LE SCIENZE

edizione italiana di

SCIENTIFIC AMERICAN

RIVISTA MENSILE. SPEDIZIONE IN ABBONAMENTO POSTALE AL GRUPPO III/70

Lire Tremila
Maggio 1985
Numero 201

Il salto della megattera

(RI)CREAZIONI AL CALCOLATORE

di A. K. Dewdney

Un bestiario di virus, bachi e altre insidie per la memoria dei calcolatori nella Guerra dei nuclei

Quando nel luglio 1984 uscì il mio articolo sulla Guerra dei nuclei, non mi rendevo conto di quanto serio fosse l'argomento affrontato. La mia descrizione di programmi in linguaggio macchina che si aggirano nella memoria di un calcolatore cercando di distruggersi l'un l'altro ha avuto notevole risonanza. Secondo i resoconti di molti lettori, abbondano gli esempi di bachi, virus e altre creature software annidate in tutti i possibili ambienti di elaborazione. Certe possibilità sono così orribili che esito a riportarle.

Il romanzo francese di spionaggio *Softwar: La Guerre Douce* presenta un'immaginaria situazione geopolitica di questo genere. Gli autori, Thierry Breton e Denis Beneich, imbastiscono un agghiacciante racconto a proposito dell'acquisto, da parte dell'Unione Sovietica, di un supercalcolatore americano. Invece di bloccare la vendita, le autorità americane acconsentono alla transazione mostrando una studiata riluttanza: il calcolatore, infatti, è stato segretamente programmato con una «bomba software». Ufficialmente acquistata per le previsioni meteorologiche sul vasto territorio dell'Unione Sovietica, la macchina, o meglio il suo software, contiene un «grilletto» nascosto: appena il Servizio meteorologico nazionale degli Stati Uniti comunica il rilevamento di una certa temperatura a St. Thomas, nelle Virgin Islands, il programma procede a sovvertire e distruggere tutti i pezzi di software che riesce a trovare nella rete sovietica. Se è vero che sceneggiature di questo genere rappresentano possibilità reali, sono tentato di dire: «Se guerra [*war*] deve essere, che sia almeno dolce [*soft*]». D'altra parte, un dubbio mi viene dalla possibilità di un incidente dovuto al collegamento stretto fra software militare e sistemi di controllo delle armi.

Prima di passare a descrivere le esperienze avute da vari lettori con programmi ostili, vale la pena di riassumere le caratteristiche principali della Guerra dei nuclei per coloro che non avessero letto l'articolo del luglio 1984.

Due giocatori scrivono ciascuno un programma in un linguaggio di basso livello chiamato REDCODE. I programmi vengono posti in una vasta arena circolare che chiamiamo Nucleo: in realtà nient'altro che una matrice di migliaia di locazioni, in cui l'ultimo indirizzo è contiguo al primo. Ogni istruzione del programma da battaglia occupa una locazione nel Nucleo. Il programma esecutivo MARS (acronimo per *Memory Array Redcode Simulator*, ovvero «simulatore di Redcode nella matrice di memoria») fa girare i programmi da battaglia eseguendo alternativamente un'istruzione dell'uno e una dell'altro, come un semplice sistema a partizione di tempo: i due programmi si attaccano e, a turno, cercano di evitare danni o di riparare quelli subiti. Una semplice modalità d'attacco può essere eseguita per mezzo di istruzioni MOV. Per esempio

MOV # 0 1000

fa sì che il numero 0 sia posto nella locazione il cui indirizzo si trova 1000 locazioni al di là di questa istruzione, cancellando il precedente contenuto di quella locazione. Nel caso lo 0 venisse posto su un'istruzione dell'avversario, anch'essa sarebbe tolta di mezzo e il programma non sarebbe più eseguibile: l'avversario avrebbe perso il gioco.

Dato che nessun calcolatore, personale o *mainframe*, è dotato all'origine di REDCODE e di un'adeguata matrice da battaglia, queste caratteristiche devono essere simulate. È ancora disponibile la traccia per la stesura di un programma di simulazione (può essere richiesta ancora a «Le Scienze», rubrica «(Ri)creazioni al calcolatore», via del Lauro 14, 20131 Milano, inviando 2000 lire, anche in francobolli, per le spese di fotocopia e di spedizione). L'anno scorso, parecchie centinaia di lettori hanno acquistato questa traccia e molti fra loro hanno scritto programmi di gioco per la Guerra dei nuclei.

