riguarda i sistemi di cifratura, ai quali ho dedicato un intero capitolo, poiché rappresentano una delle parti più interessanti del lavoro svolto.

L'altro motivo di difficoltà è causato dal fatto che il reverse engineering associato all'emulazione, così com'è stato applicato allo sviluppo di MAME, non è facile da schematizzare. Più che di una tecnica si tratta di un'arte, fatta di un misto di esperienza e di intuizione, che si apprende e si sviluppa sul campo. In questi anni molte persone ci hanno avvicinato sperando di poter collaborare al progetto, chiedendo una sorta di "manuale d'istruzioni per il programmatore MAME", ma un manuale del genere non può essere realizzato. Le uniche persone che sono riuscite a collaborare sono quelle che hanno capito da sole come farlo.
Senza entrare nei dettagli, mi sono dunque limitato a descrivere alcuni dei metodi che si possono seguire durante il reverse engineering di un videogioco arcade, fornendo alcuni esempi. Il lettore interessato saprà sicuramente utilizzarli come spunti di riflessione.

Struttura della tesi

I capitoli 1 e 2 introducono i concetti di reverse engineering e di emulazione.
Il capitolo 3 elenca le parti di cui è composto un videogioco arcade e mostra come l'hardware si è evoluto nel corso degli anni. Tutte le informazioni riportate in questo capitolo sono state raccolte durante lo sviluppo di MAME.
Il capitolo 4 mostra alcuni esempi di crittografia applicata al mondo dei videogiochi, derivati anch'essi dall'esperienza diretta.
Il capitolo 5 descrive la storia di MAME, la sua struttura ed organizzazione.
Il capitolo 6 propone una rapida carrellata dei principali altri emulatori di videogiochi arcade, passati e presenti.

Convenzioni

I numeri tra [] indicano riferimenti citati in Bibliografia. I numeri tra <> indicano pagine Internet elencate tra i Links.
Le note a piè pagina del tipo "src/drivers/pacman.c" indicano parti del codice sorgente di MAME pertinenti l'argomento trattato.

I grafici sono basati su dati aggiornati alla versione 0.62 di MAME, rilasciata il 13/11/2002.
Tutte le immagini dei giochi sono prese dall'emulazione realizzata da MAME.

Il CD-R allegato contiene, oltre al testo della tesi in formato elettronico, il codice sorgente di MAME 0.62, ed alcuni giochi (*Robby Roto, Gridlee* e *Poly-Play*) che possono essere distribuiti senza fini di lucro.

RINGRAZIAMENTI

Rivolgo il primo ringraziamento a tutti coloro che hanno collaborato a MAME durante questi anni. Ne cito alcuni e mi scuso con tutti quelli che non sono nominati qui.
Aaron Giles, grande professionista al quale mi lega una profonda stima e che ha dato a questo progetto più di chiunque altro <6>. Mirko Buffoni, il cui apporto nei primi mesi di lavoro è stato fondamentale per la crescita futura. Allard Van Der Bas, dalla cui intuizione è scaturito tutto. L'amministratore del sito ufficiale, Santeri Saarimaa. Il primo curatore della versione Macintosh, Brad Oliver. Yasuhiro Ogawa, grande esperto di storia dei videogiochi e punto di riferimento per tutti i collaboratori giapponesi. Bryan McPhail, Ernesto Corvi, Luca Elia, Zsolt Vasvari, Juergen Buchmueller, Tatsuyuki Satoh, Phil Stroffolino, David Haywood <7>, Stephane Humbert, Al Kossow, Tim Lindquist, JoseQ, Mathis Rosenhauer, Bernd Wiebelt, Mike Balfour, Quench, Manuel Abadia, Valerio Verrando, Dan Boris, Andrea Mazzoleni, Keith Wilkins, Marco Cassili, Jarek Parchanski, Mike Coates, Darren Olafson, Paul Leaman, Jarek Burczynski, Chris Hardy, Takahiro Nogi, Frank Palazzolo, Ron Fries, David Graves, Gerardo Oporto, Yochizo, Karl Stenerud, Hiromitsu Shioya, Ian Patterson, Stefan Jokisch, Farfetch'd, Acho A. Tang, Paul Priest, John Butler, Gareth Long, Mike Haaland, Christopher Kirmse, John Hardy IV, Michael Soderstrom, Randy Hoffman, Lawrence Gold, Alan J McCormick, Thierry Lescot, Chackn, Ruben Panossian, Gerald Vanderick, TheGuru, Greg Ember, Razoola, Mario Silva, Jim Hernandez.

Grazie a tutti gli utenti di MAME che, col loro supporto, ci hanno fatto capire che eravamo sulla strada giusta; grazie in particolare a tutti i collaboratori del sito MAME Testers <8> che verificano costantemente l'accuratezza dell'emulazione. Un saluto a tutti i frequentatori del

newsgroup `it.comp.software.emulatori` che mi hanno aiutato a compilare la bibliografia.

Ringrazio mio padre che non ha potuto vedere neanche l'inizio di questo progetto ma che, con la sua mente da ingegnere, lo avrebbe apprezzato a pieno. Abbraccio mia madre che mi ha sopportato con pazienza quando sembrava che i miei studi non finissero più.

Ricordo Andrea Sorbi che ebbi già come insegnante alla scuola media e che mi indirizzò allo studio della matematica. Sono molto grato a Roberto Magari che attraverso le sue lezioni di Algebra mi fece cogliere per la prima volta l'essenza di questa disciplina.
Ringrazio Alfio Andronico che ha appoggiato senza riserve la mia proposta di tesi e ne ha seguito con entusiasmo lo sviluppo. Infine, Maria Piccione per i preziosi consigli che mi ha offerto con cortesia e disponibilità.

E grazie a tutte le persone, e sono tante, che hanno visto in me più di quello che ci vedevo io.

Siena, 24 novembre 2002

1. REVERSE ENGINEERING

DEFINIZIONE

Una buona definizione di reverse engineering si trova in [4]:

Il reverse engineering è il processo d'analisi di un sistema per identificarne le componenti e le correlazioni tra di esse e creare rappresentazioni del sistema in un'altra forma o ad un più alto livello di astrazione.

Un'altra definizione, più descrittiva, si trova in <9>:

Il reverse engineering è il processo generale d'analisi di una tecnologia specificamente per accertare com'è stata disegnata o come funziona. Questo tipo d'indagine impegna le persone in un costruttivo processo di apprendimento sul funzionamento di sistemi e prodotti. Il reverse engineering come metodo non è confinato ad uno scopo in particolare, ma è spesso una parte importante del metodo scientifico e dello sviluppo tecnologico. Il processo di smontare qualcosa e rivelare il modo in cui funziona è spesso un modo efficace per imparare come costruire una tecnologia o apportarvi migliorie.

Attraverso il reverse engineering, un ricercatore raccoglie i dati tecnici necessari per la documentazione del funzionamento di una tecnologia o di un componente di un sistema. Nel reverse engineering a "black box", i sistemi sono osservati senza esaminare la struttura interna, mentre nel reverse engineering a "white box" si ispeziona il funzionamento interno del sistema.

APPLICAZIONI

Si può dire che il reverse engineering sia una pratica "naturale", figlia della curiosità umana, utilizzata in tutti i campi dell'ingegneria. È probabilmente nell'esperienza di ogni genitore l'aver osservato il proprio pargoletto smontare con entusiasmo il giocattolo nuovo per "guardare cosa c'è dentro".

Senza dubbio da quando l'uomo ha iniziato ad utilizzare il suo ingegno per costruire oggetti, ci sono stati altri uomini pronti ad osservarli, usarli, smontarli, per capire come funzionavano e come costruirne di uguali – o di migliori.

Ad esempio, un aneddoto riferisce che l'inventore della fisarmonica, Paolo Soprani, nel 1863 ospitò un austriaco che aveva con sé una misteriosa "scatola sonora"; se la fece regalare, la aprì, ne studiò il funzionamento, e apportando varie modifiche ai meccanismi e alle dimensioni arrivò a creare il capostipite del popolare strumento <10>.

Il reverse engineering ha molteplici applicazioni e svariati metodi di utilizzo. Di seguito ne elenchiamo alcuni.

- Nella sua forma più ovvia, il reverse engineering è semplicemente lo studio di un qualsiasi oggetto per individuarne i principi di funzionamento o di produzione. Per mantenersi competitiva sul mercato, è indispensabile che ogni azienda esamini cosa producono le sue concorrenti in modo da offrire caratteristiche migliori ad un prezzo più basso.

  Proprio per proteggere gli innovatori dalle facili copie, esistono i *brevetti*. Chi inventa un nuovo prodotto o una nuova tecnica può registrarli e garantirsi, per un periodo di tempo limitato, il diritto esclusivo a sfruttarli. Il brevetto rende inutile il reverse engineering, per due motivi: primo, perché la tecnica non potrebbe in ogni caso essere usata da altri senza pagare un congruo compenso al detentore del brevetto; secondo, perché per ottenere un brevetto è necessario fornire una descrizione particolareggiata, accessibile a tutti. Si può dunque dire che i brevetti sono un'istituzionalizzazione dell'importanza dello scambio d'idee al fine di accelerare il

progresso. Per evitare i lunghi e costosi processi di reverse engineering, si garantisce agli inventori l'uso esclusivo delle loro opere, chiedendo loro in cambio di renderle pubbliche invece di tenerle segrete.

- Grazie all'evoluzione delle tecnologie per la digitalizzazione di oggetti tridimensionali e allo sviluppo di nuovi programmi per il trattamento dei relativi dati, negli ultimi anni il reverse engineering ha acquisito grossa importanza come metodologia per la creazione e lo sviluppo di modelli tramite CAD (Computer Aided Design).
  In questo caso quello che si studia non è un oggetto costruito da altre persone, ma un prototipo (eventualmente in scala) del prodotto di cui si vuole creare un modello matematico. L'oggetto può essere qualsiasi cosa, da una scarpa alla carrozzeria di un'automobile. Il prototipo viene creato dal progettista o dal designer, dopodiché la sua forma viene scandita mediante apposite apparecchiature e memorizzata in un computer come una "nube di punti". Una volta acquisiti, i dati vengono elaborati e convertiti in curve e superfici che descrivono l'oggetto all'interno di un programma CAD. Con questo sistema, è possibile passare in tempi molto rapidi dall'idea del progettista ad un modello computerizzato, su cui viene eseguito tutto lo sviluppo, arrivando infine alla produzione in serie. I costi e i tempi sono sensibilmente inferiori a quelli della progettazione tradizionale.
  La stessa tecnica può essere utilizzata anche per ricavare modelli computerizzati di oggetti già esistenti sul mercato, ma prodotti inizialmente senza utilizzare tecniche CAD.
  A titolo d'esempio, <11> e <12> sono siti di due delle tante ditte specializzate in questo campo.

- Anche i programmi di OCR (Optical Character Recognition) si possono pensare come una forma di reverse engineering: in questo caso gli oggetti da studiare sono pagine di testo, dalle quali si ricavano documenti modificabili in un word processor.

- Tra tutti gli oggetti fisici di cui si può fare reverse engineering, particolarmente interessanti sono gli apparecchi elettronici. È noto che i computer PC-compatibili nacquero in seguito al reverse engineering dell'originale prodotto dall'IBM. IBM tentò anche di bloccare la produzione di *cloni* con una causa, ma fallì nel suo intento perché il reverse engineering era stato eseguito nel massimo rispetto di tutte le leggi.
  Fare il reverse engineering di una scheda elettronica non serve solo a copiarla, ma può anche servire per ripararla, in caso di mancanza degli schemi elettrici.
  Con la diffusione dell'elettronica di consumo, un gran numero di appassionati ha potuto dedicarsi al reverse engineering amatoriale. In <13> si trovano link a progetti (non tutti seri!) di reverse engineering di vari apparecchi elettronici di larga diffusione, venduti spesso come giocattoli, ma complessi al loro interno: da LEGO Mindstorms a Tamagotchi a Furby.

- Il reverse engineering del software è un campo che ha acquisito molta importanza negli ultimi anni. I sistemi informativi hanno un bisogno costante di essere mantenuti aggiornati, per le più svariate ragioni: cambiamenti di tecnologia, ampliamento dei servizi offerti, nuove leggi, la moneta unica, l'anno 2000...
  Dato un programma o un sistema esistente, se ne può studiare la struttura e il funzionamento, in modo da migliorare la comprensione, riscrivere la documentazione, aggiungere nuove funzionalità, correggere errori, o convertire il programma in un altro linguaggio di programmazione.
  Comprendere un programma anche relativamente piccolo, però, è un processo complesso che richiede buona conoscenza del linguaggio in cui il programma è scritto, delle librerie di sistema, degli algoritmi utilizzati, dell'ambito di applicazione (matematico, economico ecc.).

- Se si ha a che fare con un programma il cui codice sorgente non è disponibile oppure è andato perduto, si può eseguire la

decompilazione del programma, mediante programmi che consentono di ricavare dal solo eseguibile un sorgente in linguaggio assembly o anche in un linguaggio più ad alto livello (ad esempio C).

- A volte il programma in sé non interessa, ma si vogliono solo comprendere i dati che utilizza. Ad esempio si può voler migrare un database, oppure esaminare i file salvati da un programma di una ditta concorrente in modo da poterli caricare nel proprio.
  Un utile esempio di reverse engineering dei dati, a cui tra l'altro ho collaborato, riguarda una marca di telecomandi universali programmabili. Questi telecomandi memorizzano la configurazione su una EEPROM, che può essere letta e scritta dall'esterno mediante un connettore che si trova all'interno del vano batterie. Interfacciandosi al telecomando con un PC, si è studiato il modo in cui i dati erano organizzati nella EEPROM, ed è stato possibile scrivere un programma per modificare la configurazione direttamente dal PC; non solo, ma dato che la EEPROM può anche contenere piccoli programmi, si è riusciti a modificare anche il funzionamento del telecomando aggiungendo funzionalità non previste dal costruttore. Una descrizione del lavoro svolto si trova in <14>.

2. SIMULATORI ED EMULATORI

DEFINIZIONI ED ESEMPI

Simulazione ed *emulazione* sono due termini molto usati in ambito informatico, ma il cui significato può variare secondo il campo d'utilizzo, tanto da poterli considerare, in alcuni casi, quasi sinonimi.

Quelle che seguono sono le definizioni riportate dal dizionario on line Digita <Web> Garzanti <15>:

Lemma: **emulazione**
Sillabazione/Fonetica: [e-mu-la-zió-ne]
Etimologia: Dal lat. *aemulatio¯ne(m)*, deriv. di *aemu°lus* 'emulo, rivale'
Definizione: *s. f.*
1 l'emulare; l'atteggiamento di chi cerca di raggiungere o superare gli altri, per lo più in un ambito di valori positivi: *spirito d'emulazione,* fervore competitivo
2 (*atto di*) emulazione, (*dir.*) atto emulativo
3 (*inform.*) capacità di un'apparecchiatura hardware o di un prodotto software di funzionare imitando le procedure di una diversa macchina o programma.

Lemma: **simulare**
Sillabazione/Fonetica: [si-mu-là-re]
Etimologia: Dal lat. *simula¯re*, propr. 'fare, rendere simile', deriv. di *simi°lis* 'simile'
Definizione: *v. tr.* [*io sìmulo* ecc.]

1 manifestare sentimenti insinceri: *simulare amicizia, interesse per qualcuno* | mostrare ciò che non si ha, cercare di far credere qualcosa che non è: *simulare la pazzia*
2 (*estens.*) imitare: *simulare il canto degli uccelli*
3 (*scient.*) riprodurre artificialmente le condizioni in cui si svolge un processo o un fenomeno, per studiarne e verificarne gli effetti: *simulare il volo spaziale*.

Come si vede, la definizione di emulazione è molto chiara e specifica, mentre la simulazione copre un ambito più vasto e generico. Le cose sono ulteriormente complicate dal fatto che quest'ultimo termine può essere usato sia nell'accezione n. 3 (riprodurre artificialmente un processo) che in quella n. 2 (imitare). Secondo il tipo di utilizzo, simulatori ed emulatori dello stesso oggetto possono fare cose simili oppure completamente diverse, e, quasi paradossalmente, il simulatore può essere sia molto meno dettagliato che molto più dettagliato del corrispondente emulatore. Vediamo alcuni esempi.

- Uno dei più noti simulatori in campo elettronico è SPICE (Simulation Program for Integrated Circuits Emphasis). Questo programma consente di analizzare il funzionamento di circuiti elettronici, sia analogici che digitali. SPICE utilizza le leggi della fisica per riprodurre nei dettagli le caratteristiche delle varie componenti del circuito, tenendo in considerazione anche la temperatura di funzionamento.
  Se il circuito di cui si esegue la simulazione è di tipo digitale, se ne potrebbe eseguire anche un'emulazione, riproducendo il comportamento delle porte logiche utilizzando l'algebra di Boole anziché le proprietà fisiche dei *chip*.
  In questo caso, simulatore ed emulatore hanno un diverso scopo: il primo è più orientato alle caratteristiche fisiche del circuito, il secondo a quelle logiche e pratiche. Il simulatore riproduce con maggior precisione i dettagli del funzionamento di tutti i componenti del circuito, mentre l'emulatore riproduce solo il

funzionamento logico del circuito nel suo insieme. Il simulatore, essendo più complesso, è più lento dell'emulatore.

- Una delle applicazioni più comuni di simulatori ed emulatori si ha nello sviluppo di software per microprocessori. In questo ambito, di solito si chiama *simulatore* un software che riproduce il funzionamento del microprocessore, consentendo di eseguire il programma attraverso un interprete; in questo modo si può studiare come questo si comporterebbe sul vero processore, esaminando quale sarebbe il contenuto dei registri interni ad ogni passo, eccetera. Con *emulatore*, invece, di solito si indica una combinazione hardware/software che si collega ad una scheda elettronica al posto del microprocessore, e ne riproduce in tutto e per tutto il funzionamento, sempre però consentendo di verificare il flusso del programma come nel caso del simulatore. Mediante l'emulatore è dunque possibile studiare il funzionamento di tutto il sistema che si sta sviluppando, non solo del microprocessore.
  In questo caso simulatore ed emulatore hanno più o meno lo stesso scopo, e la precisione con cui riproducono il funzionamento del microprocessore è la stessa; l'unica differenza è che l'emulatore è in grado di interfacciarsi con altre parti, mentre il simulatore produce solo una riproduzione "virtuale". Inoltre l'emulatore deve funzionare in tempo reale, mentre il simulatore potrebbe anche non essere in grado di farlo.

- Supponiamo di avere un oggetto elettronico, ad esempio una calcolatrice tascabile, di cui si vuole riprodurre il funzionamento su un personal computer. Possiamo procedere in due modi. Il primo è usare la calcolatrice, osservando come reagisce alla pressione dei vari tasti, e scrivere un programma sul PC che si comporta nello stesso modo. Il secondo è creare un modello dell'hardware della calcolatrice (microprocessore, display LCD, eccetera) e scrivere un programma che lo riproduce; al centro di questo programma ci sarà un emulatore del microprocessore che, al contrario dell'esempio precedente, sarà interamente software, e invece di collegarsi

fisicamente con la calcolatrice al posto del suo microprocessore, si interfaccerà con la riproduzione software dell'hardware della calcolatrice. Si può dire che il simulatore riproduce il software della calcolatrice, mentre l'emulatore riproduce l'hardware.
In questo caso simulatore ed emulatore hanno esattamente lo stesso scopo. L'emulatore riproduce con maggior precisione i dettagli del funzionamento della calcolatrice, mentre il simulatore ne riproduce solo il comportamento generale. L'emulatore, che deve interpretare in tempo reale il programma del microprocessore, è più lento del simulatore.

L'ultimo esempio è quello più vicino all'argomento di cui ci occuperemo in questa tesi.

DEFINIZIONE FORMALE

Nella letteratura riguardante gli emulatori manca una trattazione che fornisca una sistemazione rigorosa dei concetti e dei metodi di questo settore. Le definizioni formali di emulatore e simulatore che propongo di seguito consentono di avviare una trattazione del tipo indicato.

Siano $U_1 = (Q_1, \Gamma, \delta_1, q0_1)$ e $U_2 = (Q_2, \Gamma, \delta_2, q0_2)$ due macchine di Turing universali, dove

Q_1, Q_2 insiemi finiti degli stati
Γ insieme finito dei simboli
$\delta_1: Q_1 \times \Gamma \rightarrow \{\{STOP\} \cup Q_1\} \times \Gamma \times \{L, R\}$
$\delta_2: Q_2 \times \Gamma \rightarrow \{\{STOP\} \cup Q_2\} \times \Gamma \times \{L, R\}$ funzioni di transizione
$q0_1 \in Q_1$, $q0_2 \in Q_2$ stati iniziali

Siano inoltre:
Γ^* l'insieme di tutte le sequenze finite di simboli di Γ
Ω l'insieme di tutte le macchine di Turing che usano un insieme di simboli contenuto in Γ

Esistono due funzioni d_1, d_2: $\Omega \to \Gamma^*$ che associano ad ogni $M \in \Omega$ la descrizione di M appropriata per la relativa macchina universale, cioè t.c.
$\forall M \in \Omega, \forall \sigma \in \Gamma^*, U_1(d_1(M) \frown \sigma) = M(\sigma)$
$\forall M \in \Omega, \forall \sigma \in \Gamma^*, U_2(d_2(M) \frown \sigma) = M(\sigma)$

Definizione 1
Si chiama *emulatore per U_2 di M* la descrizione di M per U_2, cioè $d_2(M)$.

Ogni emulatore gode della seguente fondamentale, seppur semplice, proprietà.

Sia $\varepsilon = d_2(U_1)$ l'emulatore per U_2 di U_1. Si ha che

$\forall \sigma \in \Gamma^*, U_2(\varepsilon \frown \sigma) = U_2(d_2(U_1) \frown \sigma) = U_1(\sigma)$

da cui segue che

$\forall \tau \in d_1(\Omega), \forall \sigma \in \Gamma^*, U_2(\varepsilon \frown \tau \frown \sigma) = U_1(\tau \frown \sigma)$

Quindi *l'emulatore per U_2 di U_1 consente di eseguire su U_2, con gli stessi risultati, tutte le descrizioni di macchina valide per U_1.*

L'altra cosa che si può fare è convertire una descrizione valida per U_1 in una descrizione della stessa macchina valida per U_2. Infatti:

$\forall \tau \in d_1(\Omega) \exists s(\tau) \in d_2(\Omega)$ t.c. $\forall \sigma \in \Gamma^*, U_2(s(\tau) \frown \sigma) = U_1(\tau \frown \sigma)$

basta prendere $M \in \Omega$ t.c. $d_1(M) = \tau$ e porre $s(\tau) = d_2(M)$, da cui:

$U_2(s(\tau) \frown \sigma) = U_2(d_2(M) \frown \sigma) = M(\sigma) = U_1(d_1(M) \frown \sigma) = U_1(\tau \frown \sigma)$

Definizione 2
$s(\tau)$ si chiama *simulatore per U_2 di U_1 che esegue τ.*
Osserviamo che $s(\tau)$ è anche l'emulatore per U_2 di M.

La differenza sostanziale tra simulatore ed emulatore è dunque che l'emulatore riproduce integralmente il funzionamento di una macchina (universale e non), mentre il simulatore riproduce il funzionamento di

una macchina universale solo mentre questa sta emulando un'altra specifica macchina.

Emulazione e Reverse Engineering

Precedentemente abbiamo accennato agli emulatori di microprocessore. In quel campo non solo è possibile avere una riproduzione pressoché perfetta dell'originale, ma si può anche pensare di generare automaticamente gli emulatori partendo da una definizione formale del linguaggio del microprocessore.

