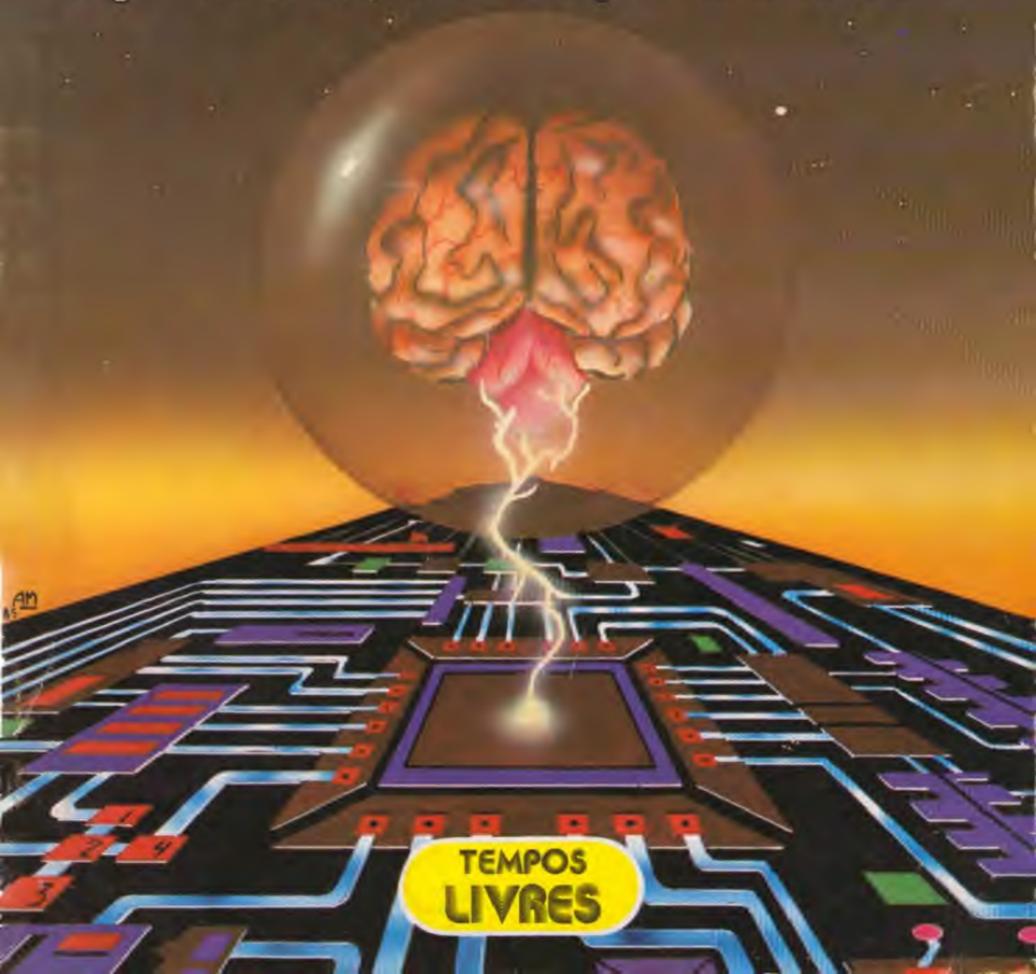


TIM HARTNELL

# INTELIGÊNCIA ARTIFICIAL

no seu  
Spectrum e Spectrum +



TEMPOS  
LIVRES

**TIM HARTNELL**

**INTELIGÊNCIA ARTIFICIAL  
NO SEU SPECTRUM E  
SPECTRUM +**

EDITORIAL  PRESENÇA

Titulo original:

**EXPLORING ARTIFICIAL INTELLIGENCE ON YOUR  
SPECTRUM + AND SPECTRUM**

Copyright © Tim Hartnell, 1984

Tradução de Conceição Jardim e Eduardo Nogueira

Capa de António Marques

Reservados todos os direitos

para a língua portuguesa à

**EDITORIAL PRESENÇA, LDA.**

Rua Augusto Gil, 35-A - 1000 LISBOA

## PREFÁCIO

Você prepara-se para iniciar uma viagem fascinante, aos domínios onde os factos científicos interactuam com a ficção científica.

Desde que foram produzidos os primeiros computadores, têm-se arrastado longos debates sobre os seguintes tópicos:

- Poderá uma máquina pensar verdadeiramente?
- Qual é a natureza da inteligência, e poderá ser alguma vez construída uma máquina que a possua?

Depois de ter lido este livro, estará também pronto a entrar neste debate, e com conhecimento de causa. De facto, vamos investigar aqui o mundo fascinante da inteligência artificial, apresentando réplicas de alguns dos seus programas mais famosos.

Desde os programas que aprendem e raciocinam, até àqueles que falam consigo, lhe obedecem e o aconselham, referir-nos-emos a muitos assuntos.

Foi fascinante escrever este livro. Ler a extensa literatura sobre o tema, conhecer as aspirações dos pioneiros da inteligência artificial, e escrever programas que – embora de um modo grosseiro – permitem reproduzir algumas das suas descobertas num microcomputador foi certamente uma experiência bastante interessante.

Espero que parte do fascínio que senti possa ser transmitido pelo livro a todos os que o lerem.

Tim Hartnell  
Londres, 1984

## I PARTE PENSAR

### 1

#### APRENDER E RACIOCINAR

Decorre actualmente um debate sobre se a produção de uma máquina capaz de actuar de um modo aparentemente inteligente nos aproxima de facto da produção da inteligência. Uma questão relacionada com isto, inextricavelmente ligada a este debate, tem a ver com a natureza da inteligência.

Os programas deste livro permitem certamente ao computador apresentar respostas inteligentes a situações, tomando decisões e actuando de acordo com elas. No entanto, não se pretende sugerir que o computador tenha alguma consciência das suas acções. Não se ri com os resultados obtidos em DOCTOR, e não consegue admirar – ou sequer reconhecer – um poema particularmente eficaz produzido por HANSHAN.

Existirá então alguma justificação para afirmar que estamos a produzir «inteligência artificial»? Parece-me que sem o tipo de percepção capaz de reconhecer coisas como a «eficácia» de um poema, ou a incongruência de uma resposta, não podemos de facto afirmar que está presente uma inteligência.

A inteligência artificial encontra-se na sua infância, e esperar consciência e percepção num curto programa Basic executado num microcomputador, quando mesmo os maiores computadores ainda não as apresentam, não é obviamente realista.

No entanto, existem duas áreas de comportamento que se podem considerar como razoáveis candidatos a serem aceites como inteligentes, e que podem ser exploradas no seu computador. Trata-se dos campos da *aprendizagem* e do *raciocínio*.