Ispirandosi a un articolo di L. S. Penrose sui meccanismi che si autoriproducono, apparso su «Scientific American» nel giugno 1959, Frederick G. Stahl di Chesterfield, Missouri, ha creato un universo lineare in miniatura in cui umili creature vivono, si muovono e (in un certo senso) compiono il proprio destino. Scrive Stahl:

«Come nella Guerra dei nuclei, ho

ISTRUZIONE	MNEMONICA	CODICE	ARGOMENTI		SPIEGAZIONE
Sposta	MOV	1	A	B	Sposta il contenuto dell'indirizzo *A* all'indirizzo *B*
Somma	ADD	2	A	B	Somma i contenuti dell'indirizzo *A* e dell'indirizzo *B*
Sottrae	SUB	3	A	B	Sottrae il contenuto dell'indirizzo *A* dall'indirizzo *B*
Salta	JMP	4	A		Trasferisce il controllo all'indirizzo *A*
Salta se zero	JMZ	5	A	B	Trasferisce il controllo all'indirizzo *A* se il contenuto dell'indirizzo *B* è zero
Salta se maggiore	JMG	6	A	B	Trasferisce il controllo all'indirizzo *A* se il contenuto dell'indirizzo *B* è maggiore di zero
Decremento: salta se zero	DJZ	7	A	B	Sottrae 1 dal contenuto dell'indirizzo *B* e trasferisce il controllo all'indirizzo *A* se il contenuto dell'indirizzo *B* diventa zero
Confronta	CMP	8	A	B	Confronta i contenuti degli indirizzi *A* e *B*; se sono diversi, salta l'istruzione successiva
Divide	SPL	9	A		Divide l'esecuzione nell'istruzione successiva e nell'istruzione in *A*
Enunciato di dati	DAT	0		B	Enunciato non eseguibile; *B* è il valore dei dati

Elenco di istruzioni per la Guerra dei nuclei

isolato un segmento lineare chiuso di memoria principale in cui una creatura era simulata dal linguaggio macchina modificato. La macchina era un IBM Tipo 650 con memoria a tamburo. La creatura era programmata per strisciare lungo il suo universo mangiando cibo (parole diverse da zero) e creando un duplicato di se stessa quando era stato accumulato abbastanza cibo. Come nella Guerra dei nuclei, avevo un programma esecutivo che teneva conto di chi era vivo e distribuiva il tempo d'esecuzione tra le creature viventi. Lo chiamavo «la mano sinistra di Dio». Stahl prosegue analizzando la capacità del suo programma di riprodursi e descrivendo un interessante meccanismo di mutazione: un programma potrebbe subire, durante la copiatura, un piccolo numero di cambiamenti casuali nel suo codice. Tuttavia, riferisce Stahl, «abbandonai questa linea di lavoro dopo una sessione di produzioni in cui un mutante sterile uccise e mangiò l'unica creatura feconda dell'universo. Era chiaro che per ottenere qualche risultato interessante sarebbero state necessarie memorie incredibilmente grandi e tempi di elaborazione lunghissimi.»

Una storia analoga riguarda un gioco chiamato Animale, in cui un programma cerca di stabilire a che animale sta pensando un uomo. David D. Clark, del Laboratory for Computer Science del Massachusetts Institute of Technology, scrive che gli impiegati di una certa azienda giocavano con vero ardore ad Animale. Pur non somigliando a un programma di battaglia e nemmeno alle semplici creature di Stahl, Animale aveva la capacità di riprodursi nei corridoi del nucleo sfruttando gli sforzi del programmatore di potenziare una caratteristica chiave del gioco: quando sbaglia nell'indovinare l'animale che il giocatore umano ha in mente, il programma chiede a quest'ultimo di suggerire una domanda che esso potrebbe porre per migliorare le sue prestazioni future. Questa caratteristica, prosegue Clark, portò il programmatore a inventare un trucco per assicurarsi che ognuno avesse sempre la stessa versione di Animale.

«Su un sistema di elaborazione antiquato, privo di una struttura di catalogo condivisa e privo anche di mezzi di protezione, un programmatore inventò un modo molto originale per rendere disponibile il gioco a più utenti. Una versione del gioco esisteva nel catalogo degli archivi di un utente; quando questi giocava, il programma produceva una copia di se stesso in un altro catalogo di archivi. Se quel catalogo conteneva già una copia del gioco, la vecchia versione veniva cancellata, così che il comportamento del gioco cambiava in modo inatteso per il giocatore. Se quel catalogo non conteneva in precedenza una versione di Animale, un altro utente si trovava ad avere a disposizione il gioco.»