Questa situazione ideale, però, non sempre si presenta. Spesso il sistema che si vuole emulare non è ben documentato, ed è quindi necessario eseguire un lavoro di reverse engineering. La creazione dell'emulatore può dunque essere fatta in molti modi diversi, alcuni più "formali", altri più "pratici", in funzione della quantità di informazioni disponibili sul sistema da emulare.
Vediamo alcuni esempi, tutti riguardanti la riproduzione su PC di videogiochi.

1. È possibile tentare di capire il funzionamento di un gioco senza fare altro che utilizzarlo a lungo; poi si può scrivere un programma che gli assomigli il più possibile. In questo caso si può parlare di simulatore solo nel senso di "imitatore", perché senza accesso al programma e ai dati utilizzati dall'originale, per quanto accurato possa essere il suo studio, la riproduzione difficilmente potrà essere completamente fedele. Questo è vero in particolar modo per i giochi, che sono ricchi di grafica e di particolari difficili da replicare: si pensi, ad esempio, agli algoritmi di intelligenza artificiale necessari per bilanciare la difficoltà del gioco.
   All'atto pratico, una simulazione anche solo approssimativa è realisticamente possibile solo per giochi con poca grafica e strutture di gioco molto semplici.

Risultati molto interessanti sono stati raggiunti da Luca Antignano, in una serie di simulatori di giochi portatili con display LCD <16>.
Il vantaggio di questo tipo di simulatore è che non richiede nessuna conoscenza dell'hardware e del software del gioco: l'unica cosa necessaria è la possibilità di osservare il gioco in funzione, quindi può essere scritto da chiunque. Date le sue limitate possibilità, però, è in generale una strada da utilizzare solo se non ci sono alternative, cioè quando il software della macchina originale è inaccessibile o inutilizzabile.

2. Se si ha a disposizione il codice sorgente del gioco originale, si possono utilizzare le tecniche di reverse engineering del software per trasportarlo su un'altra architettura hardware, eventualmente anche cambiando linguaggio di programmazione e/o riscrivendo alcune parti dopo averne formalizzato le specifiche. Ovviamente per eseguire la conversione è necessario avere anche un modello dell'hardware originale, presumibilmente derivabile dai commenti del codice sorgente o dalla documentazione ad esso associata.
   Con questo tipo di simulatore, gli algoritmi del gioco possono essere preservati e la riproduzione, in teoria, potrebbe anche essere perfetta. In pratica, però, durante la scrittura di un simulatore è facile prendere scorciatoie, ed abbandonare la fedeltà assoluta all'originale per motivi di efficienza o di comodità. Può capitare anche di fare intenzionalmente dei cambiamenti per adattare il gioco all'architettura su cui viene trasportato, ad esempio variandone il livello di difficoltà, oppure aggiungendo la scritta **PRESS F2 T0 PLAY** invece di lasciar inserire monete "virtuali", come è successo nel pacchetto *Microsoft Return of Arcade*. Si tratta di piccoli cambiamenti, che però fanno storcere il naso ai puristi del gioco arcade.

   Dato che è necessario avere accesso al codice sorgente e ai dati originali, possibilità che in pratica si ha soltanto nell'ambito di una licenza autorizzata, questo tipo di simulatori è usato principalmente per la produzione di videogiochi commerciali. In quell'ambito, la

simulazione è senza dubbio da preferire all'emulazione, poiché richiede molta meno potenza di calcolo per funzionare a piena velocità.

***Pac-Man* secondo Microsoft (simulazione)**

***Pac-Man* secondo MAME (emulazione)**

Negli anni '80 erano molto popolari le conversioni di giochi da bar per home computer come il Commodore 64 o il Sinclair Spectrum. Date le limitate capacità di queste macchine, le conversioni erano usualmente molto diverse dagli originali, tanto che a volte l'unica relazione era il nome del gioco. In quei casi, le conversioni erano imitazioni che sfruttavano ben poco materiale del gioco originale.
Attualmente le conversioni sono forse meno frequenti, ma quelle che vengono fatte sono di solito di buona qualità e molto simili all'originale; spesso danno la possibilità di giocare sia una versione fedele a quella arcade che una modificata per essere più adatta all'uso casalingo.

3. Se il codice sorgente del gioco originale non è disponibile, si può ugualmente cercare di capirne il funzionamento usando la decompilazione. L'operazione può essere eseguita su tutto il programma, per creare un simulatore, oppure solo su alcune parti,

durante la creazione di un emulatore. Questa pratica è sottoposta a restrizioni dalle leggi sul diritto d'autore.

4. La maggior parte dei giochi utilizza una o più CPU reperibili sul mercato e quindi ampiamente documentate. Per prima cosa va scritto un emulatore di queste CPU, basandosi sulla documentazione disponibile. Si carica poi il programma originale sull'emulatore e si prende nota degli accessi alla memoria e alle porte di I/O. In una gran quantità di casi, queste informazioni sono sufficienti per intuire con ottima approssimazione il funzionamento dell'hardware. Gli emulatori di CPU sono componenti standard, che una volta scritti si possono utilizzare più volte per emulare giochi diversi. Per questo motivo, chi si occupa dell'emulazione di un nuovo hardware non ha neanche necessità di conoscere il linguaggio *assembly* della CPU usata: è sufficiente leggere l'elenco degli accessi alla memoria e alle porte di I/O.
   Questa tecnica è la più usata nello sviluppo di MAME, per vari motivi, ad esempio perché non richiede l'accesso diretto all'hardware da emulare e perché non può violare le leggi sul copyright: **il reverse engineering dell'hardware viene fatto interamente attraverso il solo studio del *comportamento* del software**.
   Lo svantaggio è che spesso il modello dell'hardware che si ricava dallo studio del solo software è incompleto. Ad esempio, spesso il software non utilizza tutte le potenzialità dell'hardware; in tal caso, se esistono più giochi che girano sulla stessa scheda, dal loro confronto di solito si riesce a migliorare la comprensione dell'hardware.
   Dal solo comportamento del software non sempre è facile dedurre quale debba essere il comportamento dell'hardware; se dopo un po' di tentativi non si riesce a costruire un modello dell'hardware coerente col software, non resta altro da fare che rinunciare e riprovare in un altro momento, aspettando nuove ispirazioni. La

dipendenza di questo metodo dall'intuito e dall'esperienza di chi esegue il reverse engineering ne rappresenta il limite principale.

5. Una strategia alternativa per il reverse engineering dell'hardware è, invece di affidarsi solo al programma originale, scrivere nuovi programmi da far girare sulla scheda originale per verificare gli effetti degli accessi a certe locazioni di memoria o porte di I/O. Questa tecnica si può usare come completamento di quella descritta nell'esempio 4, per risolvere le ambiguità o raccogliere maggiori informazioni. Ovviamente lo svantaggio è che, al contrario dell'esempio precedente, è necessario avere accesso diretto all'hardware da emulare.

6. Se la piattaforma che si vuole emulare ha un manuale per il programmatore, lo si può usare per creare un modello dell'hardware e riprodurlo fedelmente. Questa tecnica è subordinata prima di tutto all'esistenza di tale manuale, cosa tutt'altro che scontata per i videogiochi arcade, ed anche alla sua reperibilità, perché anche se il manuale esiste, non è detto che sia facile averlo. L'emulazione degli home computer è probabilmente quella che meglio si presta all'uso di questo metodo.
   Una cosa da tener presente è che il manuale del programmatore potrebbe essere incompleto, omettendo di segnalare caratteristiche nascoste o anche veri e propri *bug* dell'hardware. Se ciò accade, la corretta riproduzione di tali caratteristiche può avvenire solo a posteriori, attraverso l'esame degli eventuali programmi che non funzionano correttamente sull'emulatore.

7. Esaminando gli schemi elettrici, se sono disponibili, si può ricavare un modello del funzionamento della scheda da emulare. Questo metodo consente di ottenere emulazioni molto fedeli dell'originale, ma i risultati migliori si ottengono solo con i giochi precedenti alla prima metà degli anni '80. Infatti, il grado di integrazione dei circuiti elettronici è andato via via aumentando, cosicché, studiando gli schemi elettrici di schede più recenti, ci si trova di fronte a

circuiti integrati di cui si può solo intuire il funzionamento, considerandoli dei "black box". In tal caso, è necessario affiancare a questa tecnica una o più delle altre descritte in precedenza.
Se gli schemi elettrici non sono disponibili, si può comunque tentare di usare questo metodo, esaminando fisicamente la scheda del gioco, in modo analogo a quanto accennato nell'esempio a pag. 16.

Riassumiamo in una tabella le caratteristiche degli esempi proposti. Ricordiamo che il reverse engineering a "black box" non esamina la struttura interna del sistema studiato, mentre quello a "white box" sì.

Esempio	Tipo di riproduzione	Tipo di reverse engineering	
		Software	Hardware
1	Imitazione	Black Box	N/A
2	Simulazione	White Box	Black Box
3	Simulazione/Emulazione	White Box	Black Box
4	Emulazione	Black Box	Black Box
5	Emulazione	N/A	Black Box
6	Emulazione	N/A	White Box
7	Emulazione	N/A	White Box

QUANTO È ACCURATO UN EMULATORE?

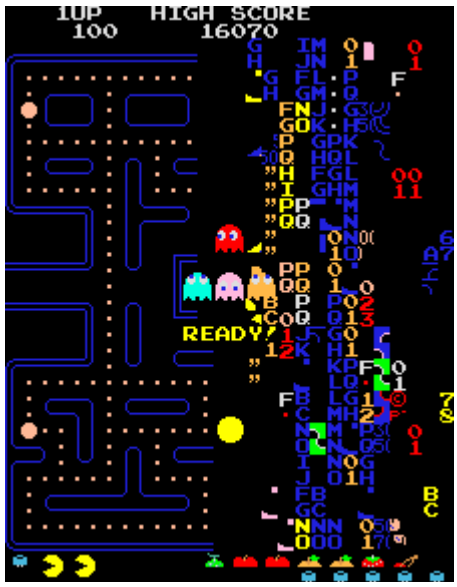
In teoria, come abbiamo visto dalla definizione formale, simulatori ed emulatori dovrebbero essere equivalenti. I simulatori hanno il vantaggio di richiedere meno potenza di calcolo, perché c'è un passaggio interpretativo in meno, mentre gli emulatori hanno il vantaggio di far funzionare *qualsiasi* programma scritto per il sistema emulato, anziché uno solo; a parte questo, i risultati dovrebbero essere entrambi indistinguibili dall'originale.

In pratica, non sempre le cose vanno così. Gli esempi precedenti mostrano come la creazione di questi programmi sia soggetta ad una

serie di errori, sia nella fase di comprensione del sistema da emulare, che durante la scrittura del programma.

Dati i differenti approcci, è facile che simulatori ed emulatori si comportino in modo diverso. Gli emulatori, lavorando a livello dell'hardware, possono riprodurre più facilmente i dettagli della scheda originale; i simulatori invece, lavorando a livello del software, hanno la tendenza a riprodurre solo il comportamento generale.
L'emulazione dei videogiochi è particolarmente delicata a causa della loro interattività e del loro funzionamento in tempo reale, per cui anche piccoli cambiamenti di velocità possono alterare il modo in cui il gioco si comporta. Per questo motivo, si è diffusa nella comunità dei giocatori la convinzione che gli emulatori sono riproduzioni fedeli degli originali, mentre i simulatori no.

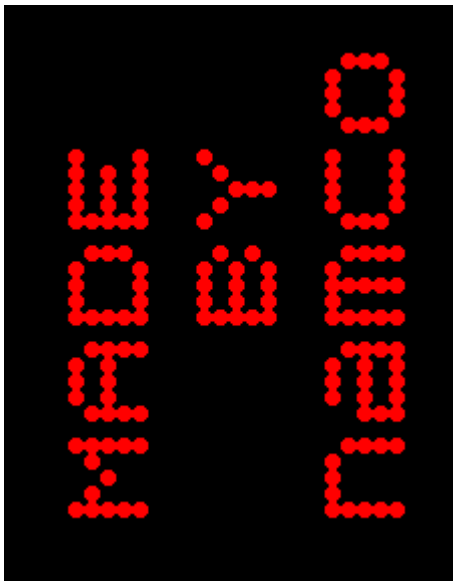
In generale, questo è vero. Come detto nell'esempio 4 a pag. 26, la maggior parte dei giochi utilizza microprocessori standard, che sono ampiamente documentati ed emulabili accuratamente. Questo significa che la *logica* del gioco può essere riprodotta con grande precisione. Altre parti dell'hardware sono soggette ad un più alto tasso d'errore, per cui la riproduzione di grafica e sonoro può essere imprecisa, anche in modo evidente; ma tutti i meccanismi interni del gioco, dal movimento dei nemici all'attribuzione dei punti, sarà replicato fedelmente, fino a comprendere anche gli eventuali *bug* del gioco originale, che in un simulatore possono facilmente andare perduti.
Ad esempio, *Pac-Man* ha un noto *bug* dovuto al fatto che il livello raggiunto è memorizzato in una variabile a 8 bit. Quando si raggiunge il livello 256, si supera la capacità della variabile, causando conseguenze inattese, come si può vedere nell'immagine a lato. In



Il *bug* del 256° livello di *Pac-Man* emulato da MAME

un simulatore, probabilmente la capacità assegnata alla variabile sarebbe diversa, eliminando il comportamento anomalo. In un emulatore, invece, il programma viene eseguito esattamente come nell'originale.

Un altro dettaglio che in un simulatore può facilmente essere trascurato sono le cosiddette *Easter egg*, cioè messaggi o altre sorprese nascoste nel codice dai programmatori che possono essere rivelate eseguendo particolari azioni, come premere una sequenza di tasti in un determinato ordine.
Le foto seguenti mostrano due esempi per i giochi *Pac-Man*[2] e *Xevious*[3], entrambi emulati da MAME.



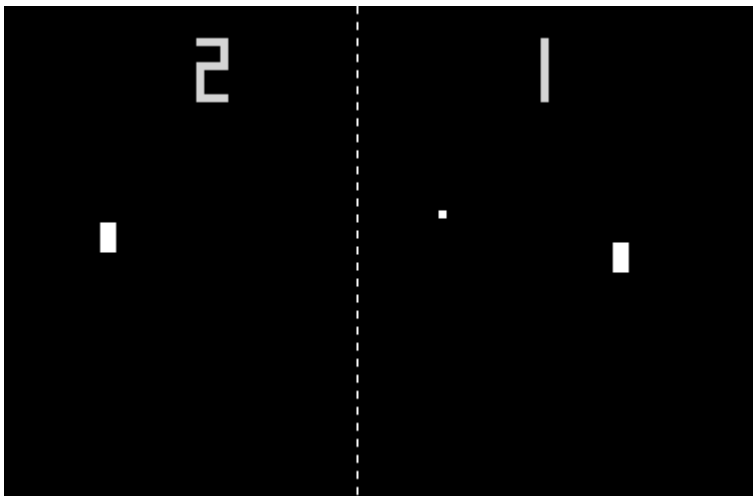
Easter egg* di *Pac-Man
(Namco, 1980)

Easter egg* di *Xevious
(Namco, 1982)

[2] Mettere il gioco in *service mode*; rapidamente mettere l'interruttore del *service mode* su off e poi di nuovo on, in modo da passare alla griglia per il test del monitor; tenere premuti contemporaneamente i pulsanti di inizio gioco per uno e due giocatori e fare di nuovo un rapido off/on con l'interruttore del *service mode*; muovere il joystick 4 volte in alto, 4 volte a sinistra, 4 volte a destra e 4 volte in basso.

[3] Iniziare una partita normalmente, portarsi immediatamente in basso a destra e tenere premuto il pulsante delle bombe. Dopo qualche secondo appare la scritta "Namco original program by EVEZOO".

Esistono anche videogiochi più vecchi, risalenti agli anni '70, che non sono dotati di CPU, ma che eseguono un semplice programma realizzato mediante porte logiche. Formalmente, le loro schede non sono dunque equivalenti a macchine di Turing universali, e quindi, secondo le definizioni che abbiamo dato in precedenza, si può parlare solo di emulatori e non di simulatori, dato che questi ultimi sono definiti solo per le macchine di Turing universali. Ovviamente si può continuare a parlare di simulatori nel senso lato di "imitatori" che anzi, vista la semplicità di questi giochi, sono anche piuttosto facili da programmare.



***Pong*: emulazione o simulazione?**

Si è tentato di estendere MAME al supporto di giochi privi di CPU, ma già nel caso di uno dei più elementari, il celebre *Pong* (Atari, 1972), ci si è resi conto della difficoltà di stabilire se l'emulazione fosse realmente accurata. Mancando la CPU, mancavano garanzie sul comportamento del programma, e ogni minimo errore nell'interpretazione degli schemi elettrici avrebbe potuto alterare il funzionamento del gioco. Per questo motivo, abbiamo deciso di escludere dagli obiettivi di MAME l'emulazione di questo tipo di videogiochi, concentrandoci solo su quelli dotati di CPU.

3. ARCHITETTURA DEI VIDEOGIOCHI ARCADE

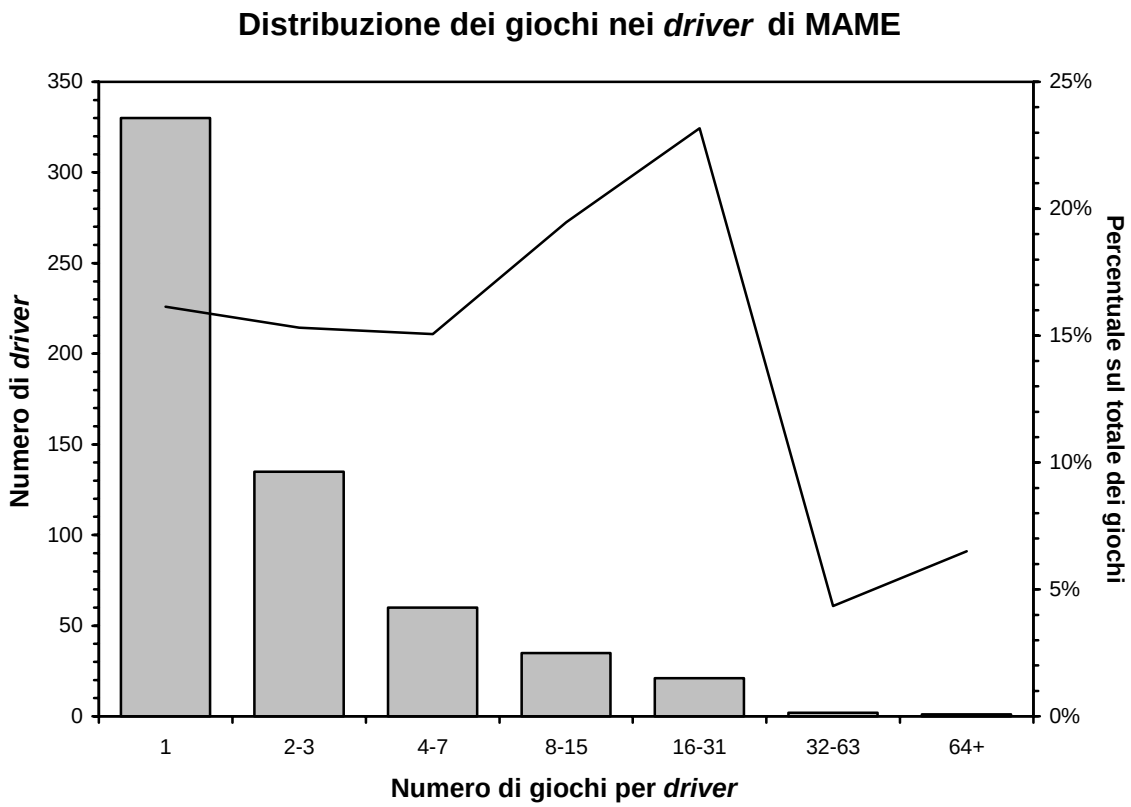
INTRODUZIONE

Nel capitolo precedente abbiamo visto alcuni esempi di reverse engineering applicato all'emulazione di videogiochi. In questo capitolo esamineremo più in dettaglio di quali parti è composta una macchina da gioco arcade, elencando anche alcune delle problematiche connesse all'emulazione dei vari sottosistemi.

Una delle differenze principali tra un emulatore di videogiochi arcade e un emulatore di console è la quantità di software che i due devono far girare. Una console rimane sul mercato per alcuni anni, durante i quali vengono prodotti molti giochi. Una scheda per giochi arcade, invece, solitamente ha una vita più breve, e può far girare pochi giochi, spesso soltanto uno. Ne consegue che un emulatore come MAME, che supporta un gran numero di giochi, deve gestire tutte le architetture hardware sulle quali quei giochi girano. In MAME, i moduli software utilizzati per questo scopo sono chiamati *driver*. Ogni *driver* emula l'hardware di una scheda, o di più schede molto simili tra loro, e contiene le informazioni necessarie per far girare i giochi prodotti per quel particolare hardware.

Come si può vedere nel grafico seguente, l'hardware è così vario che la maggior parte dei *driver* gestisce un solo gioco; circa il 16% dei giochi supportati richiedono un *driver* dedicato. Un solo *driver* riconosce più di 100 giochi: non a caso, riguarda il Neo-Geo[4], un sistema a cartucce venduto sia per l'utenza domestica che per quella arcade.

[4] src/drivers/neogeo.c



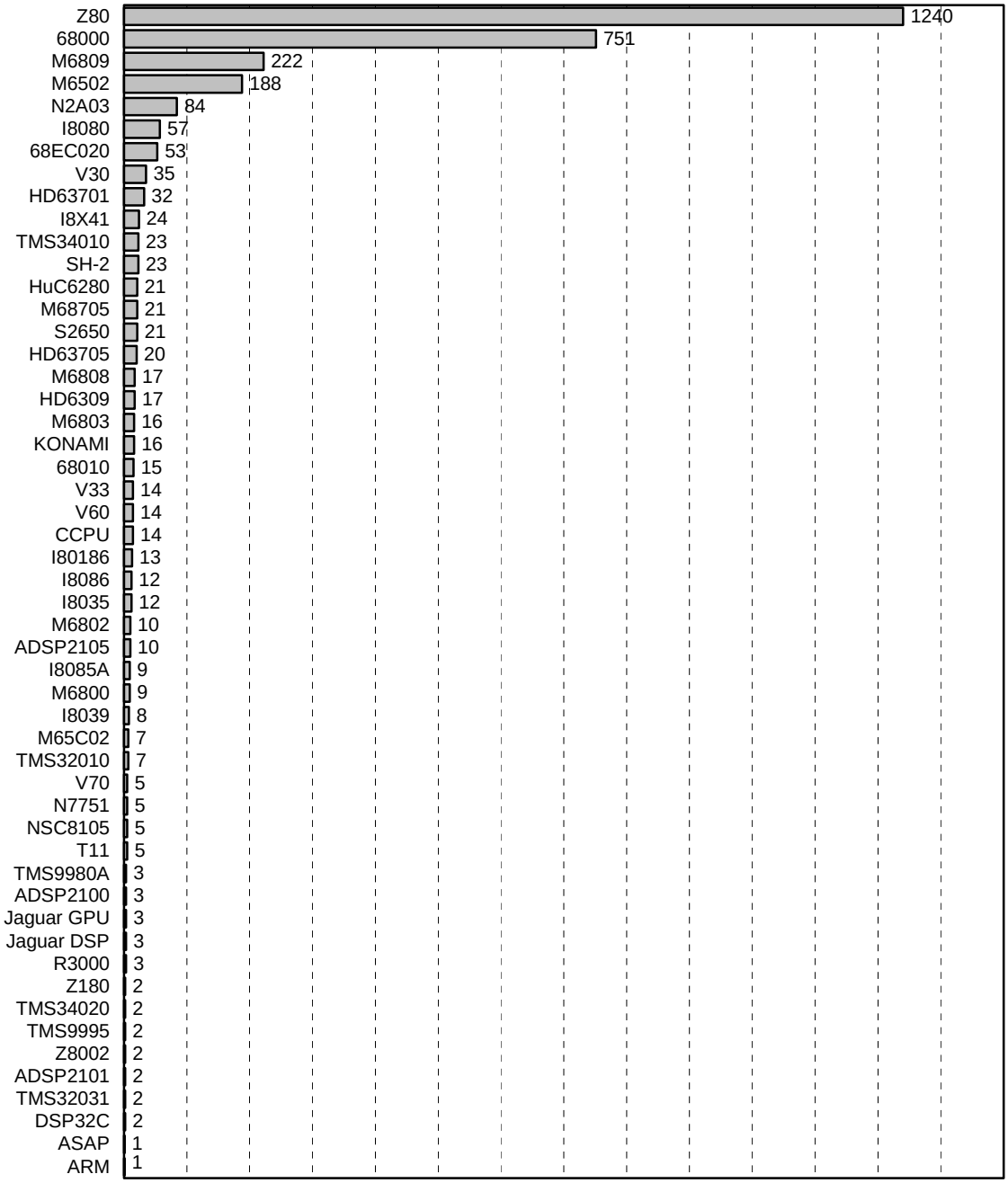
CPU

La CPU (*Central Processing Unit*) è il cuore di ogni computer, l'elemento che dirige il funzionamento di tutto il sistema; i videogiochi non fanno eccezione. Per quanto concerne le prestazioni, però, non è la parte più importante: questo ruolo, come vedremo più avanti, spetta all'hardware grafico.