O primeiro programa, que executa um jogo muito conhecido – «Três em Linha» – começa por ter apenas um conhecimento sumário da forma de ganhar e do modo como deve impedir a vitória do adversário. Não sabe quais os primeiros movimentos

que deve fazer a fim de aumentar a sua probabilidade de ganhar. De facto, a sua base inicial de conhecimento é tal que joga tão mal quanto possível.

Mas, quando enfrenta um adversário que jogue completamente ao acaso (um opositor que nem sequer saiba que o jogo é ganho colocando três marcas em linha), o programa terá aprendido em pouco tempo a vantagem de ocupar o quadrado central tão depressa quanto possível, e terá ordenado os seus outros movimentos numa sequência que – apesar de diferir daquela que qualquer de nós poderia criar em circunstâncias semelhantes – lhe permite ganhar uma proporção cada vez maior de jogos, mesmo contra um adversário inteligente.

O programa foi escrito de modo a apresentar ao utilizador o estado actual dos seus conhecimentos. Torna-se assim muito mais interessante executá-lo, e o leitor poderá sem dificuldade ampliar o programa de modo a investigar a sua capacidade de aprender.

SILOGISMO será o nosso programa seguinte, capaz de «raciocinar». Pretende resolver silogismos, como o que apresentamos em seguida:

```
Sócrates é um homem
Todos os homens são mortais.
Portanto, Sócrates é mortal.
```

Das duas premissas iniciais, SILOGISMO consegue tirar uma conclusão razoável. O aspecto importante a notar é que SILOGISMO pode chegar a conclusões baseando-se em informações que não lhe foram apresentadas explicitamente.

Vou explicar isto um pouco melhor. Observemos as duas premissas seguintes:

```
Uma novela é um livro.
Um livro é impresso em papel.
```

Se bem que não se tenha dito explicitamente ao computador que uma novela é escrita em papel, este responderá SIM quando lhe for perguntado:

```
É uma novela impressa em papel?
```

O leitor poderá certamente divertir-se bastante apresentando ao computador uma longa lista de premissas e fazendo depois várias perguntas à máquina, observando as respostas que SILOGISMO é capaz de fornecer. Algumas delas serão: «Não possuo dados sobre isso», «Não», «Não sei», etc.

Nas primeiras fases do debate sobre a possibilidade de uma máquina ser inteligente tornou-se óbvio que os termos básicos discutidos necessitavam de ser melhor definidos. Que significam de facto as palavras «pensar» e «pensamento»? Se não sabemos de facto a que nos referimos ao usar estes termos a propósito de nós próprios, como podemos fazer juízos sobre o comportamento das máquinas neste campo?

Este tipo de ideia constitui um dos muitos efeitos produzidos pelo estudo da inteligência artificial. O homem foi forçado a observar-se melhor a si mesmo, e a examinar áreas do comportamento humano de um modo que poucos tinham considerado até então.

Já afirmei que apesar de as máquinas não estarem sequer a aproximar-se do tipo de consciência que parece ser um requisito vital para afirmar que a inteligência existe num sistema, alguns aspectos da inteligência – o raciocínio e a capacidade de aprender – estão dentro das suas potencialidades actuais.

Existem diferentes tipos de aprendizagem. Podemos aprender observando os outros, lendo, ouvindo dizer (o que é uma espécie de «leitura verbal», pelo que ambas estão relacionadas entre si) e por experiência. Os computadores podem aprender de qualquer um destes modos. O programa aqui apresentado, que chamaremos de «Três em Linha», aprende essencialmente por experiência, se bem que já disponha de algum conhecimento inicial (que lhe foi «dito»).

## **Feedback**

Como é óbvio, os erros de «Três em Linha» não terião qualquer significado se não recebesse alguma informação quanto ao êxito ou fracasso dos seus esforços. Este contacto com o exterior é um elemento essencial da aprendizagem.

Uma das primeiras máquinas capazes de «aprender» foi a

tartaruga, a primeira de uma multidão de *robots*, construída em 1948 por Grey Walter, um fisiologista que se especializou em problemas do cérebro. Construiu a sua tartaruga – um semi-globo que se movia pelo chão, rodeava obstáculos e voltava a casa quando as suas baterias estavam fracas – para demonstrar a sua tese de que o comportamento complexo se baseava em interacções entre apenas algumas ideias básicas.

A tartaruga aprendia a mover-se utilizando uma realimentação negativa do exterior, ou seja, tendia a não repetir um comportamento que não era produtivo. Uma tartaruga que não compreendesse que mover-se repetidamente contra uma parede não é um bom modo de passear não conseguiria certamente andar muito.

### **Como pensam as máquinas?**

Os computadores actuais são processadores seriais. Isto é, passam de um ponto a outro, um passo de cada vez, sendo os passos futuros determinados pelos resultados dos presentes. O cérebro humano, pelo contrário, utiliza não só o processamento serial, mas também o processamento paralelo, no qual um certo número de direcções de pensamento – umas conscientes, outras não – são executadas simultaneamente.

O processo de pensamento e tomada de decisões de um computador é essencialmente um trajecto através de um labirinto de construções IF/THEN:

SE (If) isto é verdadeiro E (And) isto é verdadeiro  
E (And) isto é falso ENTÃO (Then) fazer isto

O computador, como é óbvio, pode tomar decisões OU (Or) do mesmo modo que decisões E (And):

SE (If) isto é verdadeiro OU (Or) isto é verdadeiro  
ENTÃO (Then) fazer isto

Ambos os raciocínios lógicos podem ser combinados:

SE (If) isto é verdadeiro E (And) isto é verdadeiro  
OU (Or) aquilo é verdadeiro ENTÃO (Then) fazer  
isto.

Como faz isto? O primeiro dispositivo electrónico de cálculo foi construído (na cozinha) por George Stibitz que trabalhou para os Bell Telephone Laboratories na década de 1940. Ligou baterias, lâmpadas e alguns relés telefónicos, e conseguiu efectuar cálculos em binário (este sistema numérico só possui os algarismos 0 e 1. Um interruptor ligado pode ser considerado como encontrando-se ao nível «1», e desligado ao nível «0»). Stibitz compreendeu que esta máquina incipiente, se fosse suficientemente desenvolvida, poderia resolver qualquer tipo de problemas matemáticos (o que aparentemente não compreendeu – como veremos daqui a pouco – é que os mesmos circuitos que estava a usar podiam ser utilizados para tomar decisões).

No entanto, alguns anos antes, em 1937, Claude Shannon (que também veio a trabalhar para a Bell) fizera uma tese sobre a relação entre a álgebra booleana e o fluxo de energia através de circuitos de comutação.

A álgebra booleana – que é onde se inicia a parte «pensante» das máquinas – baseia-se na obra de George Boole, um professor do Queens College, Cork, em meados do século dezanove. O seu livro *An Investigation of the Laws of Thought on Which Are Founded the Mathematical Theories of Logic and Probabilities* (publicado em 1854) lançou os fundamentos da lógica simbólica moderna. A álgebra booleana baseia-se nas regras por ele apresentadas, e é o suporte em que se baseia a capacidade de raciocinar do computador.