Clark ricorda che Animale era un gioco talmente popolare che, alla fine, tutti i cataloghi del sistema dell'azienda ne contenevano una copia. «Quando poi degli impiegati dell'azienda venivano trasferiti ad altre divisioni... portavano con sé anche Animale, che così si diffuse di macchina in macchina all'interno dell'azienda.» La situazione non sarebbe mai diventata seria se non fosse stato che tutte quelle copie del gioco, per altri versi innocue, cominciarono a intasare la memoria su disco. Solo quando qualcuno inventò una versione più «virulenta» del gioco, la situazione poté tornare sotto controllo. Quando la nuova versione di Animale veniva giocata, essa si copiava in altri cataloghi non una ma due volte. Dandogli abbastanza tempo, si pensava, questo programma avrebbe alla fine cancellato tutte le vecchie versioni di Animale. Dopo un anno, al raggiungimento di una data prefissata, in ogni copia del nuovo programma Animale si innescò un nuovo meccanismo. «Invece di replicarsi due volte, ora esso giocava una partita finale, augurava "arrivederci" all'utente e poi si cancellava. Così Animale venne espulso dal sistema.»

Una volta Ruth Lewart di Holmdel, New Jersey, creò un mostro (per così dire) senza neanche scrivere un programma. Mentre lavorava, sul sistema a partizione di tempo della sua azienda, alla preparazione di una versione dimostrativa di un programma didattico, decise di produrre una copia di riserva su un altro sistema a partizione di tempo. Quando il sistema originale cominciò ad apparire lento, riferisce, «inserii il sistema ausiliario, che era molto sensibile, per tre minuti interi, durante i quali non ci fu alcuna risposta e il caos più completo regnava sullo schermo del mio terminale grafico. Nessuno era più in grado di inserirsi o di uscire dal sistema. La conclusione che si poteva trarre era una sola: in qualche modo la colpa era del mio programma! Nonostante il panico, mi resi improvvisamente conto di aver usato una "e" commerciale (&) come carattere separatore di campo del terminale. Ma la & era anche il carattere usato dal sistema per generare un processo di fondo. Alla prima lettura dallo schermo, il calcolatore deve aver intercettato le & indirizzate al terminale e deve aver generato un gran numero di processi, ciascuno dei quali, a sua volta, generò altri processi, e così via all'infinito.» Una telefonata frenetica informò un responsabile di sistema dell'origine del disturbo e il calcolatore *mainframe* venne spento e fatto ripartire. Inutile dire che Lewart cambiò la & in un altro carattere e il suo programma «da allora ha sempre funzionato felicemente».

Anche se i programmi per la Guerra dei nuclei non vengono generati in questo modo, copie aggiuntive possono aumentare le loro capacità di sopravvivenza. Parecchi lettori hanno proposto la realizzazione di tre copie del programma, in modo che la copia in esecuzione possa utilizzare le altre due per stabilire se qualche sua istruzione è sbagliata. Il programma in esecuzione potrebbe sostituire un'istruzione difettosa con una idonea a garantire la sopravvivenza. Un'idea analoga sta alla base di Scavenger (Spazzino), un programma ideato per proteggere da errori gli archivi di una memoria di massa quando si preparano copie di riserva su nastro magnetico. Arthur Hudson, che vive a Newton nel Massachusetts (e lavora per un'altra azienda che non citiamo), scrive: «Chiunque abbia usato spesso il nastro magnetico si è trovato assediato da una forza aliena chiamata Legge delle probabilità composte.» Hudson prosegue citando vari errori connessi con l'uso di nastri e dimostra che, sebbene ogni tipo di errore abbia una probabilità relativamente piccola di prodursi, la probabilità che se ne verifichi almeno uno è

```
1   IF PEEK (104) = 134 GOTO 10
2   POKE 104, 134: POKE 134 * 256,0
3   PRINT CHR$(4) "RUN APPLE WORM"
10  HOME : POKE - 16302,0: POKE - 16304,0: POKE 1023,160
20  FOR I = 0 TO 94: READ D: POKE 1024 + I, D: NEXT I
30  POKE - 16368,0
40  IF PEEK ( - 16384)<128 GOTO 40
50  CALL 1024
100 DATA 160,225,200,185,255,3,153,127,4,192,95,208,245,
    160,18,190,76,4,24,189,128,4,105,128,157,128,4,189,129,
    4,105,0,157,129,4,192,13,208,18,238,23,4,173,23,4
200 DATA 141,151,4,206,31,4,173,31,4,141,159,4,136,208,211,
    173,167,4,72,173,176,4,141,167,4,104,141,176,4,76,128,
    4,7,20,25,28,33,46,55,61,65,68,72,75,4,16,40,43,49,52
```