Nella maggior parte dei casi, le schede dei videogiochi sono multiprocessore. Quasi sempre una CPU è dedicata esclusivamente al controllo del sonoro; a volte sono presenti anche altre CPU che si dividono la gestione del gioco, nel caso questo sia troppo complesso per essere governato da una sola CPU. Ad esempio una CPU secondaria potrebbe essere utilizzata per fare i controlli di collisione tra tutti gli oggetti presenti sullo schermo, oppure per eseguire una conversione di coordinate da un formato interno a quello riconosciuto dall'hardware grafico.

MAME emula una cinquantina di microprocessori[5], la maggior parte dei quali utilizzata da un numero molto ridotto di giochi.

CPU emulate da MAME e numero di giochi che le utilizzano



Tra i microprocessori a 8 bit, il più diffuso è stato senza dubbio lo Zilog Z80, utilizzato nella maggior parte dei giochi fin quasi alla fine degli anni ‘80. Quando 8 bit non bastarono più, avvenne il passaggio di consegne al

[5] src/cpuintrf.h, src/cpu

Motorola 68000, microprocessore a 16/32 bit di grande successo. Lo Z80 ha comunque continuato ad essere largamente utilizzato come CPU secondaria.

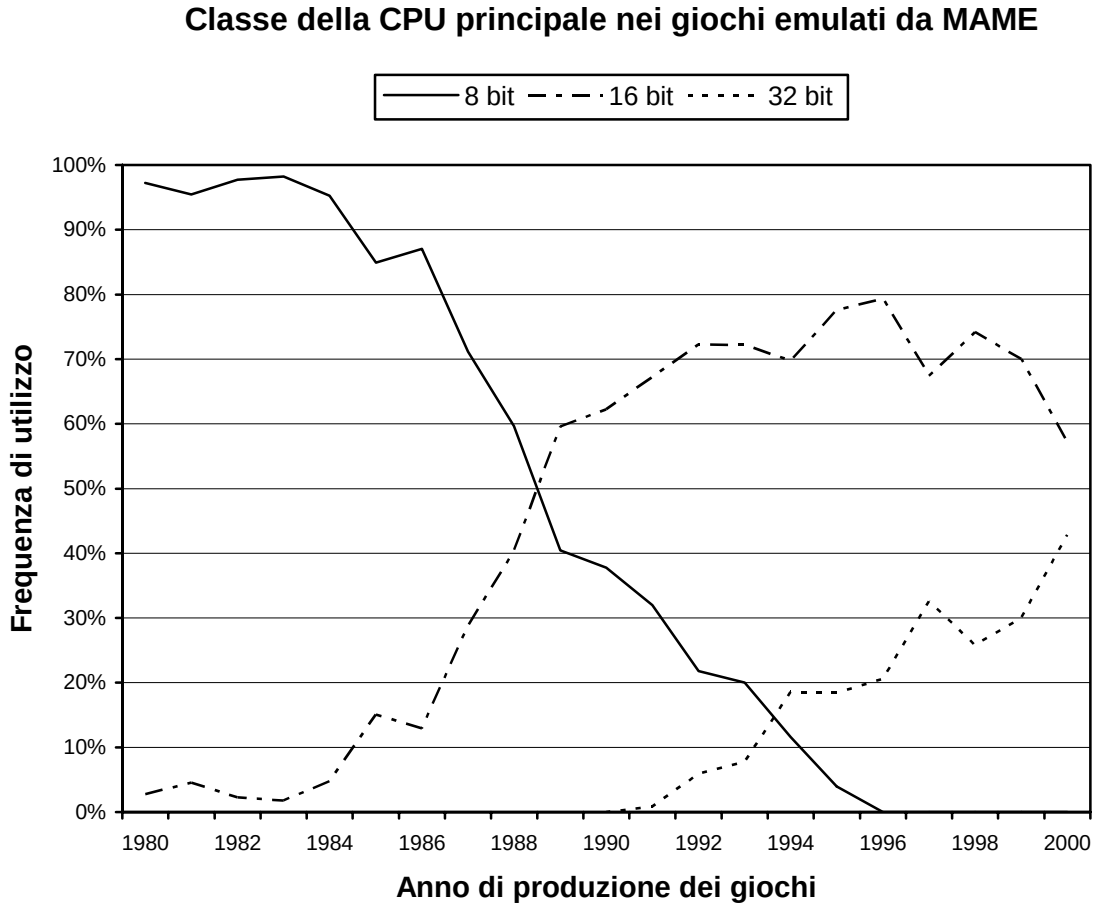
Se tra i microprocessori a 8 bit esistevano alternative valide allo Z80, come il Motorola 6809 e il Rockwell 6502, tra quelli a 16 bit il 68000 si è rivelato dominatore incontrastato.

Molti dei microprocessori utilizzati per i videogiochi degli anni '80 sono gli stessi che si trovavano negli home computer dell'epoca: Z80 nel Sinclair ZX Spectrum, 6502 nell'Apple II e nel Commodore 64, 68000 nell'Apple Macintosh e nel Commodore Amiga. Degno di nota il fatto che l'Intel 8086, che si sarebbe imposto come standard *de facto* nei PC, sia stato quasi completamente ignorato nella progettazione di videogiochi.

Una menzione speciale va fatta alla cosiddetta CPU Cinematronics, così chiamata perché usata dall'omonima ditta per i suoi giochi in grafica vettoriale del periodo 1978-1981. La peculiarità di questa CPU è che non si tratta di un microprocessore, cioè di un singolo *chip*, ma di un intero circuito realizzato mediante porte logiche. Il circuito è stato accuratamente studiato e documentato da Zonn Moore [8]. L'emulatore di questa CPU scritto da Moore è alla base di quello contenuto in MAME[6].

Il grafico seguente mostra com'è cambiata nel corso degli anni la classe della CPU principale. I dati sono limitati ai giochi emulati da MAME, quindi il grafico non dà un'immagine completa della situazione, specialmente per gli anni più recenti, dei quali MAME supporta solo un numero limitato di giochi (vedere il grafico a pag. 79 per il numero di giochi emulati anno per anno). Esso fornisce comunque un'idea di massima dell'andamento del mercato.

[6] src/cpu/ccpu/ccpu.c



Come detto nel capitolo precedente, l'emulazione della CPU è ciò che consente agli emulatori di essere più accurati dei simulatori nel riprodurre tutte le caratteristiche dell'originale. Questi sono i motivi principali:

- I microprocessori sono generalmente componenti standard, prodotti in gran quantità e ampiamente documentati, per cui su di essi non è necessario compiere un lavoro di reverse engineering.

- L'emulatore di un microprocessore è molto sensibile ai *bug*: un errore nella riproduzione di qualche istruzione di solito fa sì che il programma che gira sull'emulatore abbia dei malfunzionamenti evidenti, portando quindi rapidamente all'identificazione e alla correzione del *bug*.

- Spesso lo stesso microprocessore è usato da molti giochi diversi, quindi si può verificare l'accuratezza dell'emulatore con un buon numero di programmi.

In generale, dunque, l'emulazione della CPU non presenta particolari difficoltà teoriche: si tratta solo di realizzare un'implementazione software delle caratteristiche dichiarate dal costruttore. Ovviamente possono esserci alcune difficoltà pratiche che crescono insieme alla complessità dei microprocessori da emulare. Inoltre, poiché l'emulatore deve funzionare in tempo reale, la velocità del computer su cui viene eseguito pone dei limiti alla velocità di clock che può avere il microprocessore emulato.
Ci sono dei casi in cui invece le difficoltà sono di tipo teorico, e non sempre facilmente superabili. Vediamone alcune.

Caratteristiche non documentate

Molti microprocessori sono in grado di eseguire istruzioni che non sono menzionate nella documentazione ufficiale, oppure sono indicate come "non supportate". Inoltre, le istruzioni possono avere effetti collaterali non documentati, ad esempio sul registro di stato. In alcuni casi queste caratteristiche cambiano secondo la revisione del microprocessore, quindi diventa ancora più complicato riprodurle accuratamente. Si vedano, ad esempio, <17>, <18> e <19> per informazioni sulle particolarità di alcuni microprocessori molto usati.

Nell'emulazione degli home computer è praticamente indispensabile duplicare fedelmente tutte le caratteristiche non documentate della CPU per avere una buona compatibilità col software esistente. Nei videogiochi arcade l'uso di tali funzioni è invece rarissimo, ma ci sono alcuni casi noti. Ad esempio in *King of Fighters '98* il contatore del tempo di gioco rimasto non si azzera se l'istruzione SBCD non imposta correttamente il flag di stato N, che

è indicato come "non definito" nella documentazione ufficiale del Motorola 68000[7].

Crittografia

La cifratura del programma è uno dei più comuni sistemi di protezione dalla copia. Ne parleremo più approfonditamente nel capitolo 4.

Programma inaccessibile

A volte può capitare di essere in grado di emulare un microprocessore, ma... di non avere il programma da farci girare sopra. Normalmente le istruzioni in linguaggio macchina si trovano in alcuni *chip* di memoria ROM che sono facilmente leggibili con l'apparecchiatura adeguata, ma non sempre questo è possibile. Ad esempio il *chip* potrebbe essere stato reso fisicamente inaccessibile inserendolo insieme alla CPU in un blocco di resina epossidica, materiale molto difficile da penetrare senza distruggere il contenuto <20>. Più frequentemente, la ROM può trovarsi all'interno di una MCU (*MicroController Unit*), componente che di solito può essere configurato in modo da proibire la lettura del contenuto.
Le opzioni possibili in questi casi sono essenzialmente tre:

1. In alcuni casi si può trovare il modo di leggere ugualmente i dati protetti.
   In <21>, Clayton Cowgill illustra, con l'aiuto di ottime foto, com'è riuscito a penetrare la resina epossidica di *Pac-Man Plus*.
   Nel caso delle MCU, alcuni modelli possono essere letti sfruttando debolezze nei loro sistemi di sicurezza. I lavori [10] e <22> spiegano alcuni di questi procedimenti (non riguardanti MAME).

2. Se è protetta solo una parte del programma, mentre il resto è normalmente accessibile, si può modificare la parte accessibile sostituendola con un *trojan horse*. Il *trojan horse* ha accesso ai

[7] Cfr. [9], pag. 606.

dati protetti, e li può quindi copiare da qualche altra parte, dove possano essere letti più facilmente.
Questa tecnica è stata usata con alcuni giochi della Orca (*Changes, Funky Bee*, *Marine Boy* e *Springer*) dove i soli primi 256 byte del programma erano contenuti all'interno di un modulo chiamato "CPU Pack II".
Un altro caso in cui questa tecnica è stata fruttuosa riguarda una serie di giochi della Namco, tra cui *Pac-Land*, *Sky Kid* e i sistemi Namco System 1 e Namco System 2. Tutti questi giochi utilizzano una MCU Hitachi HD63701, ma il programma è diviso in due parti, la prima contenuta nella ROM interna della MCU, l'altra contenuta in una ROM esterna in cui è stato possibile inserire il *trojan horse*.

3. Se non si riesce in alcun modo ad accedere al programma protetto, l'unica alternativa che rimane è la simulazione. Frequentemente, le MCU con programma protetto sono utilizzate solo per controlli antipirateria: la CPU principale e la MCU si scambiano dati attraverso una porta di comunicazione, e i valori ritornati dalla MCU sono utilizzati per controllare il flusso del programma (ad esempio tabelle di salto), oppure contengono dati non deducibili dal solo programma principale (ad esempio le coordinate di oggetti mostrati durante il gioco). Si può dunque eseguire un reverse engineering di tipo *black box* sulla MCU: mediante un programma fatto girare sulla scheda originale, si inviano tutte le possibili sequenze di input e si registrano le risposte. Si è riusciti così ad emulare, per esempio, alcuni giochi della Data East, come *Heavy Barrel*[8], *Bad Dudes*, *Chelnov*[9] e *Wonder Planet*.
   Questo metodo è stato utilizzato anche per il reverse engineering di altri dispositivi diversi dalle MCU, ad esempio

[8] src/machine/dec0.c

[9] src/drivers/karnov.c

Aaron Giles e Frank Palazzolo lo hanno applicato durante lo studio dello *slapstic*[10], un *chip* di protezione della Atari.

Microprocessori sconosciuti

In alcune rare occasioni, la CPU da emulare può essere di un tipo sconosciuto, di cui non si riesce a trovare documentazione. Se le ricerche non danno frutto, non resta altra possibilità che tentare un'operazione di reverse engineering, esaminando il codice in linguaggio macchina e cercando di intuirne le funzioni. In MAME, una cosa del genere è stata fatta per un microprocessore *custom* della Konami[11] usato in diversi giochi nel periodo 1987-1991, ad esempio *Ajax* e *The Simpsons*. Ci sono tre versioni del *chip*, funzionalmente equivalenti ed interscambiabili, indicate con le sigle 052001, 052526 e 053248. La CPU è derivata dal 6809, ma molto diversa e con un gran numero di istruzioni in più, anche complesse come la divisione. Il reverse engineering, realizzato per la maggior parte da Ernesto Corvi e Manuel Abadia, richiese alcune settimane di lavoro, ma fu completato con ottimi risultati, consentendo di emulare tutti i giochi che usano quella CPU.

ROM

Il dispositivo d'elezione per la memorizzazione di programmi e dati in un videogioco arcade sono i *chip* di memoria ROM (*Read Only Memory*), che possono essere dei tre tipi descritti di seguito [11].

Masked ROM. Il contenuto è fissato prima della produzione, e non può più essere cambiato. Hanno il vantaggio di essere poco costosi, ma solo se prodotti in grandi quantità.

PROM (Programmable ROM). Alla produzione sono vuoti e possono essere programmati una sola volta.

[10] src/machine/slapstic.c

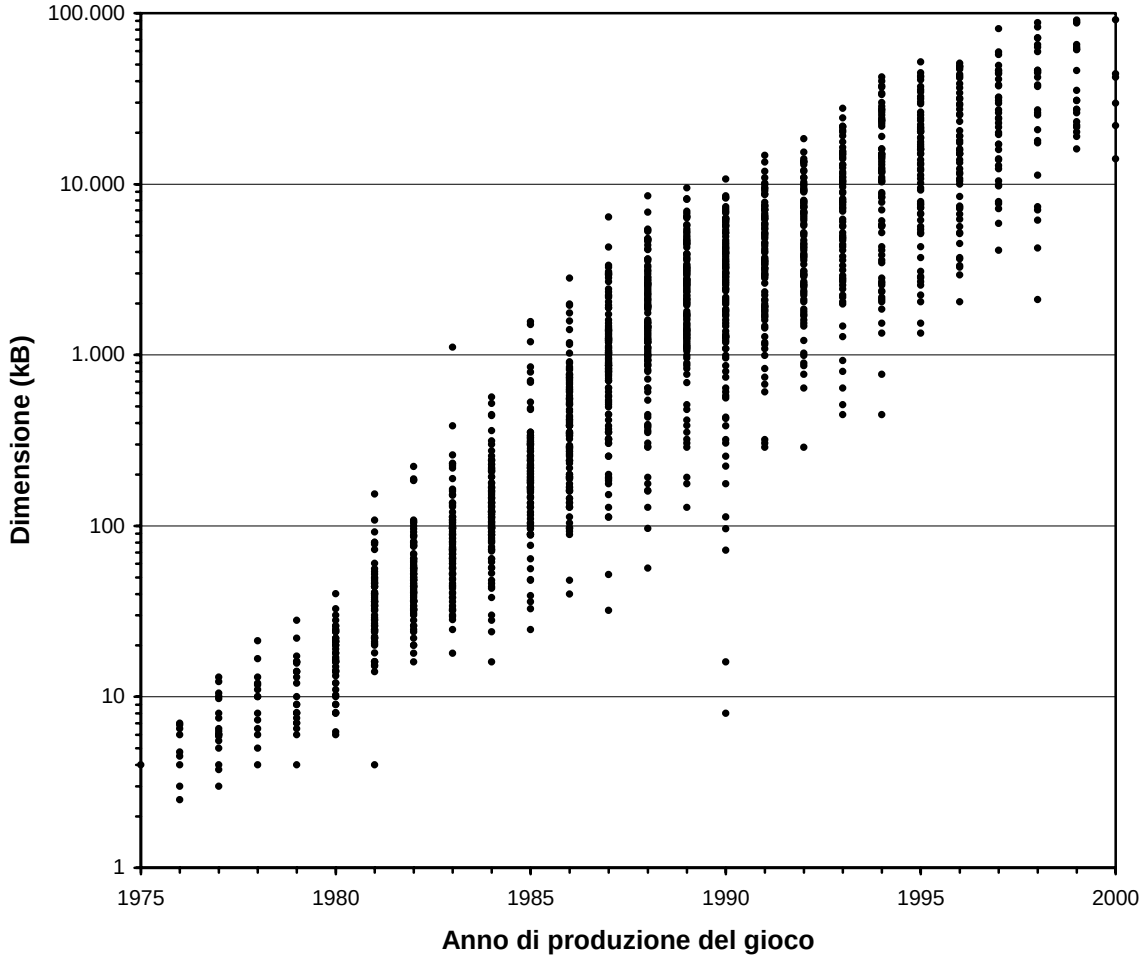
[11] src/cpu/konami/konami.c

EPROM (Erasable Programmable ROM). Programmabili come le PROM, possono inoltre essere riprogrammati più volte, dopo averli cancellati mediante esposizione a luce ultravioletta.

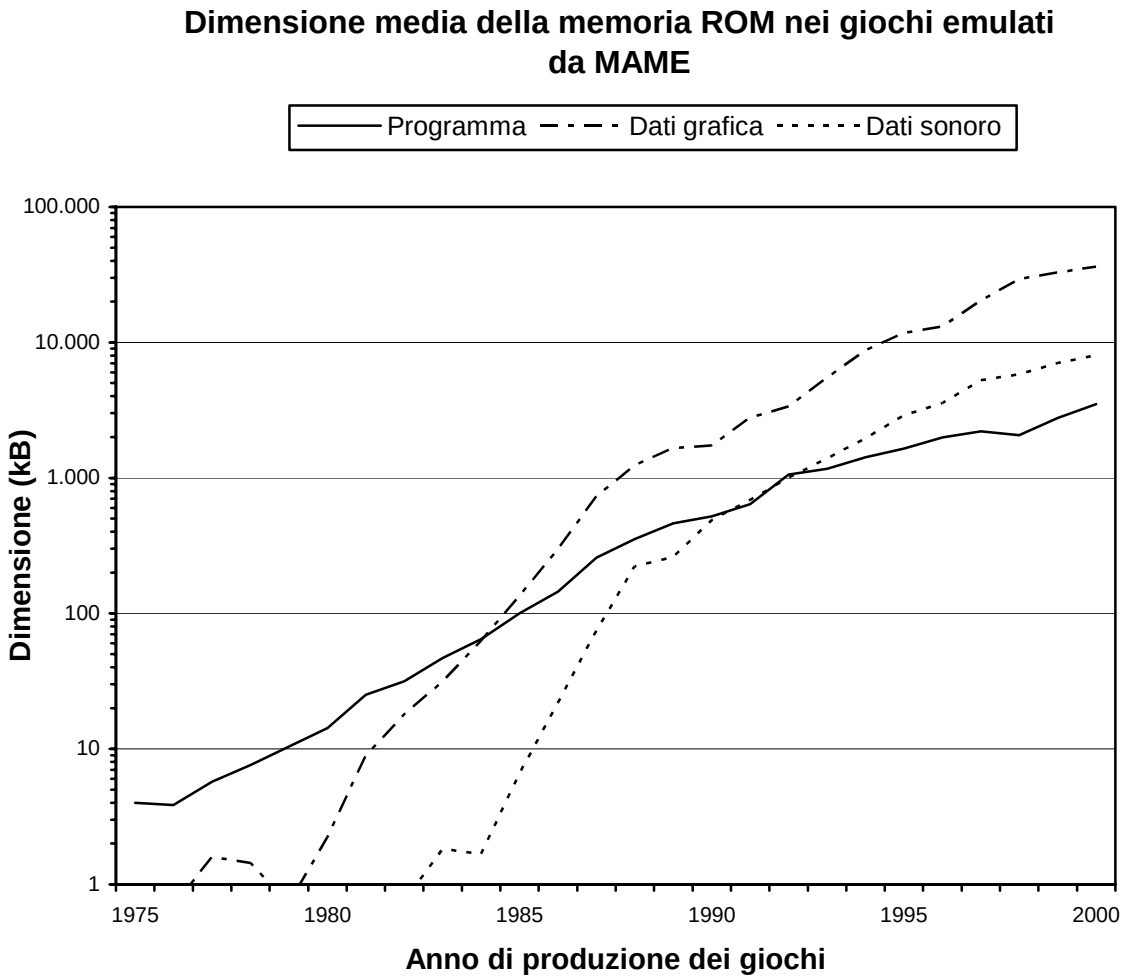
La differenza tra i tre tipi di ROM è in ogni caso irrilevante ai fini dell'emulazione e in gergo le immagini necessarie per far funzionare un emulatore sono chiamate genericamente ROM, o "*ROM set*".

Il progresso nell'industria dei semiconduttori è descritto dalla cosiddetta "legge di Moore", secondo cui la potenza raddoppia circa ogni 18 mesi [12], [13]. Il grafico seguente mostra la dimensione della memoria ROM in tutti i giochi emulati da MAME; l'accordo con la legge di Moore è piuttosto buono.

Dimensione della memoria ROM nei giochi emulati da MAME



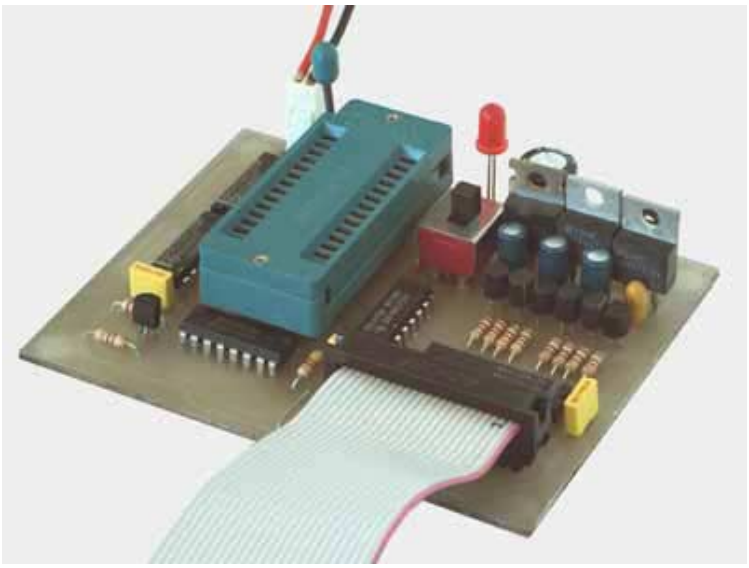
Il grafico seguente mostra le dimensioni medie, anno per anno, dei tre tipi principali di dati memorizzati su ROM: quelli riguardanti il programma (comprensivi del programma stesso), quelli riservati all'hardware grafico, e quelli riservati all'hardware sonoro.