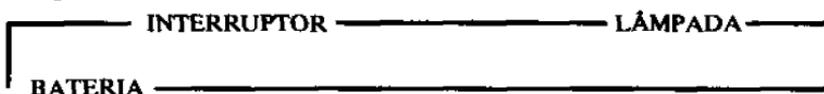
Boole escreveu no prefácio da sua obra:

«As leis que temos de examinar são as leis que regulam uma das nossas faculdades mentais mais importantes. As matemáticas que temos de construir são as do intelecto humano.»

Até às descobertas de Boole, considerara-se que a lógica era um ramo da filosofia. Boole mostrou claramente que, em vez disso, pertencia sem dúvida ao campo das matemáticas.

## Interruptores e decisões

Podemos investigar as afirmações de Boole, e verificar o modo como se relacionam com o seu computador, a tomada de decisões e a inteligência artificial, reconstruindo mentalmente alguns dos dispositivos que Stibitz montou na mesa da sua cozinha. Começaremos por um circuito muito simples, contendo uma unidade de alimentação, um simples interruptor, e uma lâmpada:



O leitor poderá verificar que quando o interruptor está fechado, passará energia e a luz acender-se-á. Indicaremos que o interruptor foi ligado, dizendo que o seu estado é «1». Quando o interruptor está aberto, e a corrente não passa, o seu estado será «0». Ligado é igual a 1, desligado é igual a 0. Adoptaremos por outro lado, a convenção segundo a qual quando a lâmpada está acesa, o seu estado é 1; quando está apagada, o seu estado é 0.

Diremos que este circuito indica uma afirmação. Quando o interruptor está fechado, a lâmpada está acesa. Ou seja, o estado do interruptor é igual ao estado da lâmpada. Se construirmos uma pequena tabela que mostre a relação entre os estados da lâmpada e do interruptor num circuito deste tipo, obteremos algo do seguinte tipo:

Interruptor	Lâmpada
0	0
1	1

Uma tabela deste tipo, a propósito, é designada por «tabela de verdade».

Vejamos agora outro circuito simples:



Se observar o diagrama, verificará que a lâmpada está acesa (estado da lâmpada igual a 1) quando o interruptor está ligado (o estado do interruptor é 0) e – depois de o interruptor estar fechado (estado do interruptor igual a 1) – a corrente passará através dele e não através da lâmpada.

Este é um circuito de negação, e a sua tabela de verdade é a seguinte:

Interruptor	Lâmpada
0	1
1	0

Agora chegamos às questões interessantes, os circuitos que podem «tomar decisões». Imaginemos que temos um circuito com dois interruptores, como se segue:

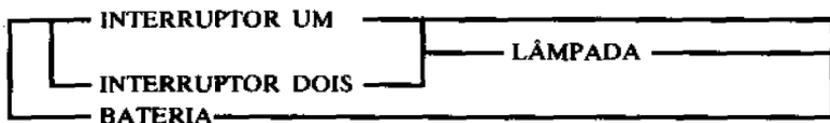


Com ambos os interruptores fechados (isto é, ligados, com ambos os estados iguais a um) a lâmpada acende-se. Se qualquer dos interruptores estiver desligado (um deles igual a 1, o outro igual a 0) ou ambos desligados a lâmpada ficará apagada. Chama-se a isto um circuito de porta AND.

A sua tabela de verdade é a seguinte:

Interruptor Um	Interruptor Dois	Lâmpada
0	0	0
1	0	0
0	1	0
1	1	1

Da AND passamos à OR. O circuito OR é do seguinte tipo:



Neste circuito, com os circuitos em paralelo (encontravam-se em série no circuito AND), a lâmpada estará ligada (no estado um) se qualquer dos interruptores estiver a esse nível ( [0 1] ou [1 0] ), ou se ambos o estiverem [1 1]. Antes de continuar a ler, construa uma tabela de verdade para o circuito OR.

Interruptor Um	Interruptor Dois	Lâmpada
0	0	0
1	0	1
0	1	1
1	1	1

### Portas lógicas «reais»

O seu computador utiliza portas lógicas como estas, se bem que obviamente não existam grandes interruptores para ligar e desligar. Uma das razões de Shannon e Stibitz usarem relés foi por estes serem interruptores que podem ser ligados e desligados sem se tocar neles (quando é aplicada uma corrente eléctrica, é produzida uma força magnética que fecha o interruptor).

Não existem relés do tipo empregue por Shannon no seu computador, se bem que certos elementos dos circuitos integrados nele usados actuem como relés. Nos diagramas esquemáticos destes circuitos, as portas lógicas que examinámos até agora são representadas do seguinte modo:

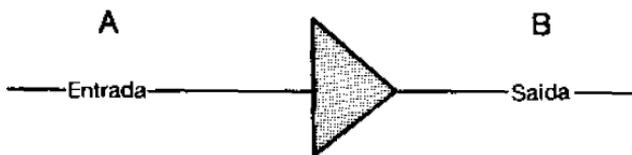


Fig. 1 - A é igual ao oposto de B

Em primeiro lugar encontramos o «inversor». Se um sinal de entrada está ao nível 0, abandona o circuito ao estado 1, e vice-versa.

Eis agora a porta AND:



Fig. 2

E a porta OR:



Fig. 3

Existe ainda uma outra porta, que será muito útil ao leitor quando tentar compreender como os circuitos realizam decisões. Trata-se da porta XOR, ou OR EXCLUSIVO. Se qualquer das entradas for 1 ( [1 0] ou [0 1] ), o estado da saída é sempre 1. No entanto, se ambas as entradas forem 1 ( [1 1] ), ou ambas 0 ( [0 0] ), o estado da saída é 0.

Vejamos a representação esquemática da porta XOR:

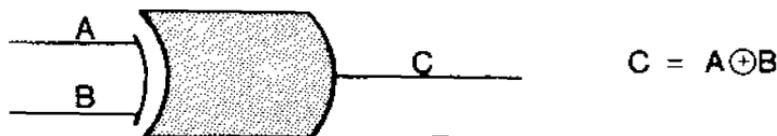


Fig. 4

E agora a tabela de verdade da porta XOR:

Interruptor Um (A)	Interruptor Dois (B)	Lâmpada (C)
0	0	0
0	1	1
1	0	1
1	1	0

Usando apenas estes poucos elementos, pode já construir-se «circuitos» lógicos, capazes de tomarem decisões. Pode escolher-se facilmente, por exemplo, a partir das portas que examinámos (e sugiro que tente descobrir quais) uma sequência de portas que representem declarações como a que se segue:

IF A AND B são verdadeiros AND C OR D  
(mas não ambos) verdadeiro, THEN D é verdadeiro

A definição dos arranjos de comutadores que executam a declaração acima – e outras como ela – é fascinante, e pode dar uma boa ideia da forma como uma sequência simples de operações booleanas pode processar decisões e atingir resultados (se o circuito citado fosse correctamente construído, poderia ser ligada uma lâmpada em D). Fazê-lo, ajudaria o leitor a compreender melhor o modo de funcionamento do computador, e a partir dessa compreensão ser-lhe-ia mais fácil ter uma noção da complexidade que estes circuitos devem ter para poderem simular qualquer tipo de comportamento «inteligente».