Un bacò che vive negli Apple

```
DAT     |        |  0   | /puntatore all'indirizzo da proteggere
ADD     |   #1   |  —1  | /incrementa l'indirizzo della protezione
PCT     |  @—2   |      | /protegge l'indirizzo
CMP     |  #102  |  —3  | /se tutte e 102 le istruzioni sono protette...
JMP     |   2    |      | /... allora lascia il ciclo
JMP     |  —4    |      | altrimenti riprendilo
                  :
CORPO DEL PROGRAMMA DI BATTAGLIA PRINCIPALE
                  :
                  :
```

Questo ciclo protegge i combattenti da bombe vaganti

spiacevolmente alta. Poi continua così:

«Niente paura, Scavenger è con voi: se gli affidate un archivio di una memoria di massa, copierà l'archivio su tre nastri magnetici senza seccarvi con banali dettagli di gestione. Anche se il calcolatore cade in errore logico (come avveniva parecchie volte al giorno) di solito non verrà distrutta la registrazione dell'esecuzione; quando il calcolatore viene riattivato, tutti i bachi presenti nella registrazione si metteranno a loro volta in funzione. Alla scrittura di ciascun nastro presiede un compito distinto, sotto il coordinamento di un programma principale.»

Chi possiede un calcolatore Apple dovrebbe guardarsi da uno spregevole programmino chiamato Apple Worm (letteralmente Baco della mela), creato da Jim Hauser e William R. Buckley della California Polytechnic State University a San Luis Obispo. Scritto per l'Apple II in linguaggio di assemblatore del microelaboratore 6502, questa specie di baco si riproduce nel corso di un allegro viaggetto attraverso l'Apple ospite. All'inizio si carica un particolare programma BASIC *(si veda l'illustrazione della pagina precedente),* il quale a sua volta carica il baco nella parte bassa della memoria (quella con indirizzi bassi). Il programma BASIC occupa invece la parte alta della memoria.

«Dato che il Baco viene caricato in una delle aree grafiche della macchina, si può osservare il Baco che inizia il suo attacco a capofitto (o, per meglio dire, a codafitta) nella memoria alta... Una volta che il Baco ha lasciato la finestra grafica... si può aspettare che abbia cancellato la parte alta della memoria (compreso il programma BASIC) e vada a scontrarsi con le ROM di sistema.»

Hauser e Buckley hanno in programma di pubblicare, in un futuro non lontano, una raccolta di bachi. Hanno progettato un Worm Operating System (Sistema Operativo Baco) e hanno perfino scritto un videogioco in cui Baco ricopre uno dei ruoli principali.

Un'altra minaccia software viene proposta da Roberto Cerruti e Marco Morocutti di Brescia. Prendendo spunto dalla traduzione italiana dell'articolo sulla Guerra dei nuclei apparsa su «Le Scienze», hanno pensato a un modo per infestare l'Apple II, non con un baco ma con un virus. Scrive Cerruti:

«Marco pensò di scrivere un programma capace di passare da un calcolatore all'altro, come un virus, e di "infettare" in questo modo altri Apple. Non fummo però in grado di idearlo finché non mi resi conto che il programma doveva infettare i dischetti e usare il calcolatore solo come mezzo per migrare da un disco all'altro. Fu così che il nostro virus cominciò a prendere forma.

«Come si sa, ogni dischetto Apple contiene una copia del DOS (il sistema operativo per dischi), che viene lanciato dal calcolatore non appena riceve l'alimentazione. Il virus era una modificazione di questo DOS, che a ogni operazione di scrittura verificava la propria presenza sul disco e, in caso negativo, modificava il DOS sul disco, copiandosi così su ogni dischetto messo nell'unità di lettura e registrazione dopo la prima accensione. Pensammo che, se avessimo installato un simile DOS su alcuni dischi usati nel maggiore negozio di calcolatori della nostra città, Brescia, in breve avremmo provocato la diffusione di un'epidemia in tutta la città.