Come si può vedere, la grandezza del programma è cresciuta più lentamente degli altri due tipi di dati ed è stata gradualmente superata da entrambi. Per quasi tutti gli anni '80, lo spazio dedicato prima solo alla grafica, poi anche al sonoro, è cresciuto molto rapidamente, attestandosi in seguito su una velocità più vicina a quella prevista dalla legge di Moore. Si può notare, soprattutto dalla fine degli anni '80 in poi, una concordanza d'andamento tra la dimensione delle ROM per la grafica e per il sonoro, che indica come questi due aspetti della qualità dei giochi progrediscano insieme. Il ritardo con cui lo spazio per il sonoro ha iniziato a crescere è probabilmente dovuto al fatto che musica ed effetti

sintetizzati, che richiedono pochi dati, erano comunque un'alternativa valida all'utilizzo di suoni campionati che hanno bisogno di molto più spazio.

L'emulazione della memoria ROM richiede anzitutto di leggere il contenuto dei *chip*, cosa che si fa mediante apparecchi detti *programmatori di EPROM*, che servono, come suggerisce il nome, non solo a leggere ma anche a scrivere sui *chip*.



Un programmatore di EPROM

La lettura può presentare difficoltà di ordine pratico poiché esistono moltissimi modelli sia di *chip* che di programmatori: può capitare che il programmatore non sia compatibile col *chip* che si deve leggere, che richieda la costruzione di un adattatore, oppure che il *chip* sia del tipo a montaggio superficiale e richieda quindi di essere dissaldato dalla scheda del gioco col rischio di essere rovinato.

Un'eventualità da tenere ben presente, soprattutto se il gioco è vecchio, è che il *chip* possa essere danneggiato e contenere dati difficili da leggere in modo affidabile, oppure del tutto errati. La scheda originale del gioco potrebbe anche essere guasta, rendendo impossibile verificare l'integrità delle ROM. Gli emulatori sono un importante ausilio per chi cerca di archiviare e preservare i videogiochi, perché consentono di verificare se i dati letti dalle ROM sono validi o se invece è necessario ripetere la

lettura, magari da una scheda diversa. Purtroppo il fatto che un certo insieme di ROM funzioni su un emulatore non è, da solo, garanzia che i dati letti siano identici a quelli programmati in fabbrica: per questo scopo è necessario che il gioco contenga una funzione di verifica della *checksum* delle ROM, eseguita all'avvio o dal menù di servizio. Un tale controllo è abbastanza frequente sulle ROM del programma ma, dato che le ROM contenenti gli altri dati di solito non sono direttamente accessibili da parte della CPU, è raro che il controllo venga eseguito su tutte le ROM della scheda.

Una volta lette le ROM, si può passare alla loro emulazione, che non presenta di per sé alcun problema: si tratta solo di caricare in memoria una copia del contenuto delle ROM. L'unica accortezza da seguire è assicurarsi che la caratteristica di "sola lettura" di questo tipo di memoria sia rispettata; ci sono, infatti, casi di giochi che, a causa di *bug* o di azioni deliberate, eseguono istruzioni che causano un accesso in scrittura alla memoria ROM, che ovviamente non ha alcun effetto sulla scheda originale. Se si consentisse a questi programmi di alterare il contenuto della memoria ROM, si potrebbero avere malfunzionamenti. Un esempio del genere è *Lady Bug* della Universal, in cui il sonoro smette di funzionare correttamente.

Per ognuno dei *chip* di memoria ROM presenti nel gioco si deve capire che tipo di dati contiene e in che posizione va caricato. L'etichetta posta sui *chip* e la loro posizione sulla scheda del gioco danno di solito già molte indicazioni. Oltre a questo, un rapido esame con un editor esadecimale permette ad un occhio allenato di riconoscere le regolarità tipiche di dati grafici o sonori, oppure altri particolari che caratterizzano i dati relativi al programma.

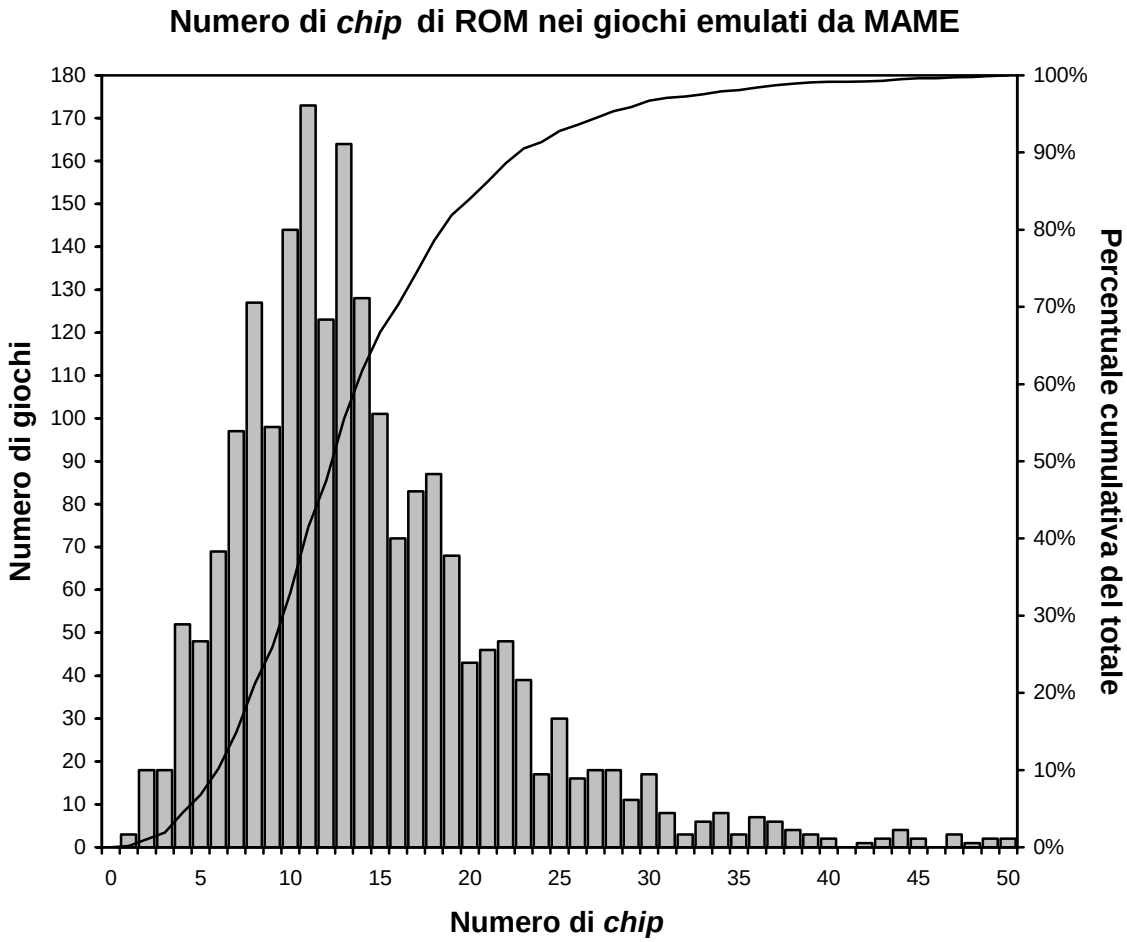
Quella che segue è parte di una ROM contenente il programma di *Pac-Man*: si possono notare numerose stringhe di testo in chiaro che consentono di identificarne facilmente la funzione.

```
00000760: 45524054 574F2F85 2F809202 47414D45 ER@TWO/./...GAME
00000770: 40404F56 45522F81 2F805202 52454144 @@OVER/./.R.READ
00000780: 595B2F89 2F90EE02 50555348 40535441 Y[/./...PUSH@STA
00000790: 52544042 5554544F 4E2F872F 80B20231 RT@BUTTON/./...1
000007A0: 40504C41 59455240 4F4E4C59 402F852F @PLAYER@ONLY@/./
000007B0: 80B20231 404F5240 3240504C 41594552 ...1@OR@2@PLAYER
000007C0: 532F8500 2F008000 9603424F 4E555340 S/../.....BONUS@
000007D0: 5055434B 4D414E40 464F5240 40403030 PUCKMAN@FOR@@@@00
000007E0: 30405D5E 5F2F8E2F 80BA025C 4028292A 0@]^_/./...\@()*
000007F0: 2B2C2D2E 40313938 302F832F 80C30243 +,-.@1980/./...C
00000800: 48415241 43544552 403A404E 49434B4E HARACTER@:@NICKN
00000810: 414D452F 8F2F8065 0126414B 41424549 AME/./.e.&AKABEI
00000820: 262F812F 80450126 4D41434B 59262F81 &/./.E.&MACKY&/.
00000830: 2F804801 2650494E 4B59262F 832F8048 /.H.&PINKY&/./.H
00000840: 01264D49 434B5926 2F832F80 76021040 .&MICKY&/./.v..@
00000850: 3130405D 5E5F2F9F 2F807802 14403530 10@]^_/./.x..@50
```

Quella che segue è invece sempre parte di una ROM di *Pac-Man*, ma questa volta contenente dati per l'hardware grafico. In questo caso si nota una certa regolarità dei dati e un'elevata frequenza di 00. Di solito in una ROM contenente dati grafici più del 50% dei byte sono 0.

```
000002A0: 0000FFFF 0000FFFF CCCCFFFF 000077FF ..............w.
000002B0: 0000FFFF 0000FFFF 000077FF CCCCFFFF ..........w.....
000002C0: 33333333 3333FFEE CCCCCCCC CCCCFF77 333333.........w
000002D0: 33333333 FFEE0000 CCCCCCCC FF770000 3333.........w..
000002E0: 00000000 0000EEFF 00000000 000077FF ..............w.
000002F0: 00000000 00000000 00000000 00000000 ................
00000300: 88CC2222 66CC8800 3377CC88 88773300 ..""f...3w...w3.
00000310: 2222EEEE 22220000 0000FFFF 44000000 ""..""......D...
00000320: 2222AAAA EEEE6600 66FFBB99 99CC4400 ""....f.f.....D.
00000330: CCEE2222 22664400 88DDFFBB 99880000 ..""""fD........
00000340: 88EEEE88 88888800 00FFFFCC 66331100 ............f3..
00000350: CCEE2222 22664400 11BBAAAA AAEEEE00 ..""""fD........
00000360: CCEE2222 22EECC00 00999999 DD773300 ..""".........w3.
00000370: 000000EE EE000000 CCEEBB99 88CCCC00 ................
00000380: CCEEAAAA 2222CC00 00669999 BBFF6600 ....""...f....f.
00000390: 88CC6622 22220000 77FF9999 99FF6600 ..f""""..w.....f.
```

Una volta divise le ROM in gruppi, metterle nel giusto ordine è di solito semplice; nella peggiore delle ipotesi ci si riesce per tentativi. Infatti, come si può verificare dal grafico seguente, il numero di *chip* è ridotto: il 90% dei giochi non ne ha più di 23.



I microprocessori a 8 bit, come lo Z80 o il 6809, di solito hanno uno spazio di indirizzamento a 16 bit, quindi possono accedere direttamente solo a 2^{16} byte di memoria. Spesso, però, la capacità della ROM dedicata al programma supera questo limite. Lo spazio di memoria ROM viene perciò diviso in *banchi*: scrivendo in un registro di controllo, la CPU decide quale parte della memoria ROM rendere visibile nella "finestra" ad essa dedicata.

RAM

La dotazione di RAM è, di solito, limitata. In generale, la maggior parte dei dati, nonché il programma stesso, sono contenuti in ROM ed è quindi necessaria solo una piccola quantità di RAM per le variabili del programma e per la memoria video.

L'emulazione non è diversa da quella della ROM, tranne ovviamente per il fatto che oltre alla lettura è consentita anche la scrittura. Va tenuto in considerazione che la decodifica degli indirizzi generati dalla CPU può essere parziale, per cui lo stesso *chip* di RAM può apparire più volte nella mappa di memoria, ad indirizzi fisici differenti, detti *mirror address*. Normalmente il programma usa solo una delle copie, ma ci sono casi in cui, spesso a causa di *bug* nel gioco, viene utilizzato anche qualche *mirror address*; in tal caso è necessario che l'emulatore ne tenga conto, per assicurare il corretto funzionamento.
Un caso molto particolare di *mirror address* è quello utilizzato da alcuni giochi della Data East tra cui, ad esempio, *Burger Time*[12]. In questo caso la memoria video appare ad un *mirror address*, ma con i bit 0-4 dell'indirizzo scambiati con i bit 5-9. Se si pensa a quello che la memoria video rappresenta sullo schermo, cioè una matrice 32x32, lo scambio dei bit significa che, mentre la versione "normale" della RAM scandisce la matrice per righe, la versione "speculare" la scandisce per colonne.

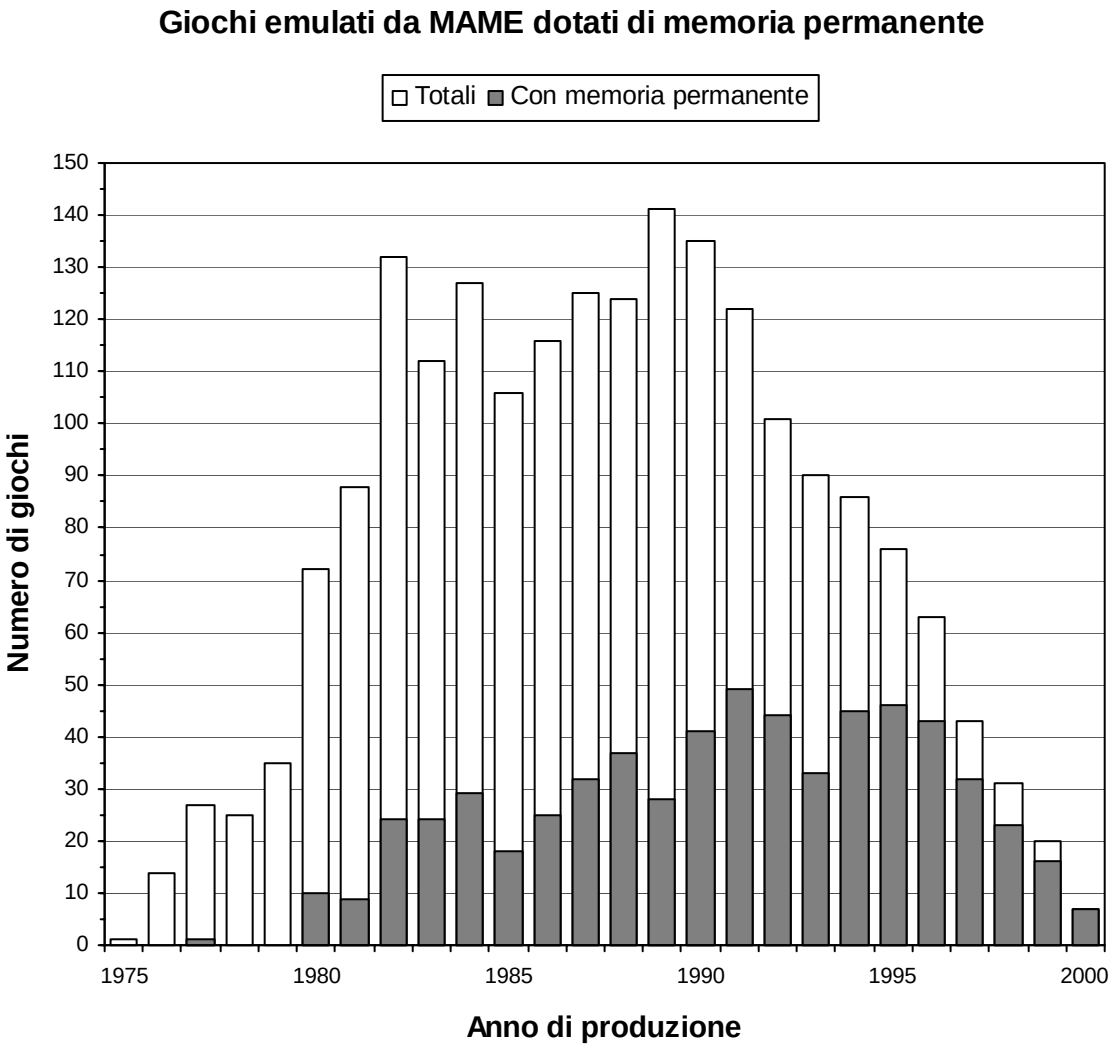
Una cosa di cui usualmente non si tiene conto nell'emulazione sono i *wait state* (tempi d'attesa) nell'accesso alla memoria, cioè i periodi in cui la CPU rimane ferma in attesa di poter leggere o scrivere un dato in RAM. I *wait state* possono influire sulle temporizzazioni, ma nei videogiochi arcade la differenza non è rilevante per il comportamento del gioco. Nel caso di console per videogiochi e home computer, invece, l'accurata riproduzione dei *wait state* è pressoché indispensabile; infatti, i giochi per queste macchine frequentemente spingono l'hardware ai suoi limiti, richiedendo una temporizzazione perfetta.

EEPROM e NVRAM

Molti giochi hanno un'area di memoria che non perde il contenuto quando viene tolta la corrente. In quest'area sono memorizzate impostazioni come la difficoltà o il numero di gettoni necessari per

[12] src/drivers/btime.c

iniziare una partita e, talvolta, anche statistiche sul numero di partite effettuate e la loro durata, in modo che il gestore possa regolare la difficoltà per massimizzare i profitti.



I *chip* di memoria permanente utilizzati nei videogiochi sono di due tipi:

- i *chip EEPROM (Electrically Erasable Programmable ROM)* sono simili come tecnologia ai *chip* EPROM, ma possono essere cancellati con un impulso elettrico anziché con l'esposizione a luce ultravioletta;
- i *chip NVRAM (Non-Volatile RAM)* sono essenzialmente *chip* di RAM alimentati da una batteria, cosicché il contenuto rimane inalterato anche in assenza di corrente elettrica.

Le differenze tra i due tipi di memoria sono sostanziali. La NVRAM si usa esattamente come la normale RAM, l'unica differenza è che il

contenuto non sparisce allo spegnimento. La EEPROM, invece, ha quasi sempre un'interfaccia seriale, quindi è necessario che il programma estragga i dati dalla EEPROM copiandoli in RAM e viceversa.

L'emulazione della NVRAM è identica a quella della RAM, con l'aggiunta che il contenuto della NVRAM viene salvato su disco all'uscita dell'emulatore e ripristinato alla successiva esecuzione del gioco.
L'emulazione della EEPROM seriale richiede invece di riprodurre il protocollo di comunicazione tra il *chip* e il resto del sistema. Sebbene ci siano sul mercato molti tipi diversi di *chip* di EEPROM, incompatibili tra loro, il principio di funzionamento è lo stesso e si può scrivere un codice generico in grado di emulare qualsiasi versione, in base ai valori dei parametri fissati all'avvio[13].

VIDEO

Si può dire che le capacità grafiche abbiano sempre fatto la differenza tra un videogioco da bar e i sistemi casalinghi. Negli ultimi anni la distanza si è ridotta, ma negli anni '80 il divario era particolarmente evidente: i giochi da bar apparivano molto più belli dei giochi per console e home computer dello stesso periodo. L'hardware grafico dei videogiochi è, infatti, specializzato per gestire nel modo più efficiente le loro necessità, senza appesantire inutilmente il lavoro della CPU.

L'hardware grafico è la parte più "personale" di una macchina da gioco. Mentre le altre parti, come le CPU o i *chip* audio, sono di solito componenti standard, documentati e utilizzati da più giochi, la parte grafica è quasi sempre sviluppata internamente dal produttore del videogioco. Questo è uno dei motivi per cui è spesso anche la più difficile da emulare.

Negli ultimi anni c'è stato un grosso cambiamento nell'aspetto dei videogiochi: grazie ai progressi dell'hardware, si è passati dalla classica

[13] src/machine/eeprom.c

grafica 2D alla grafica 3D. Già nei primi anni ’80 erano stati scritti giochi con effetti tridimensionali, grazie all’uso della grafica vettoriale. Questa particolare tecnica richiede l’uso di un monitor speciale, in cui il pennello di elettroni, invece di scandire l’immagine riga per riga, si muove liberamente attraverso tutto lo schermo, letteralmente disegnandovi delle linee.

Grafica vettoriale in *Star Wars* (Atari, 1983)

I monitor vettoriali erano troppo costosi e si guastavano frequentemente, perciò questa tecnica venne abbandonata.

MAME è solo agli inizi per quanto riguarda l’emulazione di giochi con moderno hardware 3D, quindi nel seguito ci occuperemo solo della classica grafica 2D.

Nonostante le grandi variazioni da un gioco all’altro, ci sono alcune parti dell’hardware grafico che si possono ricondurre a modelli generici; un gran numero di giochi usa qualche combinazione di queste parti.

Tilemap[14]

Le *tilemap*, termine che si può tradurre con "mappe di piastrelle", sono un metodo molto efficiente per mostrare e muovere blocchi di grafica grandi quanto tutto lo schermo; utilizzano poca memoria RAM e impegnano pochissimo la CPU. Lo schermo è diviso in una griglia composta da "piastrelle" tutte della stessa grandezza, usualmente quadrate con dimensioni di 8x8 o 16x16 pixel, raramente con dimensioni diverse, anche rettangolari. I "disegni" disponibili per le piastrelle vengono prelevati direttamente dall'hardware, di solito da memoria ROM, raramente da memoria RAM dedicata allo scopo. Per variare un elemento della griglia il programma deve solo indicare, nella locazione di memoria corrispondente, il disegno da utilizzare e la combinazione di colori. Spesso due byte sono sufficienti; ad esempio dedicando 4 bit al colore e 12 al disegno, si hanno a disposizione 2^4 diverse combinazioni di colori e 2^{12} diversi disegni. In questo modo, una tilemap di 32x32 piastrelle, ciascuna di 16x16 pixel, che contiene quindi $(32 \cdot 16)^2 = 262144$ pixel, ha bisogno di soli $2 \cdot 32^2 = 2048$ byte di memoria RAM per essere interamente descritta. A volte oltre a colore e disegno ci sono altri attributi, ad esempio un bit può indicare di riflettere il disegno rispetto all'asse verticale.

Un'altra caratteristica fondamentale delle tilemap è che possono essere fatte scorrere attraverso lo schermo cambiando il contenuto di alcune locazioni di memoria dette *scroll register*, registri di scorrimento. La tilemap è chiusa ai lati, quindi durante lo scorrimento ciò che esce da un lato dello schermo rientra in seguito da quello opposto. La tilemap può muoversi in blocco, utilizzando quindi due soli registri per lo spostamento lungo i due assi cartesiani, oppure può essere divisa in rettangoli più piccoli, liberi di muoversi indipendentemente gli uni dagli altri. Non è raro che questa indipendenza si spinga fino ad ogni singola riga (alta un pixel) che compone l'immagine.

[14] src/tilemap.c

Alcuni giochi possono gestire le tilemap in modo ancora più versatile, realizzando effetti ROZ (ROtate and Zoom), cioè di rotazione e ingrandimento.