Consideremos, por exemplo, os circuitos necessários para executar o trabalho do computador no primeiro programa desta secção, «Três em Linha».

## 2

### UM PROGRAMA QUE APRENDE

Muitos programas de inteligência artificial não são introduzidos no computador já completamente formados. Mesmo que não tenham erros, e funcionem correctamente, estão muito longe de se encontrarem terminados. O programa que vamos estudar nesta secção do livro, «Três em Linha», é deste tipo. «Três em Linha» aprende à medida que joga, modificando as suas regras à luz dos êxitos ou falta destes, no seu comportamento.

Um programa que vai aprender à medida que corre necessita de ter as suas normas de funcionamento definidas de tal modo que possam ser modificadas à medida que evolui. Neste programa, o computador conhece as regras do jogo, e dispõe de uma secção que serve especificamente para bloquear a formação de linhas pelo adversário; sabe ainda como deve completar uma linha para ganhar, mas não conhece qualquer estratégia de jogo inicialmente.

Vejamos o «tabuleiro» usado para o jogo:

1	:	2	:	3
<hr/>				
4	:	5	:	6
<hr/>				
7	:	8	:	9

Fig. 5

O programa joga escolhendo quadrados de acordo com uma sequência que desenvolve à medida que o jogo progride.

Se o jogo tem êxito, desloca as posições mais apropriadas para o início da sequência. Não faz qualquer alteração quando o jogo é um empate. Ao perder, baralha os elementos da sequência, a fim de não escolher os mesmos movimentos da vez seguinte.

O leitor sabe que o quadrado central do tabuleiro (representado pelo 5 no diagrama anterior) deve ser ocupado assim que possível se estiver livre. Inicialmente, «Três em Linha» não conhece este facto. Aliás, é-lhe deliberadamente dada no início uma sequência de jogo bastante má – cujo primeiro elemento é a posição 2 – o que permite observar mais facilmente o modo como aprende.

Ao fim de algum tempo, quando o mecanismo de aprendizagem funciona, «Três em Linha» acaba por compreender que a posição cinco é bastante boa. De facto, como veremos, «Três em Linha» chega a esta conclusão mesmo quando está a jogar contra um opositor que não possua qualquer conhecimento estratégico do jogo. É razoável partir do princípio de que quando «Três em Linha» está a jogar contra um adversário inteligente – como o leitor – melhora o seu conhecimento ainda mais depressa.

Donald Michie, um pioneiro da investigação em inteligência artificial na Universidade de Edimburgo, investigou a «aprendizagem automática» neste jogo. Recorreu a um mecanismo chamado «caixas», no qual um objectivo é dividido em vários objectivos parcelares. Forma-se uma «caixa» para guardar a informação relativa a cada um destes.

O objectivo de «Três em Linha» é ganhar. Os objectivos parcelares consistem em fazer um movimento legal (a), e se possível (b) o melhor movimento para cada posição de jogo.

Michie verificou que existem 288 posições fundamentalmente diferentes no início do jogo. Construiu então o seu adversário mecânico do seguinte modo (o leitor poderá aliás repetir esta experiência). Arranjou 288 caixas de fósforos, e pintou na tampa de cada uma delas uma posição do tabuleiro, numerando sequencialmente os quadrados vazios. Em seguida escreveu, em pequenos pedaços de papel, os números que se encontravam escritos nos quadrados vazios. Cada número foi repetido várias vezes; se os quadrados três e quatro estavam vazios numa dada posição do tabuleiro, a caixa continha, digamos, cinco

pedaços de papel com o número três, e cinco com o número quatro.

Executou depois o jogo do seguinte modo. O primeiro movimento era feito abrindo a caixa com um tabuleiro vazio no topo. No interior da caixa, obviamente, existiam cinco pedaços de papel para cada um dos números um a nove. Escolhia ao acaso um destes pedaços de papel, executando o movimento. Michie tomava nota do número escolhido, e da caixa de onde fora tirado.

No final do jogo, Michie estudava a sua lista de movimentos e caixas. Se o «computador de caixas de fósforos» tivesse ganho o jogo, era colocada na caixa relevante um papel adicional com cada um dos números jogados. Isto é, se a primeira caixa, a que possuía um tabuleiro vazio pintado, tivesse produzido o número cinco, era incluído nela mais um pedaço de papel contendo esse número. Como é natural, isto aumentava a probabilidade de este número ser seleccionado quando a caixa fosse novamente aberta.

O processo foi depois continuado para cada uma das caixas usadas no jogo. Se o jogo produzia um empate, o conteúdo das caixas era deixado tal como estava. Se o «computador» perdesse o jogo, os pedaços de papel que produziam os movimentos perdentes eram retirados das caixas, diminuindo assim a probabilidade de voltarem a sair.

Num texto publicado em 1968, *Boxes: An Experiment in Adaptive Control* [Chambers, R.A., e Michie, D., *Machine Intelligence 2*, (Ed. Dale, E. e Michie, D.), Oliver & Boyd, 1968, págs. 137-152], Michie explica que as caixas «aprenderam» tão bem que após 1000 jogos contra um adversário jogando completamente ao acaso o programa passou a ganhar entre 75 e 87% das vezes. Não se espera uma razão de êxitos tão elevada do programa que se segue (mesmo que o leitor tenha a paciência necessária para jogar 1000 vezes), mas conseguirá mesmo assim resultados bastante prometedores se lhe forem dadas possibilidades de aprender.

### **Samuel e as damas**

As «caixas inteligentes» de Michie eram apenas um brin-

quedo quando comparadas ao programa de damas criado no final dos anos sessenta por Arthur Samuel, da IBM. Estamos a discutir aqui um dos seus últimos programas, descrito no texto *Some Studies in Machine Learning Using the Game of Checkers - II - Recent Progress* (Samuel, A., IBM Journal of Research and Development, vol. II, November 1967, págs. 601-617). No entanto, é interessante notar que o programa final não saiu de facto do seu cérebro, em toda a sua majestade.

De facto, Samuel começou a programar jogos de tabuleiro em 1952 trabalhando no então potente IBM 701. Dois anos mais tarde, transferiu o programa para um IBM 704, e em 1955 começou a desenvolver a capacidade de aprender do programa. Este tinha em conta cerca de 40 factores para decidir um lance, se bem que menos de metade destes fossem pertinentes em cada situação particular. O programa sabia quando um dos factores não interessava para a situação, e ignorava-o.