«Ma era una vera epidemia, con virus così innocui? No, i nostri virus dovevano essere maligni! Decidemmo, quindi, che dopo 16 cicli di autoriproduzione, contati sul disco stesso, il programma dovesse decidere di reinizializzare il disco subito dopo il lancio. A quel punto era chiara la terribile perversità della nostra idea e decidemmo di non metterla in pratica né di parlarne con alcuno.»

Cerruti e Morocutti sono stati gentili. In un calcolatore personale, il sistema operativo per dischi è l'arbitro ultimo del destino dei programmi, dei dati e di tutto il resto. Nello schema appena descritto, il sistema operativo infetto cancella il disco da cui è stato lanciato e quindi non può più essere caricato se non da un altro disco. Il DOS contagiato potrebbe anche far apparire periodicamente un messaggio irritante del tipo:

IL VOSTRO DISCO
VI SFUGGE?
È ora di rivolgersi al
DOTTOR DOS
disponibile su disco nel
più vicino negozio di calcolatori

L'infezione virale descritta si è già verificata su piccola scala. Richard Skrenta, Jr., studente di una scuola superiore di Pittsburgh, scrisse un programma di questo genere. Invece di cancellare dischi o visualizzare avvisi, questa forma di infezione provocava la comparsa di subdoli errori nel sistema operativo.

«Tutto questo sembra molto infantile», scrive Skrenta, ma «ahimè! Non son più stato capace di liberarmi del mio flagello elettronico. Ha contagiato tutti i miei dischi e i dischi dei miei amici. È riuscito addirittura ad arrivare ai dischi con i programmi per i grafici di funzione del mio professore di matematica.» Skrenta ha inventato un programma per distruggere il virus, ma non si è mai dimostrato efficace quanto il virus stesso.

Quanto detto implica un problema interessante, e sarei privo di fantasia e irresponsabile se non lo ponessi: descrivere, in una pagina o meno, DOTTOR DOS, un programma su disco che in qualche modo sappia soffocare epidemie elettroniche di questo genere. Molti dischi usati da un calcolatore personale contengono copie del suo DOS. Quando viene fatto partire, il calcolatore ottiene dal disco la sua copia del DOS, che rimarrà operante anche quando vengono fatti girare altri dischi, sia pure contenenti anch'essi copie del DOS. Se è infetto, il DOS attivo può alterare le altre copie del DOS o addirittura sostituirle con copie di se stesso. Come ci si può opporre a questa virulenza?

Nella versione iniziale della Guerra dei nuclei, la difficoltà principale consisteva nel fornire al programma di battaglia A i mezzi per proteggersi dai colpi vaganti del programma di battaglia B. Se questo tipo di protezione riusciva a essere più o meno efficace, l'evoluzione del gioco portava al livello successivo, in cui i programmi sarebbero stati costretti a cercarsi l'un l'altro e a sviluppare attacchi concentrati.

Nel tentativo di garantire questa protezione, nell'articolo del luglio 1984 suggerivo l'istruzione

PCT *A*

dove *A* è l'indirizzo relativo (diretto o indiretto) di un'istruzione che deve essere protetta. Un tentativo di cambiare il contenuto di quell'indirizzo sarebbe bloccato da MARS, il sistema supervisore del gioco. Il tentativo successivo, però, avrebbe avuto successo. Mi sembrava che, impiegando un semplice ciclo, un programma di battaglia potesse proteggere tutte le proprie istruzioni da bombe vaganti abbastanza a lungo da riuscire a lanciare una sonda indisturbata verso l'altro programma. La figura di questa pagina mostra in forma schematica un programma ad autoprotezione di questo genere. Il ciclo di protezione è formato da sei istruzioni, quattro delle quali eseguite a ogni passaggio nel ciclo. Così, un

programma di battaglia di n istruzioni (ciclo incluso) richiederebbe circa $4 \times n$ esecuzioni per avere una protezione completa da un singolo colpo. Questo scudo non è una gran protezione contro un programma dwarf che lanci due colpi contro ogni locazione.

Esiste un altro uso di questa istruzione, non previsto nel precedente articolo sulla Guerra dei nuclei. Stephen Peters di Timaru, Nuova Zelanda, e Mark A. Durham di Winston-Salem, North Carolina, l'uno indipendentemente dall'altro, hanno pensato di usare PCT in modo offensivo. Un programma chiamato TRAP-DWARF innalza uno sbarramento di zeri nel solito modo ma poi protegge ogni deposito contro la sovrascrittura. Questo significa che un programma nemico incauto può cadere in una di queste trappole mentre si trascrive in una nuova area. L'istruzione indirizzata alla locazione occupata da uno zero protetto non avrebbe ovviamente effetto su quella locazione. Quando, in seguito, l'esecuzione raggiunge quell'indirizzo, il nuovo programma muore perché 0 non è un'istruzione eseguibile. Varrà forse la pena di includere PCT in qualche futura versione della Guerra dei nuclei, ma per ora la terrò da parte per amore di semplicità, sigillo del progettista del gioco.