Effetto ROZ sullo sfondo di *F-1 Grand Prix* (Video System, 1991)

L'emulazione delle tilemap è di solito piuttosto semplice; i dati da utilizzare si trovano nella memoria RAM, organizzati in modo regolare, e non è difficile intuire la funzione dei bit associati ad ogni piastrella.

Sprite

Una necessità primaria nei videogiochi è quella di muovere sullo schermo molti oggetti, indipendenti tra loro e dallo sfondo. Questo viene di solito realizzato mediante i cosiddetti *sprite*, termine gergale che si può tradurre "folletti". Gli sprite sono piccole immagini, ad esempio 16x16 pixel, che possono essere spostate liberamente sullo schermo indicandone le coordinate cartesiane. I disegni da utilizzare usualmente si trovano in memoria ROM utilizzata direttamente dall'hardware grafico, come nel caso delle tilemap. Oltre alle coordinate, il programma deve quindi indicare il numero del disegno da utilizzare e la combinazione di colori.

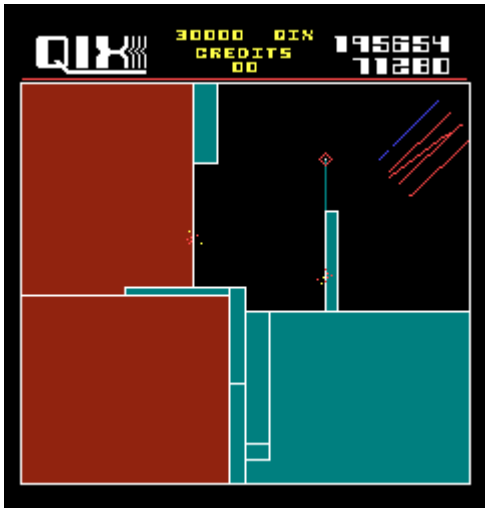
Solitamente è possibile riflettere il disegno rispetto ai due assi; a volte lo si può anche ingrandire o rimpicciolire. Le piccole dimensioni degli sprite non sono una limitazione, poiché si possono ottenere figure più grandi semplicemente usando più sprite uno a fianco all'altro; in alcuni casi ciò è anche gestito direttamente dall'hardware.

Questa descrizione è forzatamente sommaria, perché gli sprite sono la parte più "personale" dei videogiochi; ci sono, infatti, grandi variazioni da una scheda all'altra e molte altre possibili funzionalità oltre a quelle già descritte. Per questo motivo, quando si scrive l'emulatore per un gioco, il reverse engineering necessario per riprodurre gli sprite è spesso una delle cose più difficili perché richiede molto intuito ed esperienza.

Bitmap

Una *bitmap* consente di modificare indipendentemente il colore di ogni pixel dello schermo. Il suo pregio principale è senza dubbio la versatilità, infatti i PC gestiscono la grafica in questo modo.

In confronto alle tilemap, le bitmap richiedono molta più memoria RAM e sono più lente; d'altra parte, si possono ottenere effetti difficilmente realizzabili con le usuali tilemap. I difetti di solito sono più dei pregi, perciò nei videogiochi è più frequente l'uso delle tilemap.



Grafica bitmap in *Qix* (Taito, 1981)

Le bitmap sono il tipo di hardware grafico più semplice da emulare: basta copiare la bitmap dalla memoria RAM del gioco allo schermo del PC, convertendone i colori.

A meno che l'hardware video non sia costituito esclusivamente da bitmap, è necessario, prima ancora di iniziare il reverse engineering, determinare il modo in cui le "piastrelle" sono memorizzate nelle ROM. I bit che compongono ogni pixel dell'immagine possono, infatti, essere disposti in vari modi, e i progettisti scelgono di volta in volta quello più appropriato per rendere l'hardware più semplice ed economico. MAME ha una funzione generica per convertire i dati grafici da un formato qualsiasi nel formato usato internamente.

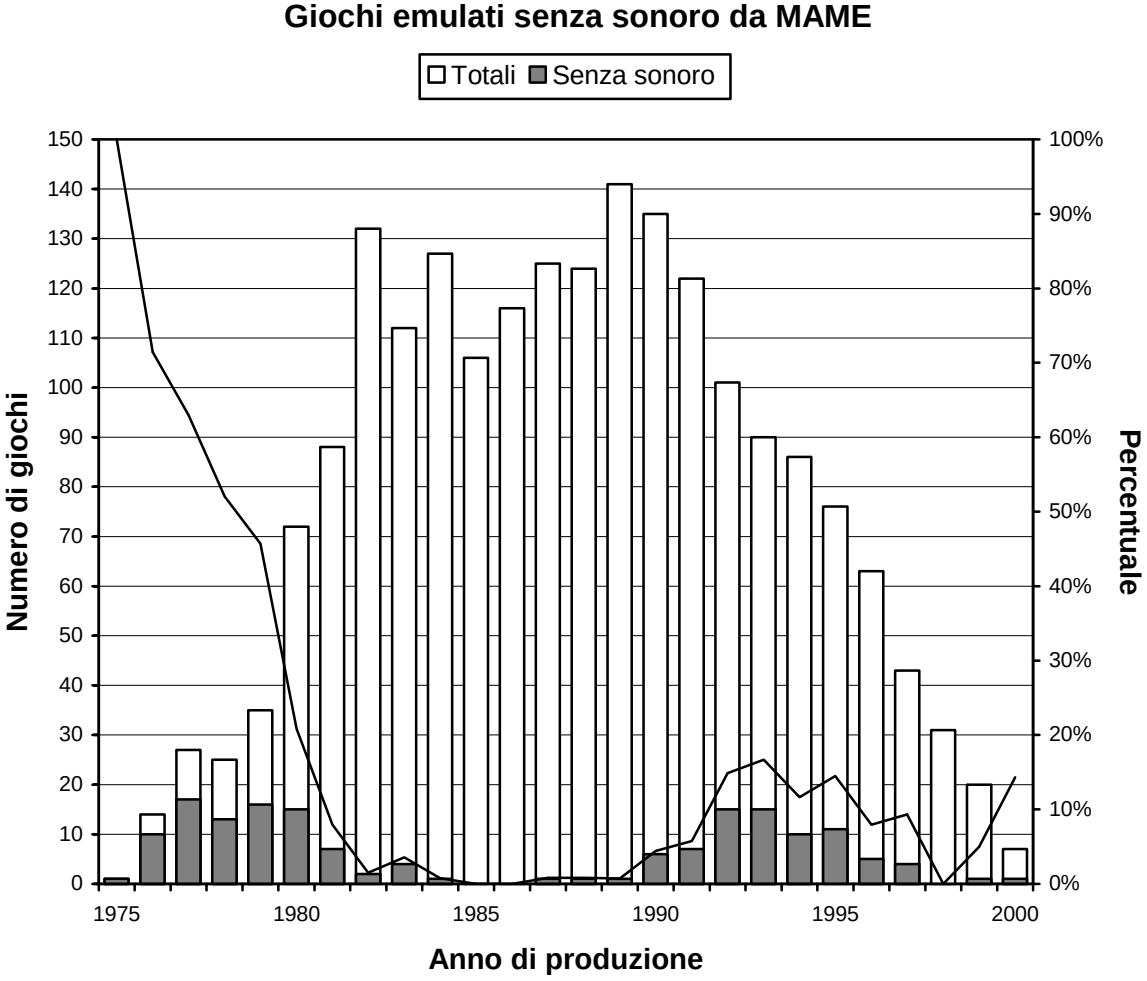
Una potenziale fonte di difficoltà durante l'emulazione dell'hardware video è il modo in cui comporre le varie parti per formare l'immagine finale. Ad esempio, se il gioco usa tre tilemap e un insieme di sprite, è comune che ogni sprite possa essere messo di fronte o dietro ai vari strati creati dalle tilemap. Anche per il meccanismo delle priorità non ci sono schemi prefissati, e i vari giochi usano molti modi diversi per gestirle in modo flessibile.

Audio

L'audio è una parte importante di qualsiasi videogioco: effetti sonori come la musichetta introduttiva di *Pac-Man* sono entrati a far parte della cultura popolare. È evidente però che, per il funzionamento del gioco, video e audio non hanno la stessa importanza: dell'audio se ne può fare a meno, mentre del video no.
La stessa cosa vale per gli emulatori: la parte sonora spesso è una delle ultime cose che si aggiungono, e può anche mancare del tutto se per qualche motivo la sua emulazione presenta delle difficoltà.

Nel grafico seguente si può vedere quanti dei giochi emulati da MAME sono tuttora privi di sonoro. Come è facile osservare, la mancanza riguarda soprattutto i giochi più vecchi: questo è successo perché nei primi videogiochi il sonoro era prodotto mediante circuiti analogici, la cui emulazione è più complessa. Per i giochi di quel periodo, in molti casi

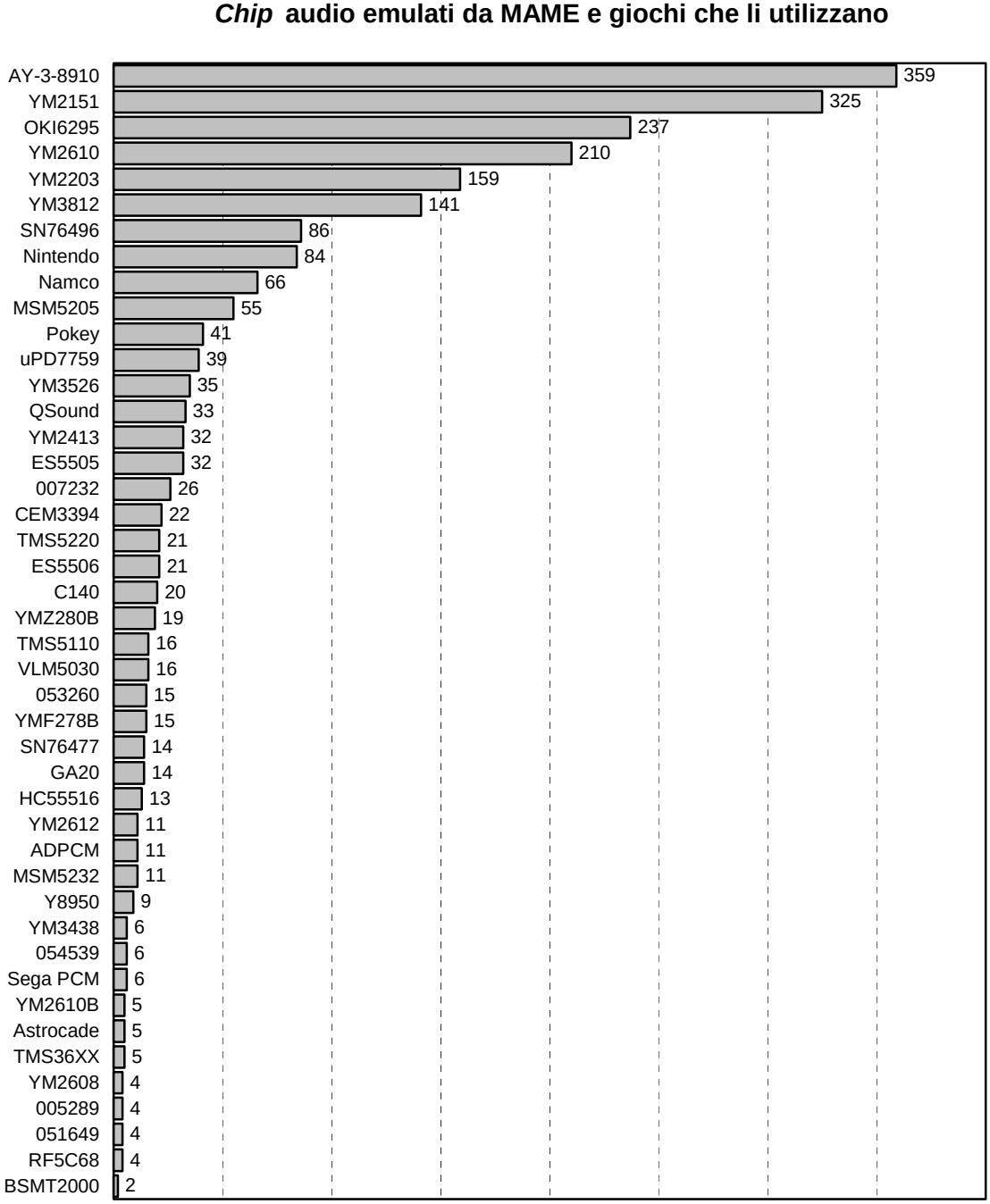
invece di emulare i circuiti si sono semplicemente utilizzati i suoni registrati dalla scheda originale.



Dal 1980 in poi, la situazione è diventata molto simile a quella delle CPU: i giochi hanno iniziato ad usare *chip* generatori di suono, di solito documentati, oppure sufficientemente semplici da capire per realizzarne il reverse engineering. Per i giochi prodotti dopo il 1990 c'è stato un aumento di quelli privi della parte sonora, per la maggior parte dovuto all'uso di *chip* non ancora emulati ma che probabilmente lo saranno in futuro.

Attualmente MAME emula una cinquantina di *chip* audio, e come si può vedere dal grafico seguente in confronto con quello a pag. 34, la distribuzione non è sbilanciata quanto nel caso delle CPU.
I tre *chip* più usati sono anche rappresentanti delle tre principali famiglie

di sintesi musicale: sintesi di toni ad onde quadre (AY-3-8910), sintesi FM (YM2151) e riproduzione di campioni ADPCM (OKI6295).



L'emulazione di un *chip* audio può iniziare, come per le CPU, con il reperimento della documentazione ufficiale, ma in questo campo la frequenza di componenti *custom* non documentati è più elevata, quindi bisogna ricorrere più spesso ad un reverse engineering di tipo black box.

Anche quando la documentazione è disponibile, raramente questa è esauriente. Ad esempio, molti *chip* possono generare un rumore pseudocasuale; i numeri pseudocasuali necessari sono generati tramite una formula che non è mai documentata, dato che la sua conoscenza non è necessaria per il normale utilizzo del *chip*. La formula può però essere trovata, grazie all'algoritmo di Berlekamp-Massey, analizzando il suono prodotto dal *chip* [14].

Perfezionare l'emulazione dell'audio è forse un po' più difficile che perfezionare la grafica, perché i piccoli errori nella grafica sono più facilmente identificabili, mentre per individuarli nel sonoro serve un orecchio ben allenato.

Dispositivi di input

Un videogioco da bar ha bisogno di gestire input da tre fonti diverse: la gettoniera, i controlli per il personale di servizio (non accessibili dall'esterno) e ovviamente i comandi del giocatore.

La gettoniera è l'unico dispositivo di input di cui nessun gioco arcade può essere privo. Oltre alla fessura per le monete, spesso è anche presente un interruttore che si attiva quando qualcuno tenta di forzare lo sportello della gettoniera o scuote il mobile con eccessiva violenza; in tal caso, il gioco si blocca.

Il personale di servizio, aprendo la gettoniera per accedere all'interno, può utilizzare alcuni comandi speciali. Di solito è presente un pulsante che consente di iniziare una partita senza inserire monete. Spesso un pulsante o un interruttore consente di far eseguire al programma alcuni controlli per verificarne il corretto funzionamento, ad esempio un test della RAM o una verifica delle ROM. Sulla scheda del gioco si possono trovare, soprattutto nei giochi meno recenti, uno o più interruttori DIP che servono per regolare la difficoltà e altri parametri di funzionamento; nei giochi più recenti, questi scomodi interruttori sono spesso sostituiti

da menu interattivi visualizzati sul monitor, e le impostazioni sono memorizzate in un'area di memoria non volatile.

Il dispositivo di input per eccellenza in un videogioco, tanto che ne è diventato il simbolo, è il joystick, cioè una leva che può essere mossa dal giocatore nelle quattro direzioni. All'interno del joystick ci sono quattro interruttori, che vengono attivati dal movimento della leva. Usualmente due interruttori adiacenti, ad esempio "alto" e "destra", possono essere attivati contemporaneamente, ma alcuni giochi richiedono che il joystick sia configurato in modo che sia meccanicamente impossibile chiudere più di un interruttore alla volta. Questa configurazione è detta "4-way", mentre l'altra "8-way". Al joystick sono di solito affiancati uno o più pulsanti.

In alcuni tipi di gioco, joystick e pulsanti non sono sufficienti per ottenere un buon controllo, perciò si utilizzano altri dispositivi, ad esempio: volante e pedali nei giochi di guida; uno *spinner,* cioè una manopola che può ruotare intorno al suo asse, per consentire spostamenti veloci e precisi, ad esempio in *Arkanoid* o *Tempest;* una *trackball* nei giochi in cui si richiedono rapidi spostamenti in due direzioni, ad esempio *Missile Command* o *Marble Madness*; pistole o fucili per mirare a bersagli sullo schermo.

Il peculiare sistema di input di *Slick Shot* (Incredible Technologies, 1990)

Spesso i produttori si sono sbizzarriti nell'inventare sistemi di controllo originali, per rendere i propri giochi più interessanti, e forse anche per rendere più difficile la produzione di copie illegali. Così si sono potuti vedere palloni da calcio, canne da pesca, biciclette, fucili con mirino telescopico, skateboard, palle da biliardo con stecca, eccetera.

L'emulazione dei dispositivi di input non presenta difficoltà sul piano teorico poiché, nella maggior parte dei casi, si tratta solamente di far comparire il valore dell'input in una locazione di memoria. Le difficoltà maggiori sono di tipo pratico: i dispositivi di input disponibili su un PC sono, infatti, fisicamente molto diversi da quelli di un videogioco arcade, ed è quindi necessario eseguire delle conversioni. Anche una cosa apparentemente ovvia come associare quattro tasti alle quattro direzioni di un joystick nasconde delle insidie, in quanto in un joystick solo una o due direzioni possono essere attive contemporaneamente, mentre l'utente potrebbe anche premere tutti e quattro i tasti insieme. L'emulatore deve assicurarsi che al gioco emulato pervengano solamente input validi, per evitare possibili malfunzionamenti.

Per i giocatori più accaniti, la diffusione degli emulatori ha spinto alcune aziende a produrre dei pannelli di controllo identici a quelli dei videogiochi arcade. <23>, <24> e <25> sono alcuni esempi.

Uno dei joystick progettati per gli emulatori

4. CRITTOGRAFIA

Motivazioni

La parola *bootleg* risale come minimo al 19° secolo, ma fu utilizzata soprattutto negli anni Venti, durante il Proibizionismo [17]. Il suo significato originale riguarda la produzione e vendita illegale di alcolici, ma si è in seguito esteso ad altri mondi, ad esempio la registrazione non autorizzata di concerti. Nel caso dei videogiochi da bar, un *bootleg* è una copia illegale di un gioco, spesso modificata, ad esempio per cambiarne il titolo o rimuovere le indicazioni di copyright.

Per evitare la duplicazione illegale, molti produttori furono costretti ad introdurre sistemi di protezione: alcuni di essi sono già stati indicati a pag. 38, mentre in questo capitolo ci occuperemo diffusamente di uno dei più utilizzati, la crittografia.

Le ROM crittografate possono essere sia quelle contenenti dati grafici che, più frequentemente, quelle contenenti il codice di una CPU, di solito quella principale. Nel secondo caso vengono utilizzate CPU *custom,* che eseguono la decodifica internamente, oppure moduli esterni. Il primo metodo è più sicuro, poiché il secondo richiede che la comunicazione tra modulo di decodifica, CPU e ROM sia protetta dall'intrusione con mezzi fisici, ad esempio resine epossidiche.

Quasi sempre la cifratura è eseguita in modo diverso per codici d'istruzione (*opcode*) e dati; questo rende più complessa la crittanalisi.

Dato che la decodifica deve avvenire in tempo reale durante l'esecuzione del programma, gli algoritmi di cifratura devono per forza essere piuttosto semplici. Nel tempo c'è stata un'evoluzione, passando dai semplici cifrari monoalfabetici dei primi anni '80 ad algoritmi più complessi che sono tuttora ignoti. In molti casi si ha l'impressione che, per compensare le limitazioni degli algoritmi, la sicurezza sia stata ottenuta mediante l'utilizzo di chiavi molto lunghe.

Talvolta, nei giochi più recenti, le chiavi sono memorizzate in una zona di NVRAM alimentata da una batteria al litio; quando la batteria (appropriatamente soprannominata "suicide battery") si esaurisce, il gioco smette di funzionare e va rispedito al produttore per essere "riparato". Sulla legittimità di una pratica del genere che, di fatto, rende le schede di alcuni videogiochi dei prodotti "a tempo" destinati ad autodistruggersi, ci sono pareri contrastanti. Alcuni collezionisti, comunque, si sono organizzati per risolvere il problema alla radice, modificando le schede per evitare l'uso delle batterie <26>.

Stern

Tra i primi appassionati di emulazione circolava una battuta secondo cui "qualsiasi gioco può girare sull'hardware di *Scramble*". La battuta nacque perché molti dei giochi supportati dalle prime versioni di MAME giravano appunto su questa scheda, che ai suoi tempi era molto diffusa.

Dato che all'epoca il mercato dei videogiochi era in forte crescita, la scheda fu utilizzata da molti per realizzare rapidamente nuovi giochi senza dover progettare anche l'hardware. Alcuni giochi prodotti dalla Stern hanno le ROM grafiche cifrate, probabilmente per evitare una facile pirateria visto che la scheda era così diffusa. L'algoritmo agisce solo sugli indirizzi, non sui dati, cioè si limita a mescolare un po' i byte all'interno delle ROM. Se la protezione non viene riprodotta, il gioco continua a funzionare regolarmente, ma la grafica è confusa.

La protezione esegue operazioni logiche sui bit dell'indirizzo; quella che segue, ad esempio, è la formula utilizzata per il gioco *Minefield*[15].

```
A5′:= A3 ⊕ A7
A7′:= A2 ⊕ A9 ⊕ (A0 ∧ A5) ⊕ (A3 ∧ A7 ∧ (A0 ⊕ A5))
A9′:= A0 ⊕ A5 ⊕ (A3 ∧ A7)
```

KONAMI-1[16]

Questa CPU è un Motorola 6809 che esegue internamente una decodifica degli *opcode*. È utilizzata da un buon numero di giochi della Konami del periodo 1983-1986.
L'algoritmo è un semplice schema XOR. Agisce sui bit 1, 3, 5 e 7 dell'*opcode* e dipende dai bit 1 e 3 dell'indirizzo; gli altri bit dell'*opcode* non sono alterati. Lo schema è il seguente:

```
D1′ := D1 ⊕ ¬A3
D3′ := D3 ⊕ A3
D5′ := D5 ⊕ ¬A1
D7′ := D7 ⊕ A1
```

La protezione è quindi riproducibile facilmente con un po' di porte logiche; il fatto che sia applicata solo agli *opcode* non rappresenta un problema poiché il 6809, come molte altre CPU, segnala all'esterno se sta eseguendo il prelievo di un *opcode* o di dati.
In effetti, la prima emulazione di questa protezione è stata proprio ottenuta studiando una scheda *bootleg* del gioco *Gyruss* che sostituiva la CPU Konami-1 con un 6809 standard e alcune porte logiche.

[15] src/machine/scramble.c
[16] src/machine/konami.c

SEGA SYSTEM 1[17]

Molti giochi della Sega del periodo 1982-1986 usano versioni *custom* dello Z80. Il nome che ho scelto è arbitrario, dovuto al fatto che la CPU è utilizzata principalmente per i giochi che girano sulla scheda System 1.

Tutte le CPU di questa famiglia utilizzano lo stesso algoritmo di cifratura, ma con chiavi differenti. La chiave è programmata nella CPU in fase di produzione e non può essere modificata.