O número de peças possuído por cada jogador era obviamente importante, e o programa de Samuel (tal como a maioria dos programas que se lhe seguiram) adorava trocar peças quando tinha mais do que o adversário, já não apreciando fazê-lo quando estava a perder. Outros factores considerados pelo programa ao avaliar a sua força incluíam o controlo do centro do tabuleiro, e o número de peças que podiam ser atacadas num único movimento.

Observaremos um pouco mais adiante os aspectos da inteligência artificial relacionados com os jogos de tabuleiro; por agora o que mais nos interessa é a capacidade de aprender do jogo Mini-Damas, que reproduz o original de Samuel, CHECKERS. Este podia aprender de dois modos: mecanicamente, e por auto-modificação.

No primeiro caso, o programa guarda os resultados das investigações de lances possíveis a partir da posição actual no tabuleiro. Isto significa que da próxima vez que esta posição seja encontrada, o programa não necessita de determinar novamente todas as implicações de cada movimento. O resultado já é conhecido. Este método consome evidentemente muita memória, se bem que seja altamente eficaz. O programa acaba por funcionar a um nível muito elevado, sendo capaz de «recordar» praticamente tudo o que interessa para cada posição no tabuleiro.



```

1310 REM inicializar base de
1320 conhecimento
1330 FOR J=1 TO 10
1340 DIMD # (J)
1350 NEXT J
1360 DATA 2,6,8,4,7,5,1,9,5,2
1370 RETURN

```

São dimensionados quatro quadros. O quadro A contém o tabuleiro actual, a «base de conhecimento» dos movimentos (que é actualizada após cada jogo perdido ou ganho), W contém os dados a partir dos quais o programa consegue reconhecer uma vitória potencial do computador ou do adversário, e D contém os movimentos do jogo em curso, a fim de poder utilizá-los para modificar a base de conhecimento no final do jogo.

Como o leitor poderá constatar na linha 1350, o programa parte de uma base de conhecimento constituída pelos números 2, 6, 8, 4, 7, 5, 1, 9, 5 e 2. Trata-se, como já disse anteriormente, de uma sequência de movimentos particularmente má, que assegura na prática a perda de uma quantidade significativa dos primeiros jogos. Se duvidar disto, imagine estes movimentos no tabuleiro que estamos a usar:

1	:	2	:	3
4	:	5	:	6
7	:	8	:	9

Fig. 6

Note que o programa não executa necessariamente os movimentos pela ordem apresentada. Tenta fazê-lo, mas pode verificar que uma dada posição já foi ocupada. Por outro lado, só utiliza uma sequência depois de ter testado as possibilidades de o seu adversário completar uma linha, ou de ele próprio o fazer.

Observar o programa a aprender é particularmente fascinante. Nestas condições, uma parte do programa serve para indicar ao utilizador as alterações que a base de conhecimento vai sofrendo. A actualização desta base de conhecimento, e a

impressão desta, são executadas pela parte do programa entre as linhas 300 e 480:

```

300 REM actualizar base de
301 REM conhecimento
302 REM J=1 TO 10
303 REM # (J)=a(b) THEN GO SUB 37
304
305 NEXT J
306 REM # (J)=0 THEN LET J=INT (R
307 + 7) : GO SUB 370
308 REM TO 480
309 REM reordenar elementos do
310 REM array
311 LET temp=#(J+flag)
312 LET # (J+flag)=#(J)
313 LET #(J)=temp
314 LET J=J-1
315 RETURN
316 PRINT : PRINT
317 PRINT "EIG A MENOR PRIORIO"
DE 480 DC
318 PRINT : PRINT
319 PRINT J=1 TO 10
320 PRINT # (J) : " "
321 NEXT J

```

Vejamos em seguida um exemplo da evolução da base de conhecimento do programa em jogo contra um adversário executando movimentos aleatórios. Apesar da falta de uma oposição inteligente, o programa conseguiu aprender com bastante rapidez. O leitor pode verificar que «Três em Linha» se apercebe rapidamente do valor da posição central (número cinco no tabuleiro):

0	0	0	4	7	0	1	0	0
0	0	4	0	7	0	0	1	0
0	0	4	0	0	7	1	0	0
0	4	0	0	0	7	0	1	0
4	0	0	0	7	0	1	0	0
0	4	0	0	7	0	0	1	0



00	4	00	00	00	00	7	4	00
00	4	00	00	00	00	7	4	00
4	00	00	00	00	00	7	4	00
00	4	00	00	00	00	7	4	00

O programa foi ligeiramente modificado, sendo-lhe comunicada uma nova sequência inicial, que pensei ser a melhor possível. O computador jogou primeiro contra um ser humano, tendo desenvolvido do seguinte modo a sua base de conhecimento:

00	4	00	7	00	00	4	00	00
4	00	7	00	00	00	4	00	00
00	7	00	4	00	00	4	00	00
7	00	00	4	00	00	4	00	00
00	00	7	4	00	00	4	00	00
00	00	7	4	00	00	4	00	00

Em seguida jogou contra um adversário «aleatório». Pode verificar-se que o programa pouco aprende, limitando-se simplesmente a trocar alguns números de forma pouco significativa:

4	00	00	00	7	00	4	00	00
00	4	00	00	7	00	4	00	00
4	00	00	00	7	00	4	00	00
4	00	00	00	7	00	4	00	00
00	00	4	00	7	00	4	00	00
00	4	00	00	7	00	4	00	00
4	00	00	00	7	00	4	00	00
4	00	00	00	7	00	4	00	00
00	00	4	00	7	00	4	00	00

0	1	0	0	7	0	4	0	0
1	0	0	0	7	0	4	0	0

Finalmente, voltei à má sequência inicial, e deixei o computador tentar vencer o seu adversário «aleatório». Ao fim de 90 jogos, a sequência era a seguinte:

0	7	4	0	0	0	0	0	0
7	0	0	4	0	0	0	0	0
7	0	4	0	0	0	0	0	0
7	4	0	0	0	0	0	0	0
4	0	7	0	0	0	0	0	0
4	7	0	0	0	0	0	0	0
7	4	0	0	0	0	0	0	0
7	4	0	0	0	0	0	0	0
4	7	0	0	0	0	0	0	0
7	4	0	0	0	0	0	0	0
4	7	0	0	0	0	0	0	0
7	4	0	0	0	0	0	0	0
4	7	0	0	0	0	0	0	0
7	4	0	0	0	0	0	0	0
7	4	0	0	0	0	0	0	0

É fácil encontrar uma fraqueza do programa. Se bem que de facto «aprenda», ao fim de algum tempo parece ser fácil persuadi-lo a trocar números mesmo que isto não conduza a um jogo melhor. Talvez o leitor queira melhorar a forma como o computador utiliza as lições que recebe com cada jogo.