Tra le altre idee suggerite dai lettori ci sono la matrice di nuclei bidimensionale proposta da Robert Norton di Madison, Wisconsin, e la regola di limitazione di campo avanzata da William J. Mitchell del dipartimento di matematica della Pennsylvania State University. L'idea di Norton si spiega da sé; la proposta di Mitchell, invece, richiede un approfondimento. Essa consiste nel consentire a ogni programma di battaglia di modificare il contenuto di qualsiasi locazione che non disti più di un certo numero prestabilito di indirizzi. Una regola di questo genere impedisce automaticamente a DWARF di fare danni al di fuori di questo intorno. La regola ha anche molti altri effetti, tra cui quello di sottolineare notevolmente il movimento: in quale altro modo un programma di battaglia potrebbe raggiungere il campo di un avversario? La regola ha molti pregi e spero che qualcuno dei molti lettori che possiedono un sistema per la Guerra dei nuclei le voglia dedicare l'ulteriore approfondimento che merita.

Norton propone anche che, in una battaglia della Guerra dei nuclei, a ogni contendente sia consentita più di una esecuzione. La stessa idea è venuta a molti altri lettori e ho deciso di accettare il suggerimento, così che ora la Guerra dei nuclei assume un carattere di grande apertura che prima mancava.

Il gioco si modifica aggiungendo la seguente istruzione, chiamata scissione, al listato ufficiale della Guerra dei nuclei *(si veda l'illustrazione di pagina 106)*.

```
SPL A
```

Quando l'esecuzione raggiunge questo punto, si divide in due parti, l'istruzione che segue SPL e quella distante A indirizzi. Questo consente immediatamente a ogni giocatore della Guerra dei nuclei di far girare più programmi alla volta, quindi è necessario definire il modo in cui MARS assegnerà queste esecuzioni. Sotto questo profilo, esistono due diverse possibilità.

Per illustrarle, supponiamo che un giocatore abbia i programmi A_1, A_2 e A_3, mentre l'altro giocatore ha i programmi B_1 e B_2. Un'alternativa è far girare tutti i programmi del primo giocatore, seguiti da quelli del secondo giocatore. L'ordine dell'esecuzione sarebbe così A_1, A_2, A_3 e poi B_1 e B_2, e il ciclo si ripeterebbe indefinitamente. La seconda possibilità è alternare i programmi dei due giocatori. In questo caso la successione sarebbe A_1, B_1, A_2, B_2, A_3, B_1 e così via. I due schemi sono molto diversi. Il primo mette l'accento su una proliferazione illimitata e sembra quindi limitare il ruolo dell'intelligenza nel gioco. Nel secondo, invece, quanto maggiore è il numero di programmi fatti girare dai due giocatori, tanto minore è il numero di volte che ciascuno di essi sarà eseguito. In questo contesto sembra appropriata una legge dei ritorni decrescenti, quindi ho adottato il secondo schema. Scopo del gioco, in ogni caso, è provocare l'arresto di tutti i programmi nemici.

La nuova istruzione è ricca di possibilità creative. Per illustrarne una delle più modeste, ecco un programma di battaglia chiamato IMP GUN:

```
SPL 2
JMP -1
MOV 0 1
```

Consideriamo quello che avviene quando l'esecuzione arriva per la prima volta alla sommità di questo programma. SPL 2 significa che in seguito saranno assegnate a questo programma due esecuzioni: saranno eseguite sia JMP −1 sia MOV 0 1. La prima istruzione farà sì che il programma rientri nel ciclo e la seconda mette in movimento un IMP. L'IMP si muoverà verso il basso, naturalmente, dato che l'obiettivo del comando MOV sarà sempre l'indirizzo successivo, come indicato dall'1 (positivo). L'IMP così viene generato a ogni ciclo del programma e un flusso senza fine di esecuzioni di IMP scorre attraverso il nucleo deciso a distruggere i programmi nemici. A prima vista, può sembrare che non ci sia alcuna difesa possibile contro un simile esercito di IMP; in realtà una c'è. Bisogna mettere in gioco IMP PIT, un programma ancora più semplice, attivato da un comando SPT inserito in un insieme più esteso di istruzioni volte a proteggere il suo fianco superiore:

```
MOV #0 -1
JMP -1
```