Qualche informazione sugli ideatori del sistema si trova, in chiaro, all'interno delle ROM di alcuni giochi[18]:

Buck Rogers	`SECULITY BY MASATOSHI,MIZUNAGA`
Super Locomotive	`SEGA FUKUMURA MIZUNAGA`
Yamato	`SECULITY BY M,MIZUNAGA`
Regulus	`SECULITY BY SYUICHI,KATAGI`
Up'n Down	`19/SEP 1983  MASATOSHI,MIZUNAGA`
Mister Viking	`SECURITY BY S.KATAGI  CONTROL CHIP M140`
SWAT	`SECURITY BY S.KATAGI`
Water Match	`PROGRAMED BY KAWAHARA&NAKAGAWA`
Flicky	`SECURITY BY S.KATAGI`
Star Force	`STAR FORCE TEHKAN. SECURITY BY SEGA ENTERPRISESE`

Una nota curiosa riguardo a *Star Force* è che, al contrario di tutti gli altri, non è prodotto dalla Sega ma dalla Tehkan; nonostante la CPU abbia un numero di codice diverso, usa la stessa chiave di *Super Locomotive*. Evidentemente alla Sega non volevano fare troppo lavoro per aiutare una ditta concorrente a proteggersi dalle copie!

La CPU decodifica internamente sia *opcode* che dati, con lo stesso algoritmo, ma codifiche differenti. L'algoritmo agisce sui bit 3, 5 e 7 del dato da codificare; i tre bit possono essere permutati in qualsiasi modo e ognuno di essi può essere invertito. Le codifiche possibili sono dunque:

---

[17] src/machine/segacrpt.c

[18] Gli errori di ortografia sono così come appaiono nell'originale

$3! \cdot 2^3 = 48$.

La permutazione e le inversioni da applicare dipendono dai bit 0, 4, 8 e 12 dell'indirizzo a cui si trova il byte e dal fatto che il byte sia un *opcode* o un dato; in teoria, quindi, la CPU potrebbe utilizzare 2^5 codifiche diverse. I giochi esaminati, però, ne usano solamente 6, tranne qualcuno che ne usa 7; in quelli che ne usano 7, una delle codifiche è l'identità. Sembrerebbe quindi che la CPU non possa gestirne un numero maggiore.

Il reverse engineering di questo algoritmo è stato realizzato studiando due versioni di *Pengo*, una cifrata e l'altra in chiaro, quasi identiche. Una volta individuato l'algoritmo, è stato possibile decifrare altri giochi. Questa operazione è piuttosto semplice e può essere fatta a mano in un paio d'ore: infatti, ogni byte può essere decifrato solo in 8 modi diversi, e man mano che si trovano parti della chiave si riducono i valori ammissibili per le altre parti, quindi la ricerca procede speditamente.

V30[19]

Il V30 della NEC è una CPU derivata dall'Intel 8086 che, tra le funzionalità aggiunte, ha anche la possibilità di avere all'interno una tabella di conversione degli *opcode*. Si tratta, quindi, di un cifrario a sostituzione monoalfabetica con un alfabeto di 256 caratteri.

Questa CPU è stata usata dalla Irem in un buon numero di giochi nel periodo 1991-1994: in alcuni, ad esempio *Bomberman*, come CPU principale; in altri, ad esempio *Gunforce*, come CPU secondaria per il sonoro.

È noto che i cifrari monoalfabetici hanno una sicurezza molto bassa quando si tratta di proteggere un messaggio di testo, ma decifrare un programma cifrato in questo modo è comunque difficoltoso. Molte

[19] src/machine/irem_cpu.c

istruzioni di una CPU hanno funzioni simili e spesso alcune di loro sono usate solo una o due volte all'interno di tutto il programma, quindi non è facile determinare con esattezza le corrispondenze tra istruzioni e *opcode*.

Nel caso dei giochi della Irem, un aiuto fondamentale è venuto da altri giochi della stessa ditta, che usano come CPU per il sonoro un normale Z80 invece del V30 protetto. In alcuni casi, i programmi delle due CPU sono pressoché identici per lunghe parti, come si può verificare in questi due stralci messi a confronto:

***Gunforce* (V30)**


```
01FF2: D2 06 00      mov  al,[$0006]
01FF5: 15 C0         and  al,al
01FF7: C7 03         jne  $1FFC
01FF9: F9 36 F1      jmp  $1132
01FFC: 82 75 45 07   inc  byte ss:[iy+$07]
02000: 82 20 45 07   mov  al,ss:[iy+$07]
02004: 52 07         and  al,$07
02006: C7 2E         jne  $2036
02008: 82 75 4D 06   dec  byte ss:[iy+$06]
0200C: C7 28         jne  $2036
0200E: 82 20 45 05   mov  al,ss:[iy+$05]
02012: 82 07 45 06   mov  ss:[iy+$06],al
02016: 82 20 5D 02   mov  bl,ss:[iy+$02]
0201A: 82 20 7D 03   mov  bh,ss:[iy+$03]
0201E: 82 20 45 04   mov  al,ss:[iy+$04]
02022: 82 AE 07      sub  al,ss:[bw]
02025: C7 03         jne  $202A
02027: F9 08 F1      jmp  $1132
0202A: 6D 01         mov  al,$01
0202C: 32 02         jnc  $2030
0202E: BA D8         neg  al
02030: 82 9D 07      add  al,ss:[bw]
02033: 82 07 07      mov  ss:[bw],al
02036: 48            ret
```

***Bomberman* (Z80)**


```
0AB8: 3A 06 FF   ld   a,($FF06)
0ABB: A7         and  a

0ABC: CA 80 00   jp   z,$0080
0ABF: DD 34 07   inc  (ix+$07)
0AC2: DD 7E 07   ld   a,(ix+$07)
0AC5: E6 07      and  $07
0AC7: C0         ret  nz
0AC8: DD 35 06   dec  (ix+$06)
0ACB: C0         ret  nz
0ACC: DD 7E 05   ld   a,(ix+$05)
0ACF: DD 77 06   ld   (ix+$06),a
0AD2: DD 6E 02   ld   l,(ix+$02)
0AD5: DD 66 03   ld   h,(ix+$03)
0AD8: DD 7E 04   ld   a,(ix+$04)
0ADB: 96         sub  (hl)

0ADC: CA 80 00   jp   z,$0080
0ADF: 3E 01      ld   a,$01
0AE1: 30 02      jr   nc,$0AE5
0AE3: ED 44      neg
0AE5: 86         add  a,(hl)
0AE6: 77         ld   (hl),a
0AE7: C9         ret
```

Dai listati è evidente che il programma per V30 è una traduzione letterale del programma per Z80.

Un'altra circostanza fortunata è stata che in molti casi il V30 usato come CPU principale in un gioco usa la stessa chiave di quello usato come CPU secondaria in un altro, quindi una volta decifrato il secondo si era già a buon punto anche per decifrare il primo.

BURGER TIME[20]

L'algoritmo utilizzato dalla Data East in *Burger Time* e, con piccole variazioni, in altri giochi dello stesso periodo, è interessante non per la cifratura in sé, che è una semplice permutazione dei bit, ma per come essa viene applicata.

La CPU è un 6502 che decodifica solo gli *opcode* il cui indirizzo ha i bit 2 e 8 impostati e, cosa più importante, solo se l'istruzione precedente aveva causato un accesso in scrittura alla memoria. La decodifica, perciò, dipende dal flusso del programma. Questo rende impossibile decifrare le ROM senza interpretare il codice in esse contenuto.

THE GLOB[21]

Questo gioco della Epos esiste in due versioni, una che gira su una scheda dedicata, l'altra che gira sulla scheda di *Pac-Man*. La prima non è protetta, mentre la seconda lo è, probabilmente per gli stessi motivi già visti per i giochi Stern.

La codifica raffina l'idea di *Burger Time*, cioè cambia sotto il controllo del programma. In questo caso la cifratura è una classica permutazione e inversione di bit, che può avvenire in più modi diversi. Quando la CPU, un normale Z80, esegue l'istruzione `IN` su una porta di I/O pari, viene selezionata la permutazione successiva, mentre se la porta è dispari, viene selezionata la permutazione precedente.

Le permutazioni e la logica di selezione della permutazione attiva sono realizzate mediante una PAL (*Programmable Array Logic*), situata insieme allo Z80 e alle ROM su una schedina ricoperta di resina epossidica. Il reverse engineering è stato eseguito penetrando fisicamente all'interno del modulo di protezione.

---

[20] src/drivers/btime.c

[21] src/machine/theglobp.c

KABUKI[22]

Kabuki è il nome in codice di una CPU usata dalla Capcom per alcuni giochi nel periodo 1989-1993, ad esempio *Buster Bros*. La CPU è dotata di una "suicide battery" ed utilizza un algoritmo non banale.

La parte principale dell'algoritmo, che chiamerò "*bitswap*", è lo scambio di ognuna delle quattro coppie di bit consecutivi del byte da decifrare. Ogni coppia può essere o non essere scambiata: questo dipende dall'indirizzo a cui si trova il byte, dal fatto che esso sia un dato o un *opcode*, e dalla chiave di cifratura memorizzata nella CPU. Il *bitswap* viene eseguito quattro volte; tra ogni coppia di *bitswap* viene eseguita una rotazione a sinistra dei bit del byte, e una volta anche uno XOR con un valore che fa parte della chiave di cifratura.
Riassumendo, le operazioni da eseguire per decifrare un byte sono nell'ordine:

```
bitswap
ROL
bitswap
XOR
ROL
bitswap
ROL
bitswap
```

Questo algoritmo è indubbiamente più complesso di quelli esaminati finora, ma ha delle debolezze che si possono sfruttare per ricercare la chiave.

Per i dettagli sul funzionamento dell'algoritmo si può far riferimento al file indicato in nota; in questa sede ci interessa notare che i primi due *bitswap* sono controllati dai bit 0-7 dell'indirizzo, mentre i secondi due dai bit 8-15.

Se un byte in chiaro è 00_{16} (o FF_{16}), dal momento che ha tutti i bit uguali non è influenzato dagli ultimi due *bitswap;* questi ultimi dipendono dai

[22] src/machine/kabuki.c

bit 8-15 dell'indirizzo, quindi la codifica di tale byte dipende solo dai bit 0-7 dell'indirizzo. Se, come spesso accade, nei dati in chiaro c'è una lunga sequenza di 00_{16} (o FF_{16}), i dati cifrati conterranno una sequenza di 256 byte ripetuta più volte. Si può sfruttare questo fatto per trovare, con una ricerca a forza bruta, i pochi valori ammissibili per una parte della chiave. La chiave può poi essere completata, sempre con una ricerca a forza bruta, tramite un attacco di tipo *known plaintext*.

DATA EAST CUSTOM 56/74[23]

Due dei tanti *chip custom* prodotti dalla Data East, utilizzati nel periodo 1991-1995, utilizzano ROM crittografate. Sono *chip* grafici, generatori di *tilemap*, e hanno come numeri di codice 56 e 74.

L'algoritmo lavora su blocchi di 2048 word da 16 bit. La chiave di cifratura è fissa, quindi una volta individuata è valida per tutti i giochi che utilizzano lo stesso *chip*.

La decodifica consiste in tre operazioni distinte, identiche per ogni blocco:

1. le word sono permutate all'interno del blocco secondo un ordine prestabilito;
2. su ogni word viene eseguito uno XOR con uno di 16 possibili valori predefiniti, in base alla posizione all'interno del blocco;
3. i bit di ogni word vengono permutati in uno di 8 possibili modi predefiniti, in base alla posizione all'interno del blocco.

Questi *chip* sono un esempio dei tentativi di supplire alle carenze degli algoritmi di cifratura usando chiavi molto lunghe: la dimensione dello spazio delle chiavi, infatti, in base alla precedente descrizione, è

$$2048! \cdot 16^{2048} \cdot 8^{2048} \cong 0.62 \cdot 10^{10210}$$

[23] src/machine/decocrpt.c

La tabella che contiene la chiave all'interno del *chip* presumibilmente utilizza

$$2048 \cdot 11 + 2048 \cdot 4 + 2048 \cdot 3 = 2048 \cdot 18 = 36864$$

bit di memoria.

Il reverse engineering dell'algoritmo è stato eseguito con un attacco di tipo *known plaintext*, grazie alla disponibilità di versioni *bootleg*, con ROM in chiaro, di *Tumble Pop* (*chip* 56) e *Funky Jet* (*chip* 74).
L'attacco è stato facilitato da una debolezza dell'algoritmo: quando un blocco è interamente costituito da 0000_{16} (o $FFFF_{16}$), i passi 1 e 3 sono ininfluenti, quindi lo schema XOR diventa chiaramente visibile.

Una volta eliminato lo schema XOR, contando per ogni n il numero di word con n bit impostati, sia nella ROM cifrata che in quella in chiaro, si è verificato che i conteggi corrispondevano per ogni blocco; questo confermava che la parte rimanente della cifratura consisteva solo nella permutazione delle word all'interno di ogni blocco e nella permutazione dei bit all'interno di ogni word.

Assumendo una distribuzione uniforme, la probabilità che una word di 16 bit abbia n bit impostati è

$$\frac{1}{2^{16}}\binom{16}{n} < 0{,}2$$

Nella ROM ci sono 128 blocchi e la permutazione delle word è la stessa in ogni blocco. Fissata una posizione nei blocchi cifrati e una nei blocchi in chiaro, la probabilità che le word in quelle due posizioni abbiano sempre lo stesso numero di bit è minore di $0{,}2^{128}$, che è un numero dell'ordine di 10^{-90}; in altri termini, è praticamente impossibile che la corrispondenza si verifichi per caso. Attraverso questo confronto si può dunque facilmente ricostruire la permutazione del passo 1.

Una volta ricostruito il corretto ordine delle word, rimangono solamente da trovare le permutazioni di bit del passo 3. Il procedimento è analogo a

quello precedente, cioè utilizzare le informazioni ridondanti fornite dai 128 blocchi per isolare l'unica permutazione valida.

NEO-GEO[24]

Il Neo-Geo è stato senza dubbio il più longevo sistema per videogiochi arcade mai prodotto. Introdotto nel 1990, ha resistito per più di dieci anni; vanta una libreria di circa 150 titoli, e sono tuttora sviluppati nuovi giochi, nonostante la casa produttrice originaria, la SNK, sia fallita nel 2001. Visitando una qualsiasi sala giochi, si troveranno sicuramente un paio di cabinati dotati di questo sistema, con l'ultima versione di *King of Fighters* o di *Metal Slug*, o magari qualche "sempreverde" come *Puzzle Bobble*.

Il successo del Neo-Geo è in parte dovuto al fatto che è un sistema a cartucce, venduto anche sotto forma di console per l'uso casalingo. Sebbene le cartucce per la versione arcade e per quella home siano diverse, il gioco contenuto è assolutamente identico. Nella versione arcade, è addirittura possibile inserire più giochi contemporaneamente: l'utente può scegliere il gioco preferito con un pulsante sul cabinato.

La pirateria deve aver rappresentato un serio problema, poiché il sistema è molto diffuso ed è privo di protezioni, ed è quindi facile realizzare copie illegali delle cartucce. Aggiungere una protezione non era però una cosa semplice, perché la scheda principale non poteva più essere modificata, mentre le cartucce contenevano solo le ROM e poco più; nonostante queste difficoltà, nei giochi prodotti a partire dal 1999 le ROM grafiche sono cifrate.

Poiché i dati dovevano uscire in chiaro dalla cartuccia, l'unica possibilità per cifrare il contenuto delle ROM era inserire dentro alla cartuccia dei *chip custom* che decifrassero il contenuto delle ROM prima di inviarlo all'esterno. Come abbiamo detto in precedenza, una soluzione del genere,

[24] src/machine/neocrypt.c

in cui la comunicazione tra le ROM e il resto del sistema non è protetta, è intrinsecamente insicura; la protezione, infatti, non ha resistito a lungo ai pirati. Le versioni *bootleg* hanno consentito un attacco all'algoritmo di tipo *known plaintext*.

L'algoritmo è un complesso schema XOR che agisce sia sulle linee dati che sulle linee indirizzi. Opera su word da 32 bit, ed oltre agli XOR ha la possibilità di permutare i due byte esterni e i due byte interni di ogni word. La chiave consiste di 9 tabelle da 256 byte. Il modo in cui le tabelle vengono usate per lo XOR sui dati è complesso e può essere studiato nel file sorgente indicato in nota; lo XOR sui 24 bit dell'indirizzo è invece più semplice e può essere riassunto nel seguente modo, dove
A_0 = valore contenuto nei bit 0-7 dell'indirizzo
A_1 = valore contenuto nei bit 8-15 dell'indirizzo
A_2 = valore contenuto nei bit 16-23 dell'indirizzo
c = costante che cambia da gioco a gioco
$K_n[x]$ = valore che si trova in posizione x nella tabella n della chiave

1. $A_0 := A_0 \oplus c$
2. $A_1 := A_1 \oplus K_1[A_2]$
3. $A_1 := A_1 \oplus K_2[A_0]$
4. $A_2 := A_2 \oplus K_3[A_0]$
5. $A_2 := A_2 \oplus K_4[A_1]$
6. $A_0 := A_0 \oplus K_5[A_1]$

L'algoritmo opera sui valori modificati nei passi precedenti: ad esempio, l'A_1 usato al passo 5 è il risultato dello XOR al passo 3.

Il reverse engineering dell'algoritmo è iniziato osservando certe regolarità e schemi ripetitivi all'interno delle ROM cifrate, causati dal fatto che normalmente più del 60% dei dati utilizzati per la grafica sono composti da byte con valore 00_{16} o FF_{16}. Si è costruito quindi uno schema XOR che massimizzasse la presenza di tali valori. Alla fine si è verificato che il conteggio dei byte nelle ROM cifrate, elaborate con tale schema, corrispondeva a quello delle ROM in chiaro prese da un *bootleg*; questo confermava che lo schema era corretto e che la parte di cifratura

rimanente agiva solo sugli indirizzi. Sebbene l'algoritmo che agisce sui dati sia, come detto in precedenza, piuttosto complesso, le regolarità individuate hanno rivelato una sua debolezza, vale a dire quella di essere funzione dell'indirizzo nella ROM *cifrata*; se, al contrario, tale algoritmo fosse stato funzione dell'indirizzo nella ROM *in chiaro*, probabilmente il reverse engineering sarebbe stato più difficile, perché gli schemi ripetitivi sarebbero stati nascosti dalla complessa permutazione degli indirizzi.

Una volta eliminata la cifratura sui dati, si è potuta sfruttare una seconda debolezza dell'algoritmo, cioè quella di lavorare su word di ben 32 bit. Con un numero di bit così elevato, c'è un gran numero di valori che appaiono una sola volta nell'intera ROM, cosicché si può costruire una tabella di corrispondenze tra l'indirizzo nella ROM cifrata e quello nella ROM in chiaro, tabella che poi si può studiare per ricostruire l'algoritmo.

5. MAME

Gli Inizi

La storia di MAME inizia ufficialmente il 5 febbraio 1997 con il rilascio della versione 0.1. Quella versione supportava cinque giochi: *Pac-Man*, *Ms Pac-Man*, *Crush Roller*, *Pengo* e *Lady Bug*.

Il mio lavoro sugli emulatori di giochi arcade era iniziato qualche settimana prima. La vigilia di Natale del 1996 capitai per caso su un sito chiamato "The Arcade Emulation Programming Repository". Il suo autore, Allard van der Bas, un lungimirante ragazzo olandese, aveva avuto l'idea di mettere a disposizione di tutti gli utenti non solo gli emulatori, ma anche il loro codice sorgente, in modo da stimolare lo scambio d'idee tra programmatori. Dal sito scaricai il prototipo dell'emulatore per uno dei giochi più popolari di tutti i tempi, *Pac-Man*. Trovai con un po' di fatica le ROM originali, provai a lanciare l'emulatore e… accidenti, non funzionava! Sullo schermo si vedeva solo una serie di caratteri casuali. Deluso, stavo per uscire dal programma e dimenticare tutto, ma all'improvviso arrivò la schermata iniziale del gioco. Funzionava! I caratteri casuali erano solo dei normali controlli sulla RAM che il gioco eseguiva prima di partire.

L'emulazione era piuttosto primitiva, i colori erano completamente sbagliati, mancava il sonoro, e lo schermo era quadrato anziché rettangolare. Incuriosito, decisi di passare qualche ora a cercare di migliorarla. Qualche ora, qualche giorno… presto mi resi conto che c'erano enormi potenzialità di sviluppo. Completata l'emulazione di *Pac-Man*, cominciai ad interessarmi anche ad altri giochi, e dopo un mese decisi che era giunto il momento di riordinare le idee e raccogliere

l'emulazione di tutti i giochi all'interno di un unico programma, dotato di un'architettura flessibile e facilmente espandibile.

Il problema principale con il quale dovetti confrontarmi all'inizio, dato che non avevo alcuna esperienza nel campo della programmazione di emulatori, fu quello di reperire informazioni, ovviamente su Internet. Le informazioni disponibili erano frammentarie, spesso contraddittorie, e sparse per la rete senza organizzazione. Pensai quindi che il principio alla base di MAME avrebbe dovuto essere non tanto la possibilità di giocare a videogiochi del passato, ma fornire una documentazione quanto più possibile esauriente ed accurata dell'hardware sul quale erano basati. Questo principio è enunciato all'inizio della documentazione di MAME:

MAME is strictly a non-profit project. Its main purpose is to be a reference to the inner workings of the emulated arcade machines. This is done for educational purposes and to prevent many historical games from sinking into oblivion once the hardware they run on stops working. Of course to preserve the games, you must also be able to actually play them; you can consider that a nice side effect.
It is not our intention to infringe on any copyrights or patents on the original games. All of MAME's source code is either our own or freely available. To operate, the emulator requires images of the original ROMs from the arcade machines, which must be provided by the user. No portions of the original ROM codes are included in the executable.

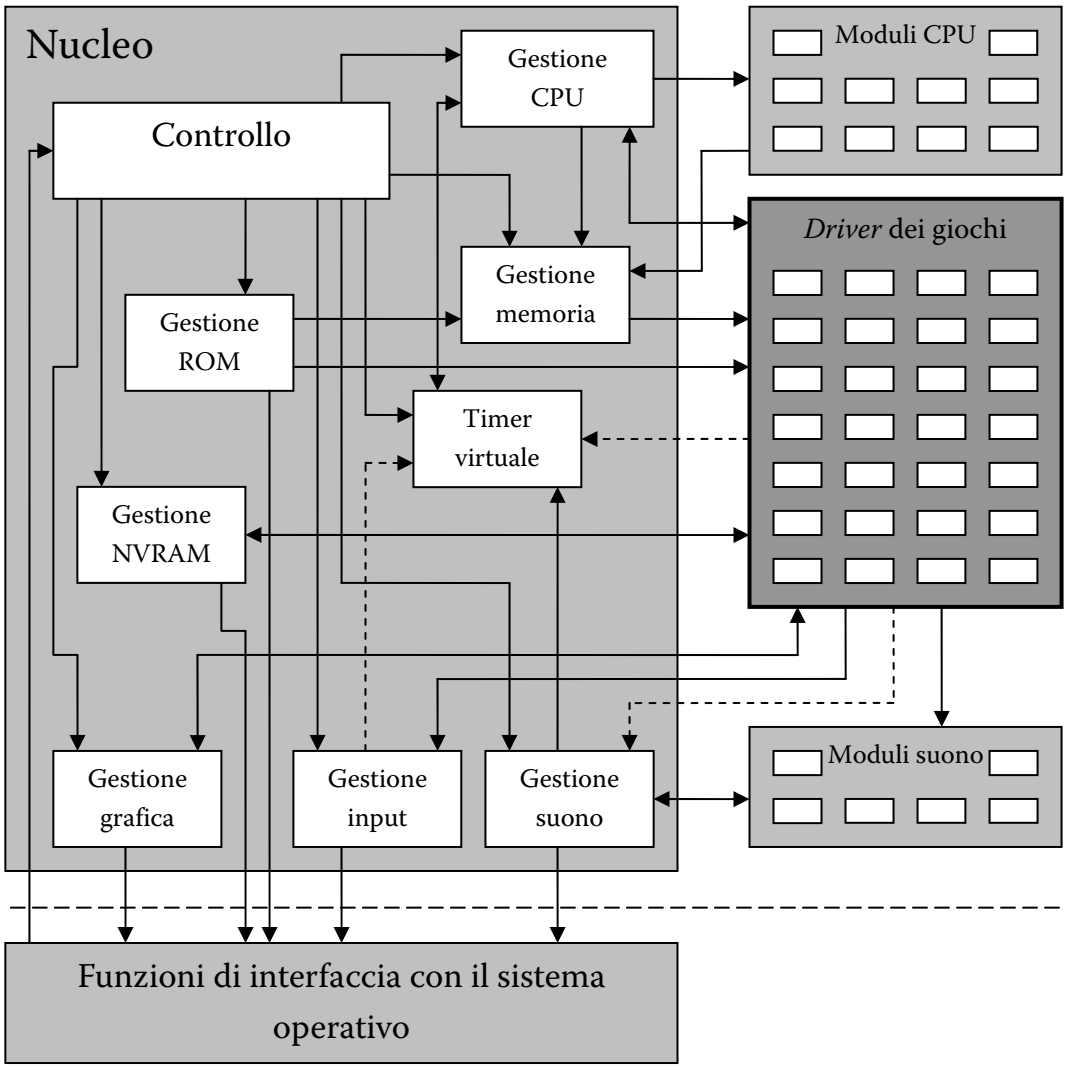
I risultati di questa politica furono ottimi. Dopo un inizio un po' in sordina, anche perché in quel periodo l'emulazione di giochi arcade era una novità e i nuovi emulatori spuntavano come funghi, molti programmatori si interessarono al progetto ed offrirono il loro aiuto. Alcuni di loro hanno continuato a lavorare su MAME fino ad oggi.