Já referi anteriormente o facto de «Três em Linha» dispor de uma estratégia que não reside apenas na sua base de conhe-

cimento. Além disso dispõe de informações quanto às linhas que tenta construir (ou que tenta impedir o adversário de construir). Vejamos agora a parte do programa que procura determinar um movimento a partir da base de conhecimento:

```

640 REM movimento do Spectrum
650 LET p=CODE "O"
660 LET x=0
670 LET j=1
680 IF a(w(j))=a(w(j+1)) AND a(
w(j+2))=00 AND a(w(j))=p THEN LE
T x=w(j+1): GO TO 750
690 IF a(w(j))=a(w(j+2)) AND a(
w(j+1))=00 AND a(w(j))=p THEN LE
T x=w(j+1): GO TO 750
700 IF a(w(j+1))=a(w(j+2)) AND
a(w(j))=00 AND a(w(j+1))=p THEN
LET x=w(j): GO TO 750
710 IF j<21 THEN LET j=j+3: GO
TO 680
720 IF p=CODE "O" THEN LET p=CO
DE "X": GO TO 670
730 REM se nao encontra bloco/
/ ganhar, usar seccao que se
segue
740 LET j=1
750 IF a(w(j))=00 THEN LET x=w(
j): GO TO 750
760 IF j<10 THEN LET j=j+1: GO
TO 750

```

Primeiramente procura um movimento ganhante (quando P é igual ao código ASCII da letra «O»), e em seguida tenta descobrir um movimento de bloqueio ao jogo adversário (quando P é igual ao código da peça adversária, «X»). Se ainda não encontra um movimento apropriado, utiliza os dados da sua base de conhecimento (linhas 630 a 670).

Se ainda não consegue executar o seu lance, experimenta posições ao acaso:

```

680 LET h=0
690 LET h=h+1
700 LET x=INT (RAND*9)+1: IF a(x
)=00 THEN GO TO 750
710 IF h<100 THEN GO TO 690
720 LET r$="D": REM empate

```

Tendo descoberto um movimento, actua de modo a assegurar que, se todas as posições estiverem preenchidas e R\$ (que significa quadro de «resultado», sendo a vitória identificada por um W, a derrota por um L e o empate por um D) não tiver sido atribuída, o jogo seja um empate:

```

750 REM movimento
760 LET a(x)=CODE "O"
770 LET count=count+1
780 LET a(count)=x
790 LET flag=0
800 FOR J=1 TO 9
810 IF a(j)=0 THEN LET flag=1
820 NEXT J
830 IF flag=0 AND r#="" THEN LET
T r#="D"
840 REM se todas as posições
ocupadas, e r# não definida
é empate
850 RETURN

```

Depois de cada movimento, humano ou não, é executada a rotina que verifica uma possível vitória:

```

870 REM verificar victoria
880 LET J=1
890 IF a(w(j))=0 THEN LET J=J+
3
900 IF J>9 THEN RETURN
910 IF a(w(j))=a(w(j+1)) AND a(
w(j))=a(w(j+2)) THEN GO TO 940
920 IF J<9 THEN LET J=J+3: GO
TO 890
930 RETURN
940 IF a(w(j))=CODE "O" THEN LE
T r#="W": REM Victoria do Spectru
m
950 IF a(w(j))=CODE "X" THEN LE
T r#="L": REM Spectrum perde
960 RETURN

```

Vejamos agora a listagem completa de «Três em Linha», que permitirá ao leitor investigar um pouco a forma de «educar» uma máquina:

```

10 REM TIC-TAC
15 INK 7: PAPER 1: FLASH 0: BR
IGHT 0: BORDER 1

```

```

20 GO SUB 1100: REM inicializa
30 REM valores previos
40 FOR J=1 TO 9
50 LET A(J)=32
60 NEXT J
70 FOR J=1 TO 5
80 LET B(J)=0
90 NEXT J
100 LET count=0
110 LET r$=""
120 GO SUB 1070: REM imprimir
tabuleiro
130 REM ciclo principal
140 GO SUB 540: REM movimento
do Spectrum
150 GO SUB 1070: REM imprimir
tabuleiro
160 GO SUB 870: REM verificar
vitoria
170 IF r$<>"" THEN GO TO 240
180 GO SUB 980: REM aceitar
lanca humano
190 GO SUB 1070: REM imprimir
tabuleiro
200 GO SUB 870: REM verificar
vitoria
210 IF r$="" THEN GO TO 140
220 REM remainder ciclo
principal
230 REM *****
240 REM final go J000
250 GO SUB 1070: REM imprimir
tabuleiro
260 PRINT : PRINT
270 IF r$="U" THEN PRINT TAB 8;
"GGGG": FOR g=1 TO 4: BEEP .2;
5: BEEP .1; 10: BEEP .4; 9: NEXT g
: LET r$=1
280 IF r$="L" THEN PRINT TAB 8;
"GGGGGGGG": FOR g=1 TO 16 STEP 2:
BEEP .1; 9: BEEP .1; g+2: NEXT g:
LET r$=1
290 IF r$="O" THEN PRINT TAB 8;
"GGGGGGGGGG": FOR g=1 TO 4: BEEP .2;
5: BEEP .1; 9: BEEP .4; 9: NEXT g:
LET r$=0: GO TO 430
300 REM actualizar base de
acordamentos
310 FOR b=1 TO 5

```



```

0070 IF J<10 THEN LET J=J+1: GO
TO 0080
0080 LET H=0
0090 LET H=H+1
0100 LET X=INT (RND*9)+1: IF a(X)
) 0110 THEN GO TO 790
0120 IF H<100 THEN GO TO 690
0130 LET r#="0": REM a#pate
0140 REM movimento
0150 LET a(X)=CODE "0"
0160 LET count=count+1
0170 LET d(count)=X
0180 LET flag=0
0190 FOR J=1 TO 9
0200 IF a (J) =32 THEN LET flag=1
0210 NEXT J
0220 IF flag=0 AND r#="" THEN LE
T r#="0"
0230 REM a# today as positions
0240 REM a# day definition
0250 REM a#
0260 REM *****
0270 REM verificação victoria
0280 LET J=1
0290 IF a (#(J))=32 THEN LET J=J+
0
0300 IF J>20 THEN RETURN
0310 IF a (#(J))=a (#(J+1)) AND a (
#(J))=a (#(J+2)) THEN GO TO 040
0320 IF J<20 THEN LET J=J+3: GO
TO 0300
0330 RETURN
0340 IF a (#(J))=CODE "0" THEN LE
T r#="0": REM victoria do espectro
#
0350 IF a (#(J))=CODE "X" THEN LE
T r#="X": REM espectro perde
0360 RETURN *****
0370 REM lance humano
0380 PRINT : PRINT
0390 INPUT move
0400 IF move<1 OR move>9 THEN GO
TO 0310
0410 IF a (move) <>32 THEN GO TO 4
0420 LET a (move) =CODE "X"
0430 RETURN

```



```

4500 LET B=B+1
4600 LET MOVE=INT (RND*B+1)
4800 IF (MOVE)=32 THEN LET a (MOVE) =CODE "X": RETURN
4900 IF B<100 THEN GO TO 4520
4900 LET B="0"
4970 RETURN

```

Para substituir o adversário humano por este opositor incansável e estúpido, substitua simplesmente a linha 180 por GOSUB 4500.