A ogni esecuzione, IMP PIT pone uno zero subito sopra di sé, nella speranza di distruggere un IMP in arrivo. Qui è fondamentale la regola di esecuzione-assegnazione. Se IMP GUN appartiene ad A e IMP PIT appartiene a B, allora A richiede n mosse per eseguire n IMP; solo un IMP può arrivare alla locazione subito sopra l'IMP PIT. A parità di altre condizioni, B deve eseguire IMP PIT solo una volta per eliminare un IMP in arrivo.

Nella versione allargata del gioco della Guerra dei nuclei, si immagina che ogni contendente generi e metta in campo piccoli eserciti di programmi formulati singolarmente per individuare, attaccare, proteggere e anche riparare. Numerose sottigliezze, come quella proposta da John McLean di Washington, D.C., richiedono un'analisi ulteriore. McLean immagina un programma trappola specializzato, che sistema comandi JMP in vari indirizzi in tutta la matrice del nucleo nella speranza di far approdare un comando JMP all'interno di un programma nemico. Ogni JMP collocato in questo modo trasferirebbe l'esecuzione del programma nemico al programma trappola, provocandone per così dire il passaggio al nemico.

Dalla mischia provocata dai programmi di battaglia emerge un problema importante, che ha bisogno di soluzione. Che cosa impedisce a un programma di battaglia di uno dei contendenti di attaccare i suoi colleghi? Appare necessario un sistema di ricognizione.

Tra i molti lettori che hanno costruito sistemi per la Guerra dei nuclei meritano una citazione particolare: Chan Godfrey di Wilton, Connecticut, Graeme R. McRae di Monmouth Junction, New Jersey, e Mike Rosing di Littleton, Colorado, perché hanno messo particolare cura nel definire e documentare i loro progetti. Mi piacerebbe in particolare rendere disponibili ai lettori i documenti di Rosing, ma ho un'altra idea migliore che include questa possibilità e risolve anche altri problemi di comunicazione. Se qualche lettore con un sistema per la Guerra dei nuclei già funzionante si offrirà come direttore di una rete di Guerra dei nuclei, allora si potranno comunicare a tutti gli utenti della Guerra dei nuclei una documentazione dei vari sistemi, proposte di regole, programmi interessanti e battaglie. Un volontario sarà scelto come direttore; gli altri volontari potrebbero dar vita, secondo i loro interessi, a un bollettino, a un comitato per le regole, e così via. In un articolo futuro darò il nome e l'indirizzo del direttore di rete.

Continuano ad arrivare numerosi resoconti di lettori che hanno giocato con l'ecologia del pianeta Wa-Tor (si vedano le «(Ri)creazioni al calcolatore» del febbraio 1985); potrò quindi esaminare solo poche fra le molte esperienze descritte. In linea generale, la scelta dei giusti parametri ha prodotto grosse fluttuazioni nelle popolazioni degli squali e

dei pesci. Alcuni lettori, desiderosi di rendere Wa-Tor più simile alla Terra, hanno aggiunto particolari caratteristiche ai loro programmi. Il gioco, in effetti, invita a una sua complicazione, che è certamente benvenuta. L'introduzione di un sistema variante, però, ha il grosso svantaggio (a parità di altri fattori) di rendere ardui i confronti con il sistema standard.

Costruttori di un sistema iniziale sono stati Jean H. Anderson di Lauderdale, Minnesota, Stephen R. Berggren di Satellite Beach, Florida, Milton Boyd di Amherst, New Hampshire, J. Connett di Minneapolis, Minnesota, Edgar F. Couda di Park Ridge, Illinois, Jim Lemon di El Segundo, California, e Kenneth D. Wright di Grayling, Michigan.

Tra le questioni che questi e altri lettori si sono trovati ad affrontare c'era la durata della sopravvivenza. Chiaramente, non c'è alcun problema per le popolazioni eterne, ma sarebbe utile avere una misura delle sceneggiature che non sono eterne. Come rileva Stevens, misurare con i crononi può essere fuorviante quando si scelgono la durata delle estensioni di vita e i tempi di riproduzione per gli squali. Anche la misurazione per cicli solleva problemi: che cos'è un ciclo? Stevens fa la divertente osservazione che se gli squali e i pesci sopravvivono a un sufficiente numero di ripetizioni del ciclo base con numeri casuali, si ripeterà una configurazione precedente, in accordo con il ciclo, e da lì in avanti alle popolazioni sarà ovviamente garantita la vita eterna.