Un altro risultato che mi aspettavo di ottenere con la distribuzione del codice sorgente era stimolare la nascita di altri progetti che avrebbero potuto attingere liberamente alle informazioni raccolte da MAME. Questo è accaduto, soprattutto con lo sviluppo di alcuni emulatori focalizzati al miglioramento delle prestazioni su computer poco potenti,

ma con risonanza minore di quanto avevo previsto, semplicemente perché MAME ha avuto talmente tanto successo che le persone interessate all'emulazione hanno trovato più efficace collaborare direttamente a MAME anziché tentare di creare un altro progetto in competizione.

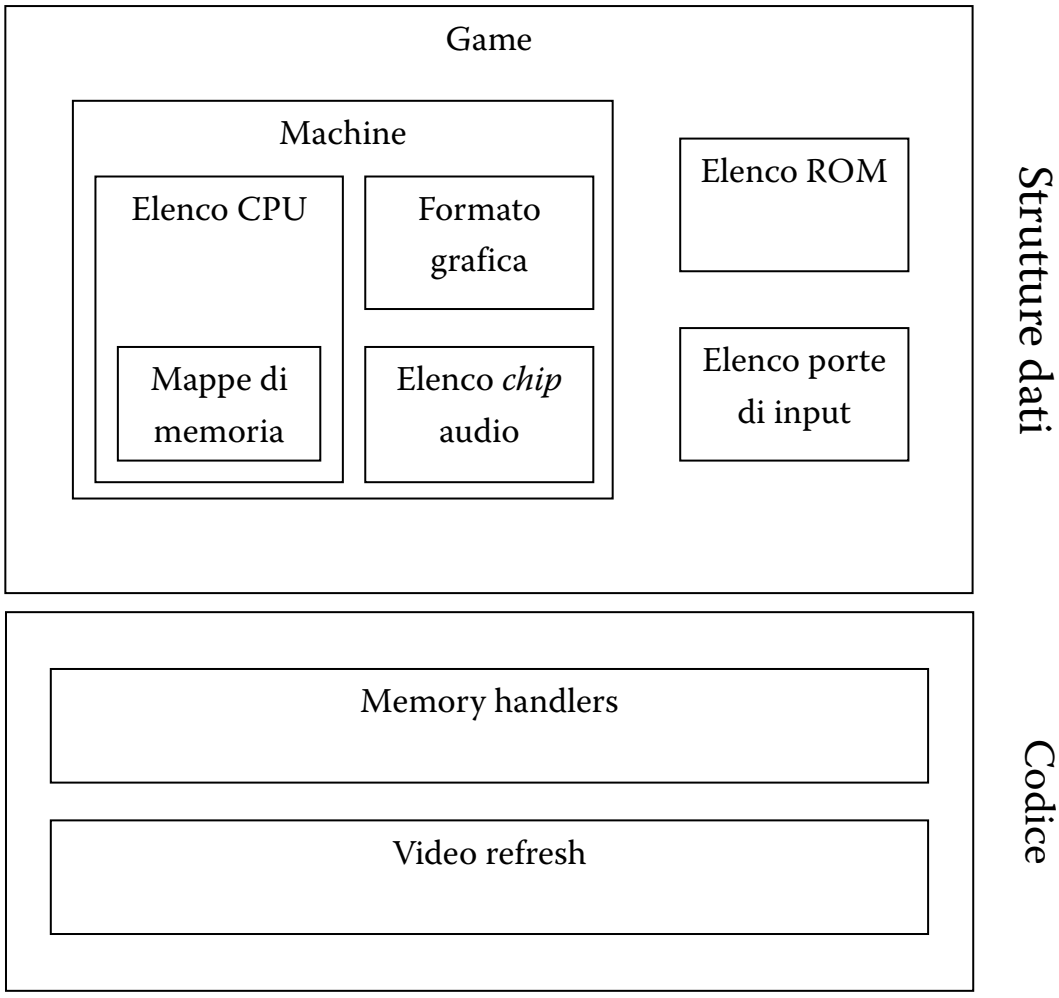
Struttura

La natura del progetto ben si presta ad una struttura modulare come quella schematizzata in figura.



Schema della struttura di MAME

Come si può vedere, i moduli ricalcano le varie parti di un videogioco arcade descritte nel capitolo 3.
Un nucleo centrale dirige le operazioni, gestisce l'interfaccia utente e mette a disposizione dei *driver* un buon numero di funzioni d'uso comune. Il nucleo delega a moduli esterni l'emulazione dei vari tipi di CPU e di *chip* audio supportati.
In pratica il nucleo fornisce un ambiente operativo specializzato nell'emulazione di videogiochi arcade, che i *driver* possono sfruttare con poco codice addizionale. Spesso la maggior parte del contenuto di un *driver* è costituita da strutture dati, gestite direttamente dal nucleo. Il *driver* deve fornire codice solo per alcuni compiti specifici, ad esempio l'aggiornamento del video.



Schema del contenuto di un *driver*

Il linguaggio di programmazione utilizzato è il C, per le sue doti di portabilità e per la sua diffusione. Probabilmente la spiccata modularità avrebbe consentito di applicare in modo proficuo un linguaggio orientato agli oggetti come il C++. La struttura del codice comunque è tale da non rendere scomodo l'uso del C nella scrittura dei *driver*. In ogni caso, la difficoltà nella scrittura di un *driver* risiede tutta nel reverse engineering e non nell'uso di un linguaggio piuttosto che un altro.

La versione principale di MAME girava inizialmente sotto MS-DOS, adesso sotto Windows. Le problematiche riguardanti il trasporto su architetture diverse sono sempre state tenute nella massima considerazione, astraendo tutte le funzioni dipendenti dal sistema operativo. Il codice sorgente è strutturato in modo da riunire in pochi file tutto il codice dipendente dal sistema operativo, che assomma a meno del 2% del totale. Il rimanente 98% del codice non richiede modifiche per essere utilizzato con altri sistemi operativi. La collaborazione dei programmatori Macintosh e Unix/Linux è stata, ed è tuttora, molto importante per garantire la portabilità del progetto. Oltre ai sistemi operativi appena menzionati, ci sono versioni di MAME per piattaforme meno diffuse come OS/2, Amiga e BeOS, che però non vengono più aggiornate. Sono state realizzate anche versioni che girano su computer palmari [18] e perfino su alcuni tipi di macchine fotografiche! <27>, [19].

OPEN SOURCE?

MAME è software gratuito di cui viene distribuito anche il codice sorgente, ma non è né *free software* secondo la definizione della Free Software Foundation <28> (in cui "free" sta per "libero" e non per "gratuito"), né *open source* secondo la definizione della Open Source Initiative <29>.

Data la particolare natura di MAME, all'inizio dello sviluppo sembrò infatti opportuno utilizzare una licenza d'uso più restrittiva di quelle del

free software, in modo da poter mantenere un certo controllo ed assicurarsi che il progetto si evolvesse nella direzione desiderata.
Oltre a questo, personalmente non ho mai gradito la clausola che consente a chiunque di lucrare sopra ad un *free software*: se decido di donare il mio lavoro alla comunità, vorrei che fosse gratuito per tutti. Ma questo, evidentemente, è un punto di vista non condiviso dalla maggior parte degli altri sviluppatori di software *free*.

Come si può immaginare, la licenza d'uso restrittiva adottata da MAME ha scatenato innumerevoli polemiche nella comunità, tra gli integralisti del *free software* e chi invece difendeva le scelte degli sviluppatori di MAME.

Giunti a questo punto dello sviluppo, MAME è ormai maturo e ha conseguito gli scopi che ci eravamo prefissati. Non sembra quindi più necessario utilizzare una licenza restrittiva e molto presto, salvo complicazioni, sarà adottata la GNU General Public License della Free Software Foundation.

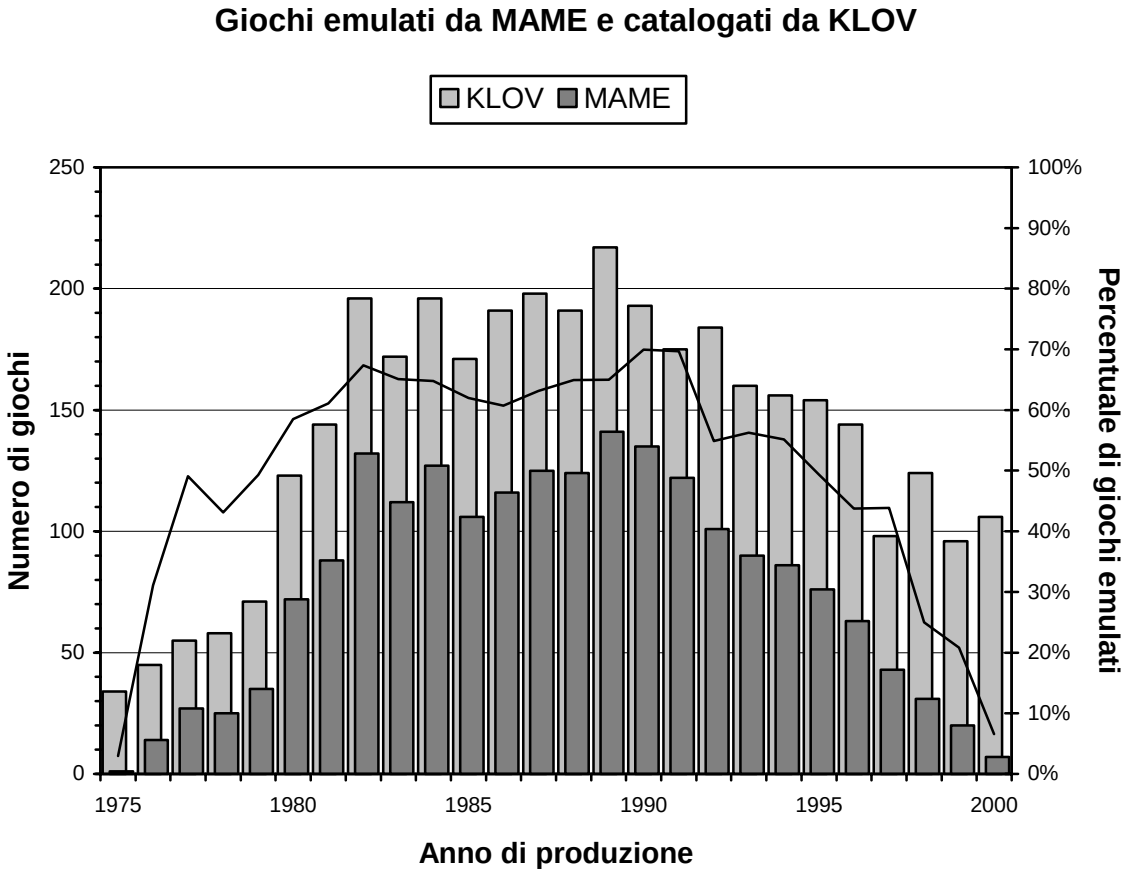
RISULTATI OTTENUTI

Pur con tutti i problemi di velocità che si porta dietro a causa della sua natura generalista, MAME si è imposto come punto di riferimento per gli emulatori di videogiochi arcade, vantando una completezza senza uguali. D'altra parte, come detto in precedenza, la completezza e l'accuratezza sono proprio gli obiettivi principali di MAME, mentre la possibilità di giocare anche su computer poco potenti è un interesse del tutto marginale.

KLOV (Killer List of Video Games, <30>) è il più popolare database di videogiochi arcade disponibile su Internet. Non è completo al cento per cento, ma dà una ragionevole visione d'insieme di quello che è stato il mercato dei videogiochi. KLOV contiene la descrizione di ogni gioco e, spesso, immagini che lo mostrano in funzione. La maggior parte di

queste ultime non sono fotografie del vero gioco, ma fotogrammi dell'emulazione fatta da MAME.

Nel seguente grafico sono posti a confronto, divisi per anno di produzione, i giochi emulati da MAME con quelli presenti su KLOV. I dati non sono perfettamente confrontabili, poiché alcuni giochi emulati da MAME non sono presenti in KLOV, mentre altri compaiono in KLOV più di una volta con nomi diversi.



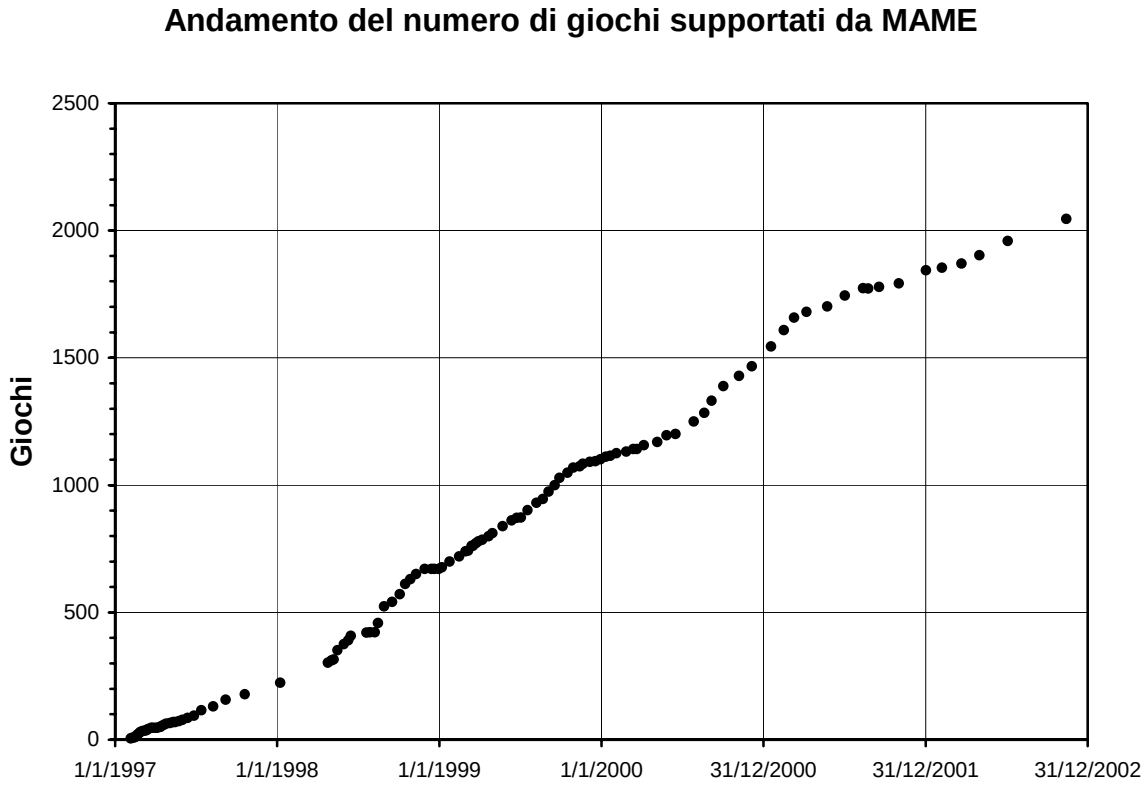
Come si può vedere, per le annate che vanno dal 1980 al 1994, MAME supporta più del 50% dei giochi contenuti nel database. È naturale che la percentuale si abbassi per i giochi più recenti, dato che l'hardware da emulare diventa via via più complesso. Meno ovvio è come mai la percentuale cali anche per i giochi più vecchi: in primo luogo, questo avviene perché più si va indietro nel tempo, più è difficile trovare esemplari funzionanti dei giochi originali; in secondo luogo, perché molti di essi sono esclusi dagli obiettivi di MAME in quanto privi di CPU[25].

[25] cfr. pag. 31

Alcuni collezionisti di videogiochi non vedono di buon occhio gli emulatori, poiché ritengono che il coinvolgimento dato da un emulatore sia solo una scialba replica di quello dato dall'originale. Un esempio di queste idee si trova in [45]. La posizione è senz'altro condivisibile, ma non tiene conto del fatto che è possibile installare MAME all'interno di un cabinato originale, come mostrato in [52]. Ancora più importante, MAME può essere un utile strumento per i collezionisti, sia per verificare la correttezza delle copie di backup delle ROM, sia come aiuto nella ricerca di guasti sulle schede.

CRESCITA

Nel grafico seguente possiamo vedere la crescita di MAME misurata in base al numero di giochi supportati: ogni punto del grafico rappresenta una versione di MAME distribuita pubblicamente.

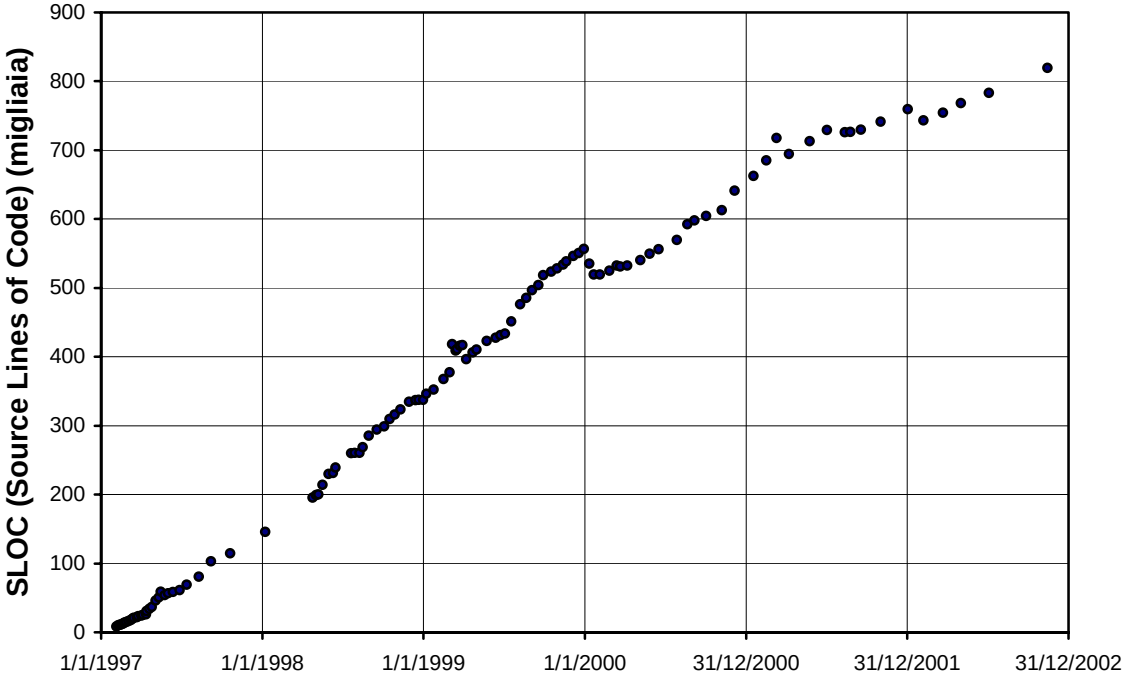


Come si vede, l'andamento è stato abbastanza regolare, con periodi di crescita più o meno rapida a seconda di quante persone stavano

collaborando in quel momento. Alcuni progressi tecnici di MAME hanno dato la possibilità di emulare nuovi tipi di giochi: ad esempio, l'accelerazione che si nota nella seconda metà del 1998 è dovuta in parte al supporto dell'hardware Neo-Geo.

Il grafico seguente mostra invece la crescita del numero di linee di codice[26].

Andamento delle dimensioni del codice sorgente



Si può notare che in alcuni casi il numero di linee di codice è *diminuito* da una versione alla successiva. Questo è avvenuto in seguito a miglioramenti strutturali che hanno diminuito la complessità del codice mantenendone immutate le funzionalità.

Altri indicatori della crescita e della popolarità di MAME si possono ricavare dalla sua diffusione su Internet. Il sito ufficiale di MAME <1> ha registrato più di 27 milioni di visitatori a partire dal 12 maggio 1997. Una ricerca di "MAME" sul più usato motore di ricerca, Google <31>, trova più di 200.000 pagine.

[26] conteggio eseguito con CodeCount 1.0, © 1998 USC-CSE

PROSPETTIVE FUTURE

Il lavoro su MAME è ben lontano dal potersi considerare concluso: un nutrito elenco di giochi ancora da emulare è contenuto in <32>.

Le difficoltà tecniche da superare per aggiungere nuovi giochi diventano via via maggiori, ma anche la nostra esperienza cresce, e piano piano gli ostacoli che sembravano insormontabili diventano superabili. I grafici di crescita mostrano un lieve rallentamento, ma non sembrano ancora indicare segni di cedimento.

Uno degli obiettivi tecnici più prossimi è quello dell'emulazione dei giochi con grafica 3D. Cercheremo nel contempo anche di generalizzare la gestione di questo tipo di grafica, molto diversa dalla grafica 2D a cui siamo abituati. Phil Stroffolino e Aaron Giles hanno fatto ottimi progressi in questo campo e un buon numero di giochi è stato aggiunto nell'ultima versione rilasciata.

L'emulazione delle veloci CPU RISC usate da molti giochi recenti è difficoltosa sui PC odierni, che ancora non riescono ad eseguirla in tempo reale. Tecniche come la *ricompilazione dinamica* potrebbero portare grossi benefici ancora tutti da esplorare.

Proseguirà poi la ricerca, attraverso i collezionisti, di giochi rari che per ora ci sono sfuggiti. Più un gioco è raro, più alto è il rischio che vada perduto per sempre: sfortunatamente, abbiamo avuto prova dei pericoli quando l'unico backup di una versione del gioco *Rafflesia* è stato perso a causa della rottura di un hard disk.

L'obiettivo a cui personalmente tengo di più è di mantenere gli standard di qualità, continuando a fare sempre ogni sforzo per rendere la documentazione che viene raccolta il più possibile accurata, secondo il principio che nulla di quello che è stato fatto è definitivo. A questo proposito, per esempio, nel corso della stesura della tesi, mentre rileggevo le parti di sorgente attinenti agli argomenti che stavo trattando, ho individuato e corretto alcuni errori che vi si annidavano da anni: questa è stata una delle cose che mi hanno fatto più piacere.