The figure displays seven tic-tac-toe boards arranged in a roughly circular pattern. Each board is a 3x3 grid with horizontal and vertical lines. The boards contain 'X' and 'O' characters in different positions, representing various game states. Some boards show a win for 'X' or 'O', while others show a draw or an ongoing game.

Fig. 7

### 3

## UM PROGRAMA QUE RACIOCINA

De um programa que aprende passamos a SILOGISMO, um programa que raciocina. Dadas duas frases relacionadas entre si, SILOGISMO é capaz de deduzir uma terceira que contém informação não apresentada explicitamente à máquina.

O programa trabalha com silogismos. Um silogismo é um tipo de dedução. Aristóteles definiu as regras que determinam a validade de um silogismo. Geralmente este assume a seguinte forma:

A é um B  
C é um A  
Portanto, C é um B

As primeiras duas linhas de um silogismo são proposições, e a terceira é a sua conclusão.

Um cão é um animal  
Um animal é peludo  
Portanto, um cão é peludo

Antes de discutirmos o programa, e o que o justifica, em pormenor, vamos mostrar a forma como trabalha. Ignoremos o material que se encontra entre parêntesis antes da conclusão, dado que este é apenas incluído para permitir ao leitor observar o funcionamento do programa. Compreenderá facilmente a que corresponde este material depois de ter estudado a explicação do programa.

A pergunta «?» surgirá sempre que SILOGISMO espera

uma entrada do utilizador. «> OK» surge sempre depois de o programa ter aceite e compreendido a sua entrada.

```
UMA AGUIA E' UMA AVE
> OK.

UMA AVE E' UMA CRIATURA COM ASAS
> OK.

E' UMA AGUIA UMA CRIATURA COM AS
ASAS?
      (Procura AGUIA)
      (Encontra-se em 1,1)
> Sim
```

À medida que o programa corre, constrói uma base de dados constituída por proposições, que depois pode consultar em qualquer momento. Vejamos agora o par seguinte de proposições por nós experimentadas:

```
UMA AVE E' UM VOADOR
> OK.

E' UMA AGUIA UM VOADOR?
      (Procura AGUIA)
      (Encontra-se em 1,1)
> Sim

E' UM VOADOR UMA CRIATURA COM AS
ASAS?
      (Procura VOADOR)
      (Encontra-se em 1,5)
> Sim
```

SILOGISMO aceitará, e acrescentará à sua base de dados, qualquer proposição da seguinte forma:

UM ... E' UM ...

Esta proposição pode incluir «um», «uma», «o», e «a», dado que o programa está preparado para aceitar estes artigos. Nestas condições, qualquer das seguintes frases é válida:

```
UMA ... E' UMA ...
O ... E' UM ...
```



```

UM LOUCO É UM IDIOTA
      OK,

É TIM UM IDIOTA?
      (Procura TIM)
      (Encontra-se em 1,1)
      Sim

Tema a verificar? TIM
R 1 LOUCO

Tema a verificar? LOUCO
D 1 TIM
D 2 IDIOTA

Tema a verificar? IDIOTA
R 3 LOUCO

```

Se bem que seja possível levar SILOGISMO a apresentar algumas conclusões absurdas, em geral funciona de forma aceitável:

```

UM GALO É UM IDIOTA
      OK,

É TIM UM GALO?
      (Procura TIM)
      (Encontra-se em 1,1)
      Não

É UM GALO UM LOUCO?
      (Procura GALO)
      (Encontra-se em 1,4)
      Sim

```

SILOGISMO funciona com base num quadro alfanumérico bidimensional, Z\$, obtendo conclusões por consulta dos seus elementos.

É muito fácil compreender o que se passa se visualizarmos o que acontece ao dar entrada a proposições. Se escrevermos TIM É UM GÊNIO, o programa ignora É UM e usa TIM como título de uma categoria, colocando GÊNIO por baixo dela. Uma segunda proposição do tipo UM GÊNIO É UM IDIOTA permite ao programa abrir uma nova referência de título GÊNIO e com IDIOTA dependente dele. Quando se pergunta ao programa se TIM é um IDIOTA o programa

tenta primeiro descobrir se possui alguma categoria intitulada TIM. Ao descobri-la, procura os temas nela incluídos. Encontra assim GÊNIO. Em seguida procura uma categoria intitulada GÊNIO. Ao descobri-la, verifica que entre os temas nela incluídos se encontra IDIOTA. Nestas condições pode responder à pergunta É TIM UM IDIOTA?

É este o procedimento que ocorre qualquer que seja a série de frases introduzidas na máquina. Num quadro de 25 x 25 elementos existe espaço suficiente para incluir muitos temas, e talvez o leitor queira até guardar em fita magnética alguns dos ficheiros assim criados.

A série TIM É UM IDIOTA foi evidentemente tratada separadamente da série A ÁGUIA É UMA AVE. Para facilitar a compreensão do modo como SILOGISMO guarda, e depois acede, as proposições que lhe permitem chegar a conclusões, vejamos a forma como se encontram guardados os dados necessários a esta segunda série:

	1	2	3	4
1	ÁGUIA	AVE	ASAS	VOADOR
2	AVE	ÁGUIA	AVE	AVE
3		ASAS		
4		VOADOR		
5				

Quando o programa encontra um tema novo (sendo este tema o primeiro substantivo da proposição) começa pelo «início» do quadro, observando sucessivamente em 1,1, depois 1,2, em seguida 1,3, etc. Se escrevemos A ÁGUIA É UMA AVE no início da execução, como 1,1 está vazio, o programa guarda ÁGUIA em 1,1 e AVE em 2,1.

Em seguida o programa troca os dois substantivos, e abre uma categoria intitulada AVE que coloca em 2,1, indicando em seguida (em 2,2), ÁGUIA. Quando obtém outra proposição que utiliza uma categoria já criada, como UMA AVE É UMA CRIATURA COM ASAS, guarda a informação CRIATURA COM ASAS em 3,2, e abre uma nova categoria intitulada CRIATURA VOADORA em 3,1, guardando AVE por baixo (3,2).

E assim continua, guardando toda a informação que recebe de modo a poder acedê-la mais tarde. A última proposição que indicámos nesta execução foi UMA AVE É UM VOADOR, pelo que SILOGISMO colocou VOADOR na primeira posição vazia sob AVE (em 4,2) e abriu uma nova categoria VOADOR em 1,4, guardando AVE por baixo, em 2,4.