Un gran numero di lettori, tra cui David Emanuel di Oak Brook, Illinois, Richard G. Fizell di Fort Washington, Maryland, e John S. Lew dello IBM Thomas J. Watson Research Center di Yorktown, New York, hanno descritto teorie moderne utili all'analisi di Wa-Tor. Non è stata ancora detta l'ultima parola a proposito del dilemma se le matrici stocastiche ci metteranno in grado di derivare specifiche probabilità di sopravvivenza da combinazioni arbitrarie di parametri o no. È interessante notare, però, che le equazioni di Lotka-Volterra (dalla loro formulazione nel 1931) sono state elaborate in modo da prendere in considerazione la diffusione come fattore che riguarda sia il predatore sia la preda. La diffusione fa assumere forme più complesse alle soluzioni di Lotka-Volterra, che di solito variano con regolarità. Una nota storica di Lew ci precisa come Alfred J. Lotka fosse un matematico americano il quale, un decennio prima, aveva formulato la stessa equazione di Volterra.

Boyd ha sfruttato un diagramma di fase per analizzare la dinamica delle popolazioni di squali e altri pesci. A ogni istante t, si riportano in grafico il numero x dei pesci e quello y degli squali come coordinate di un singolo punto. Man mano che il tempo avanza e le popolazioni seguono il loro ciclo, il punto descrive un'orbita erratica intorno a un occhio, o centro, fisso. Boyd ha usato questa tecnica per studiare l'effetto delle dimensioni dell'oceano sulla sopravvivenza e scrive che «per i mondi più rettangolari, le orbite persero il loro occhio, le traiettorie divennero più nervose, per diventare infine percorsi casuali». Evidentemente, sono preferibili oceani quadrati.

Tra le innovazioni introdotte dai lettori possiamo annoverare una forza vitale degli squali, mutazioni, doppie popolazioni di pesci e plancton. Nel mio articolo di febbraio avevo trascurato di dire che i pesci comuni di Wa-Tor si cibano di un plancton oceanico sparso ovunque in abbondanza. Lemon ha reso esplicita questa caratteristica facendo in modo di mettere plancton in ogni punto non occupato da uno squalo o da un altro pesce. Il plancton prolifica in punti altrimenti vuoti e ha con i pesci comuni la stessa relazione che i pesci comuni hanno con gli squali. Anche in questo caso esistono popolazioni eterne.

Gli squali di Couda guadagnano o perdono punti di forza vitale a seconda di quanto mangiano. Possono così sopravvivere senza cibo molto più a lungo dei semplici squali dello Wa-Tor standard. Couda (come molti altri programmatori di Wa-Tor) ha inviato diagrammi che sono notevolmente simili a quelli che si ottengono in base ai dati della Hudson's Bay Company.

Connett utilizza due specie di pesci. Una è la varietà standard di Wa-Tor; l'altra si riproduce sempre in qualsiasi punto vuoto a sud o a est. A causa della sua tendenza alla mobilità, la seconda specie spesso sopravvive alla prima. Rudy Iwasako di Sacramento, California, ha proposto di attribuire agli squali e agli altri pesci caratteristiche di dimensione, velocità e agilità, sottoposte a controllo genetico. Berggren ha scritto il suo sistema, chiamato EVOLVE, due anni fa. Esso è simile a WATOR, ma lascia che gli animali si evolvano secondo le pressioni ambientali. A giudizio di Berggren, le popolazioni arriverebbero a un equilibrio favorevole alla sopravvivenza a lungo termine.

Nessuno è riuscito a risolvere il problema dell'inseguimento toroidale. Rivelerò solo metà della soluzione, in modo da riservare ai lettori il piacere di trovare l'altra metà. Si ricordi che, a ogni turno, il pesce si muove e poi si muovono i due squali. Come in Wa-Tor, non è consentito rimanere nello stesso punto. Ogni raggio segue una diagonale e gira intorno al toro, ricongiungendosi presto o tardi con se stesso. Quando entrambi gli squali occupano una coppia di raggi opposti, non importa in che modo il pesce si muove: uno squalo insegue a distanza costante e l'altro squalo si avvicina. Il pesce è condannato. Lascio ai lettori scoprire in che modo gli squali, per così dire, vadano alla caccia dei raggi.