6. STATO DELL'ARTE

PRIMI PASSI

Gli emulatori di videogiochi arcade hanno cominciato a diffondersi solo da qualche anno. In precedenza erano stati realizzati emulatori di altre macchine, come le console e gli home computer, ma la potenza di calcolo dei PC non era ancora sufficiente per gestire in tempo reale un videogioco arcade.

Negli ultimi anni, grazie all'opera di centinaia di appassionati, le cose sono cambiate radicalmente, tanto che qualcuno ha sentito l'esigenza di catalogare tutto ciò che è stato realizzato al riguardo <33>.

Il primo vero emulatore di videogiochi da bar per PC, *Williams Arcade Classics* di GT Interactive <34>, fu commercializzato alla fine del 1995. I più noti giochi supportati erano *Joust, Robotron, Defender* e *Sinistar*, tutti della Williams. Non stupisce che fossero stati scelti proprio questi titoli: si tratta, infatti, di alcuni dei pochi giochi con grafica bitmap, la più simile a quella di un PC[27]. Per poter funzionare a piena velocità su una macchina con processore 486 a 33MHz, il sonoro non era emulato in tempo reale ma utilizzava campioni precalcolati.

Sulla scia del suddetto titolo commerciale, nel 1996 si cominciarono a diffondere anche alcuni programmi gratuiti, liberamente prelevabili da Internet. Tra i rappresentanti più significativi di quel periodo possiamo citare *Sparcade* di Dave Spicer <35>, che tra gli altri emulava *Pac-Man*,

[27] cfr. pag. 53

ed *EMU* di Neil Bradley <36>, specializzato nei giochi in grafica vettoriale della Atari, come *Asteroids*. In quel periodo supportare più di un gioco, come nei due casi appena menzionati, era un'eccezione: la maggior parte degli emulatori ne gestiva solo uno.

Crescita

L'arrivo di MAME, nel 1997, rappresentò una svolta. Non conveniva più ricominciare ogni volta da zero per creare un programma a sé stante, poiché era molto più semplice e rapido aggiungere i giochi direttamente a MAME. Fortunatamente, però, MAME non monopolizzò la scena: molti dei progressi fatti negli anni seguenti si devono ad altri emulatori, i più importanti dei quali sono elencati di seguito.

Callus <37>: il primo a supportare i giochi CPS-1 della Capcom, è considerato tuttora uno dei migliori emulatori mai scritti.

NeoRAGE <38>: il primo emulatore del sistema Neo-Geo, ancora utilizzato da molti nella sua versione per Windows.

M72: il primo a riprodurre il sistema M72 della Irem, col famosissimo gioco *R-Type*.

System 16 <39>: il primo ad iniziare il reverse engineering del System 16 della Sega, un sistema che, con tutte le sue varianti, non è ancora stato emulato perfettamente.

Cinematronics Emulator <40>: come suggerisce il nome, il primo a documentare i giochi in grafica vettoriale della Cinematronics.

Shark <41>: il primo ad affrontare i giochi della Toaplan.

Nessuno di questi emulatori è stato sviluppato ulteriormente dagli autori. I giochi da essi supportati sono stati pian piano aggiunti a MAME, spesso con un'emulazione più accurata, anche se più lenta. Molti autori di emulatori "concorrenti" ci hanno offerto il loro aiuto o si sono direttamente uniti al gruppo di sviluppo di MAME, abbandonando i progetti individuali.

SITUAZIONE ATTUALE

MAME, unico emulatore ad essere stato sviluppato senza sosta negli ultimi anni, non può non essere un punto di riferimento. Gli altri emulatori attualmente in fase di sviluppo si possono raccogliere in due categorie: quelli che intendono fare meglio ciò che MAME già fa, e quelli il cui scopo è fare ciò che MAME ancora non fa. Vediamo i più importanti.

RAINE <42>: dopo MAME, è l'emulatore il cui sviluppo dura da più tempo. Nato come emulatore di *Rainbow Islands*, estese poi la sua compatibilità ad altri giochi basati sul Motorola 68000, principalmente della Taito. Attualmente supporta circa 400 giochi; adesso sono quasi tutti supportati anche da MAME, ma in molti casi RAINE è stato il primo ad emularli. Una particolarità di RAINE è che, dopo aver avuto inizio come un progetto closed source, è in seguito diventato open source, forse anche grazie alla spinta di MAME. Tra gli sviluppatori di RAINE e MAME c'è una proficua collaborazione che ha consentito di migliorare entrambi gli emulatori.

Nebula <43> e ***Kawaks*** <44>: entrambi gestiscono in modo più efficiente di MAME alcuni sistemi molto popolari tra i frequentatori di sale giochi: CPS-1 e CPS-2 della Capcom e Neo-Geo. Lo sviluppo di *Kawaks* sembra essere rallentato, mentre l'autore di *Nebula* sta lavorando al supporto di giochi non ancora emulati da MAME.

Final Burn <45>: oltre ai "soliti" CPS-1 e CPS-2, questo emulatore supporta diversi giochi della Sega non ancora inclusi in MAME, come *Power Drift* e *Rail Chase*.

DAPHNE <46>: è uno dei progetti più interessanti, perché si occupa esclusivamente dei giochi con laserdisc, macchine che conobbero un fugace momento di popolarità a metà degli anni '80. La grafica di questi giochi era costituita per la maggior parte da veri e propri film memorizzati su laserdisc, controllati dal programma sotto la guida del giocatore. DAPHNE consente di rivedere questi giochi sia dal laserdisc originale, tramite un lettore collegato al PC, sia utilizzando una copia del film memorizzata sull'hard disk.

Modeler <47>: emulatore del potente Sega System 32, che speriamo di poter aggiungere presto anche a MAME.

Zinc <48>: è il progetto più avanzato di emulazione di giochi arcade con grafica 3D, in questo caso basata sull'hardware della Sony Playstation.

È molto positivo il fatto che la maggior parte degli autori di emulatori siano animati da spirito di collaborazione e ben disposti ad aiutarsi l'un l'altro per favorire progressi più rapidi nella ricerca. Mi piace credere che l'esempio dato da MAME sia stato importante per indirizzare verso questo modo di pensare.

GLOSSARIO

ADPCM Adaptive Differential Pulse Code Modulation. Metodo di compressione di segnali audio che utilizza la differenza tra campioni consecutivi.

arcade Abbreviazione di *amusement arcade*, sala giochi.

bit Da BInary digiT, cifra binaria che può essere 0 o 1.

bootleg Versione illegale di un gioco, spesso modificata per eliminare il nome del produttore o aggirare protezioni.

bug Errore di programmazione.

byte Unità d'informazione di 8 bit.

CAD Computer Aided Design. Tecnica utilizzata per progettare con l'aiuto di un computer.

checksum Somma di controllo. Valore utilizzato per verificare l'integrità dei dati.

chip Circuito integrato, cioè circuito elettronico miniaturizzato contenuto interamente all'interno di un singolo involucro.

clock Generatore d'impulsi che controllano la temporizzazione di un circuito elettronico.

codice sorgente Programma scritto in linguaggio sorgente che deve essere compilato per diventare eseguibile.

compilazione Traduzione da linguaggio sorgente a programma eseguibile.

console Videogioco casalingo, utilizzato collegandolo al televisore.

CPU	Central Processing Unit, unità centrale di elaborazione che esegue le istruzioni del programma.
crittanalisi	Scienza che studia come decifrare messaggi crittografati senza averne l'autorizzazione.
crittografia	Scienza che studia come codificare i dati per impedirne l'accesso senza autorizzazione.
custom	Circuito integrato o altro oggetto realizzato su ordinazione e non disponibile in commercio.
database	Insieme di dati organizzati.
debugger	Programma usato per cercare errori (bug) nel software.
decompilazione	Operazione inversa della compilazione.
DIP switch	Gruppo di interruttori contenuti in un Dual In-line Package, collegati direttamente al circuito stampato.
EEPROM	Electrically Erasable Programmable Read-Only Memory.
EPROM	Erasable Programmable Read-Only Memory.
file	Insieme di dati immagazzinati su una memoria di massa.
flag	Variabile usata per memorizzare una condizione logica.
girare	Termine gergale per dire "eseguire un programma".
hard disk	Unità di memoria di massa costituita da uno o più dischi magnetici rigidi.
hardware	La parte fisica di un computer.
home computer	Computer domestico. Categoria di computer, con prestazioni ridotte e basso costo, popolari negli anni '80.
I/O	Input/Output, ingresso/uscita.
icona	Immagine che indica un file in un sistema operativo dotato di interfaccia grafica.
joystick	Periferica di input costituita da una leva che può essere spostata nelle quattro direzioni.

kB	Kilobyte, 1 kB = 2^{10} byte.
laserdisc	Supporto ottico analogico utilizzato per la memorizzazione di film.
LCD	Liquid Crystal Display.
link	Collegamento ad una pagina Internet.
MB	Megabyte, 1 MB = 2^{20} byte.
MCU	Micro Controller Unit.
OCR	Optical Character Recognition, programma per il riconoscimento automatico del testo.
opcode	Operation Code, codice d'istruzione di un microprocessore.
PAL	Programmable Array Logic.
PC	Personal Computer.
pixel	Punto sullo schermo.
PROM	Programmable Read-Only Memory.
RAM	Random Access Memory.
RNG	Random Number Generator.
ROM	Read-Only Memory.
sistema operativo	Insieme di programmi necessari per la gestione di un computer e delle sue periferiche.
software	Insieme dei programmi eseguibili su un computer, contrapposto a hardware.
spinner	Periferica di input costituita da una manopola rotante intorno ad un asse.
stringa	Sequenza di caratteri.
trackball	Periferica di input costituita da una sfera che può ruotare su se stessa.
trojan horse	Programma utilizzato per introdursi in un sistema ed estrarne informazioni.
word	Parola indirizzabile in un solo accesso, può essere costituita da uno o più byte.

BIBLIOGRAFIA

Introduzione

[1] J. C. Herz, *Il popolo del joystick*. Feltrinelli, Milano, 1997.

[2] L. Herman, *Phoenix: The Fall & Rise of Videogames*. Rolenta Press, Springfield, 1997.

[3] S. L. Kent, *The Ultimate History of Video Games*. Prima Publishing, Rocklin, 2001.

Capitolo 1

[4] E. J. Chikofsky e J. H. Cross II, *Reverse Engineering and Design Recovery: A Taxonomy*. "IEEE Software", vol. 7, n. 1, gennaio 1990, p. 13-17.

[5] S. Rugaber, *Program Comprehension for Reverse Engineering*. In: AAAI Workshop on AI and Automated Program Understanding, San Jose, Ca., luglio 1992, p. 106-110.

Capitolo 2

[6] A. M. Turing, *On computable numbers with an application to the Entscheidungsproblem*. In: Proceedings of the London Mathematical Society, ser. 2, vol. 42, 1937, p. 230-265.

[7] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass., 1979.

Capitolo 3

[8] Z. Moore, *The Cinematronics CPU Programmer's Reference Guide*. 2000.
http://zonn.com/Cinematronics/files/CineRef.pdf

[9] G. Kane, D. Hawkins, L. Leventhal, *Assembler per 68000*. Jackson, Milano, 1986.

[10] R. Anderson e M. Kuhn, *Tamper Resistance - a Cautionary Note*. In: Proceedings of the Second USENIX Workshop on Electronic Commerce, Oakland, Ca., novembre 1996, p. 1-11.
http://www.cl.cam.ac.uk/users/rja14/tamper.html

[11] M. Barr, *Memory Types*. "Embedded Systems Programming", maggio 2001, p. 103-104.
http://www.netrino.com/Publications/Glossary/MemoryTypes.html

[12] G. E. Moore, *Cramming More Components onto Integrated Circuits*. "Electronics", vol. 38, n. 8, 19 aprile 1965, p. 114-117.
http://www.intel.com/research/silicon/moorespaper.pdf

[13] G. E. Moore, *Progress in Digital Integrated Electronics*. In: IEEE International Electron Devices Meeting Technical Digest, dicembre 1975, p. 11-13.

[14] J. L. Massey, *Shift-Register Synthesis and BCH Decoding*. "IEEE Transactions on Information Theory", vol. IT-15, n. 1, gennaio 1969, p. 122-127.

Capitolo 4

[15] B. Schneier, *Applied Cryptography, Second Edition*. Wiley, New York, 1996.

[16] C. Giustozzi, A. Monti, E. Zimuel, *Segreti spie codi[ci]frati*. Apogeo, Milano, 1999.

[17] Robert L. Chapman, *The Dictionary of American Slang.* Pan Books, Londra, 1988.

Capitolo 5

[18] P. van Sebille, *EMame: a MAME port to EPOC Release 5 and Symbian platform v 6.0 (Quartz)*. 2001.
http://www.symbian.com/developer/techlib/papers/mame/mamedream.html

[19] D. Cohen, *I Am Not Just a Camera*. "Wired", vol. 8, n. 5, maggio 2000.
http://www.wired.com/wired/archive/8.05/streetcred.html?pg=2

[20] A. R. Meo, *Software libero e "open source"*. "Mondo digitale", n. 2, giugno 2002.
http://www.aicanet.it/rivista/numero_due/Meo.pdf

MAME nei media

Molti degli articoli seguenti contengono imprecisioni, anche grossolane.

[21] S. Campbell, *Nicola Salmoria's Multi Arcade Machine Emulator illustrates perfectly the dynamism of the PC arcade emulation scene*. "Edge", n. 45, febbraio 1997.
http://dialspace.dial.pipex.com/town/estate/dh69/wos/world/edge/mame.htm

[22] M. Triulzi, *C'era una volta mister Pacman*. "Corriere della Sera", 21 giugno 1997.

[23] T. Toniutti, *Intervista con Nicola Salmoria*. "ZETA", n. 29, settembre 1997, p. 46-47.

[24] M. Alberico, *Videogiochi MAME*. "MediaMente" (trasmissione televisiva), 24 ottobre 1997.
http://www.mediamente.rai.it/home/tv2rete/mm9798/97102024/i971024.htm

[25] J. Prisco, *Multi Arcade Machine Emulator*. "PC Force", n. 1, marzo 1998, p. 14-16.

[26] J. C. Herz, *In Software Sleight of Hand, Video Ghosts Rise*. "The New York Times", 5 marzo 1998.
http://query.nytimes.com/search/abstract?res=F20713FA38550C768CDDAA0894D0494D81

[27] J. M. Moran, *Software Emulating Classic Video Games*. "The Hartford Courant", 2 aprile 1998.

[28] J. Borland, *Anti-Piracy Forces Target Arcade Classics*. "TechWeb News", 22 aprile 1998.
http://www.techweb.com/wire/story/TWB19980422S0010

[29] M. Stroh, *Blasts from the past: Digital archaeologists hoping to restore classic games*. "The Sacramento Bee", 29 aprile 1998.

[30] *Past Blasters*. "Entertainment Weekly", n. 433, 22 maggio 1998, p. 76.

[31] *First among emus*. "Edge" UK Edition, n. 61, agosto 1998, p. 74-79.

[32] G. Zanetti, *X-MAME: anche i pinguini si divertono*. "MCmicrocomputer", n. 189, novembre 1998, p. 276-279.

[33] M. Baccan, *Gli Emulatori*. "DEV", n. 62, aprile 1999, p. 30-34.

[34] D. McCandless, *History in the Taking*. "Wired", vol. 7, n. 5, maggio 1999, p. 64.
http://www.wired.com/wired/archive/7.05/mustread.html?pg=9

[35] A. S. Bub, *Emulation Conflagration - Game Fans versus the Music, Arcade and Console Companies*. "Voodoo: The Official 3DFx Magazine", vol. 2, n. 2, estate 1999.

[36] M. Lella, *Evviva la mame*. "Il Giornale", 21 febbraio 2000.

[37] J. Kroll, *GAME FOCUS-->MAME Emulation of 1951 games*. "Linux Journal", n. 71, marzo 2000, p. 14.
http://www.linuxjournal.com/article.php?sid=3836

[38] A. Ihnatko, *Penny arcade on your PC*. "Chicago Sun-Times", 4 aprile 2000.

[39] A. Lawendel, *Il pc sembra la PlayStation*. "Corriere della Sera", 12 giugno 2000.

[40] K. Poulsen, *The Arcade Underground*. "SecurityFocus Online", 10 luglio 2000.
http://online.securityfocus.com/news/57

[41] G. Bajo, *Emulare con MAME*. "ioProgrammo", n. 40, ottobre 2000, p. 30-33.
http://www.itportal.it/developer/cpp/emumame/

[42] M. Camisasca e S. Soletta, *A volte ritornano*. "Computer idea", n. 21, 29 novembre - 12 dicembre 2000, p. 10-11.

[43] G. Bajo, *MAME o non m'ame?*. "ioProgrammo", n. 42, dicembre 2000, p. 34-37.
http://www.itportal.it/developer/cpp/mame/

[44] A. Ihnatko, *The Game Room*. "Macworld", marzo 2001.
http://www.macworld.com/2001/03/opinion/gameroom.html

[45] J. Sellers, *Remembrance of Things Blast*. "Slate", 27 marzo 2001.
http://slate.msn.com/default.aspx?id=103013

[46] W. O'Neal, *Keep Your Friggin' Gameboy! : Could MameCE be WindowsCE's killer app?*. "Computer Gaming World", 31 marzo 2001.
http://cma.zdnet.com/texis/techinfobase/techinfobase/+kwq_qosXsWWKWs/cdisplay.html

[47] G. Mola, *Mame mania, rivive sul pc il mito dei giochi da bar*. "la Repubblica.it", 2 maggio 2001.
http://www.repubblica.it/online/tecnologie_internet/mame/mame/mame.html

[48] F. Santucci, *Shutdown*. "DEV.", n. 86, giugno 2001, p. 113.
http://online.infomedia.it/riviste/dev/86/articolo16/index.htm

[49] K. Kleiner, *I ♥ Space Invaders*. "New Scientist", vol. 172, n. 2313, 20 ottobre 2001, p. 46-48.

[50] M. Saltzman, *Keys to the Kingdom*. "Electronic Gaming Monthly", ottobre 2001, p. 194-200.

[51] F. Tarassi, *Gli antenati dei videogiochi tornano a vivere su Internet*. "la Repubblica", 27 ottobre 2001.

[52] P. Besser, *Arcade @Home - LA SALA GIOCHI A CASA TUA!*. "PC Action", n. 105, novembre 2001, p. 112-117.
http://www.paolobesser.it/arcade/articolo.htm

[53] B. King, *Pac-Man's Trek From Arcade to PC*. "Wired News", 26 gennaio 2002.
http://www.wired.com/news/games/0,2101,49969,00.html

[54] L. Gambetta, *XMAME: la sala giochi nel PC*. "Linux Pratico", n. 5, marzo/aprile 2002, p. 4-9.

[55] G. Moro, *Xmame: centinaia di giochi a portata di mouse*. "Linux Magazine", n. 21, luglio/agosto 2002, p. 49-51.

[56] *The MAME Game*. "Edge", n. 115, ottobre 2002, p. 76-83.

[57] P. Faranda, *«Così salvo dall'estinzione quei vecchi game da bar»*. "Corriere della Sera" - Speciale tecnologie, 23 ottobre 2002.

[58] L. Valdesi, *«Così ho salvato I video giochi»*. "La Nazione Siena", 24 ottobre 2002.
http://lanazione.quotidiano.net/chan/12/2:3790794:/2002/10/24

LINKS

Introduzione e ringraziamenti

<1> MAME - The official Multiple Arcade Machine Emulator site
http://www.mame.net/

<2> MAMEWorld - The largest MAME resource on the net!
http://www.mameworld.net/

<3> Video Arcade Preservation Society
http://www.vaps.org/

<4> Videotopia
http://www.videotopia.com/

<5> The Arcade Flyer Archive
http://www.arcadeflyers.com/

<6> Aaron's Home Page
http://www.aarongiles.com/

<7> Haze's MAME Page
http://haze.mame.net/

<8> MAME Testers
http://www.mameworld.net/mametesters/

Capitolo 1

<9> Chilling Effects FAQ about Reverse Engineering
http://www.chillingeffects.org/reverse/faq.cgi

<10> Comune di Castelfidardo – storia della fisarmonica
http://www.comune.castelfidardo.an.it/Visitatori/Fisarmonica/storia_fisa.htm

<11> Digi.Lab CAD/CAM/Reverse Engineering
http://www.digilab.it/reverse/reverse.htm

<12> Tecniche di Reverse Engineering per le calzature
http://www.microsystem.it/recalz.asp

<13> Open Directory - Kids and Teens: Sports and Hobbies: Toys: Reverse Engineering
http://dmoz.org/Kids_and_Teens/Sports_and_Hobbies/Toys/Reverse_Engineering

<14> JP1 Interface
http://www.hifi-remote.com/ jp1/index.shtml

Capitolo 2

<15> Garzanti linguistica
http://www.garzantilinguistica.it

<16> _MADrigal_'s handhelds simulators
http://madrigal.retrogames.com/

Capitolo 3

<17> 68000 Undocumented Behavior Notes
http://dynarec.com/~bart/files/68knotes.txt

<18> Z80 Undocumented Features
http://www.greew.freeserve.co.uk/Z80Undoc.html

<19> 6502 Undocumented Opcodes
http://members.chello.nl/taf.offenga/illopc31.txt

<20> University of Southern Mississippi
School of Polymers and High Performance Materials
Resine epossidiche
http://www.psrc.usm.edu/italian/epoxy.htm

<21> How to crack a Pacman Plus!
http://www.multigame.com/pacplus.html

<22> Crack PIC
http://www.piclist.com/techref/microchip/crackpic.htm

<23> HanaHo Games, Inc.
http://www.hanaho.com/

<24> SlikStick – The Worlds Best Arcade Controller
http://www.slikstik.com/

<25> XGAMING, Manufacturer of High End Gaming Accessories
http://www.x-arcade.com/

Capitolo 4

<26> The Dead Battery Society
http://www.arcadecollecting.com/dead/dead.html

Capitolo 5

<27> MAME for Digita Enabled Cameras
http://digita.mame.net/

<28> GNU's Not Unix! - the GNU Project and the Free Software Foundation (FSF)
http://www.gnu.org/

<29> Open Source Initiative OSI
http://www.opensource.org/

<30> The Killer List of Videogames
http://www.klov.com

<31> Google
http://www.google.com

<32> unMAMEd arcade games
http://unmamed.mame.net/

Capitolo 6

<33> CAESAR: Catalogue of Arcade Emulation Software - the Absolute Reference
http://caesar.logiqx.com/

<34> Digital Eclipse Software, Inc. – Williams Arcade Classics
http://www.digitaleclipse.com/live/main/main.php?v=pr&id=53

<35> "Sparcade!", aka Dave's Arcade Emulator
http://www.sparcade.freeserve.co.uk/

<36> EMU Homepage
http://www.synthcom.com/~emu/

<37> Official Bloodlust Software Callus Page
http://bloodlust.zophar.net/Callus/callus.html

<38> R.A.G.E: Homepage
http://home5.swipnet.se/~w-50884/emulator/rage.htm

<39> The System 16 Arcade Emulator!
http://www.system16.com/emu-s16.html

<40> Cinematronics Emulator
http://zonn.com/Cinematronics/emu.htm

<41> The most official Shark distribution site
http://www.c64.org/~magnus/shark.html

<42> RAINE (680x0 Arcade Emulation)
http://www.rainemu.com/

<43> Nebula
http://nebula.emulatronia.com/

<44> Kawaks Saikyo Dojo
http://kawaks.retrogames.com/

<45> Final Burn
http://www.finalburn.com

<46> DAPHNE – Laserdisc Arcade Game Emulator
http://daphne.rulecity.com/

<47> Modeler
http://www.emuhype.com/index.phtml?s=modeler&ss=index

<48> Zinc
http://www.emuhype.com/index.phtml?s=zinc&ss=index

INDICE DELLE FIGURE

INDICE DEI GRAFICI

INDICE ANALITICO

S

T

U

V

W

X

Y

Z