Quando se escreve um ponto de interrogação, a fim de pedir o conteúdo de um tema, o computador percorre a área do quadro correspondente aos nomes de temas (isto é, 1,1, 1,2, etc.) até o encontrar. Se chegar ao fim (ou seja, a 1,25) sem o descobrir, responderá que não dispõe de dados sobre o tema em causa. Se o encontrar (por exemplo, AVE, em 1,2), imprime o conteúdo correspondente: neste caso, ÁGUIA, CRIATURA COM ASAS e VOADOR.

Quando se trata de tomar uma decisão, por exemplo, É UMA ÁGUIA UM VOADOR (as decisões são provocadas pela primeira palavra da entrada) o programa começa por verificar se dispõe de alguma informação sobre o primeiro substantivo presente na frase. Se descobre que sim, indica o facto ao utilizador e em seguida determina quais as palavras que lhe estão associadas. Se encontra AVE (em 2,1) procura VOADOR. Descobre este termo em 1,4, e verifica que contém AVE em 2,4. Descobriu assim um elo comum entre as duas palavras (ÁGUIA e VOADOR), podendo nestas condições concluir que a resposta à pergunta é de facto SIM.

## **Voltando ao programa**

Vejamos agora o início do programa, onde SILOGISMO processa as entradas do utilizador. A linha 40 envia a execução



```

230 IF LEN C# <= 0 THEN GO TO 240
231 IF C#(1) TO 4) = "UNA " THEN LE
T C# = C#(5 TO )

```

A linha 80 detecta o «É» no início de uma frase, indicando que o utilizador está a fazer uma pergunta ao programa. Este facto envia a execução para a linha 480, onde se inicia a rotina que permite tomar decisões.

As linhas 90, 100 e 110 retiram os artigos da entrada, de tal modo que A\$ passa a começar pelo substantivo que será usado como título.

A rotina seguinte, de 120 a 230, divide a entrada em duas palavras, servindo as linhas 120 a 160 para obter o primeiro termo, e imprimindo «Não compreendo» (linha 170 em diante se a entrada não concorda com o formato definido. As linhas 180 a 230 extraem a segunda palavra. A linha 190 verifica se a frase contém outras formas verbais, permitindo a aceitação de «ERA» em vez de «É».

Tendo extraído as palavras importantes (e tendo atribuído a primeira a B\$ e a segunda a C\$) o programa passa a guardá-las na sua base de conhecimento. Não se esqueça de que esta secção do código é apenas usada para guardar informação. Utilizá-la depois, é feito por outra secção do programa.

O programa procura em seguida nos seus dados, a fim de verificar se (a) já dispõe de um ficheiro sobre o tema, e se não (b) se tem um espaço vazio onde possa guardar o novo tema. Se não existir espaço, é impressa uma mensagem apropriada (linha 310).

```

240) REM Guardar informacao
250) REM Verificar primeiro se
deacobre tema, antes de deaco-
brir espaço
260 LET n=0
270 LET n=n+1
280 IF n#(1,n) TO LEN b#)=b# TH
B# GO TO 320: REM existe tema
290 IF n#(1,n) TO 6)= " " TH
B# LET n#(1,n)=b#: GO TO 320
300 IF n<=0 THEN GO TO 270
310 INK B: PRINT "Nao possuo ma
12 espaço para guardar temas
": INK ?

```

A rotina seguinte, que se inicia na linha 320, é alcançada depois de o programa descobrir que dispõe já de dados sobre o tema (linha 280) ou encontrar espaço livre para os guardar e de o ter feito (linha 290).

```

320>REM O programa descobre ti-
tulo/tema
330 REM e coloca objecto sob
esse titulo
340 LET k=0
350 LET k=k+1
360 IF z#(k,n) TO LEN c#)=c# TH
EN GO TO 400: REM informacao
ja guardada sob esse titulo
370 IF z#(k,n) TO S1=" " TH
EN LET z#(k,n)=c#: GO TO 400
380 IF k<25 THEN GO TO 350
390 INK 5: PRINT "Nao disponho
de mais espaço para armazenamento
": INK 7
400 IF flag=1 THEN PRINT q#: IN
K 5: PRINT TAB 5;" OK.": INK 7:
GO TO 30: REM troca feita
410 REM trocar objecto e tema e
guardar novamente
420 LET flag=1
430 LET m#=b#
440 LET b#=c#
450 LET c#=m#
460 GO TO 250

```

Não é necessário que SILOGISMO guarde ÁGUIA sob o título AVE mais do que uma vez, mesmo que a proposição UMA ÁGUIA É UMA AVE seja indicada ao computador mais do que uma vez. A linha 360 garante que não são guardadas informações repetidas. Depois de o substantivo ter sido guardado, o programa troca este pelo título (linhas 420 a 450) e guarda-os novamente. Isto é, se guardou ÁGUIA como título, com AVE sujeito a ele, guarda agora AVE como título e ÁGUIA como substantivo dependente deste título.

Chegamos agora à parte verdadeiramente interessante (pelo menos em termos de qualidade dos resultados produzidos por SILOGISMO): a secção que toma decisões:

```

480>REM Conclusoes
490 REM Primeiro dividir en-
trada

```

```

0000 LET F$=S$: IF S$( TO 3)="E"
0001 THEN LET S$=S$(4 TO ): REM tirar verbo
0002 IF S$( TO 4)="ERA " THEN LE
0003 T S$=S$(5 TO ): REM tirar verbo
0004 IF S$( TO 2)="O " OR S$( TO
0005 )="A" THEN LET S$=S$(3 TO ): R
0006 EM tirar artigo se existir
0007 IF S$( TO 3)="UM " THEN LET
0008 S$=S$(4 TO ): REM tirar artigo
0009 se existir
0010 IF S$( TO 4)="UMA " THEN LE
0011 T S$=S$(5 TO ): REM tirar artigo
0012 se existir
0013 REM obter primeira palavra
0014 (F$)
0015 LET X=LEN S$
0016 LET N=0
0017 LET N=N+1
0018 IF S$(N)=" " THEN LET F$=S$
0019 ( TO N-1): GO TO 0000
0020 IF N=X THEN GO TO 0000
0021 PRINT TAB 6; INK 5;"Nao com
0022 preendo": GO TO 00

```

Em primeiro lugar é eliminada da entrada a palavra «É», juntamente com os artigos se estes existirem. Esta secção de código obtém a primeira palavra, e iguala-a a F\$. A secção seguinte extrai a segunda palavra, tornando-a igual a S\$.

```

0000 REM obter segunda palavra -
0001 S$
0002 LET S$=S$(N TO )
0003 IF LEN S$<=0 THEN GO TO 0000
0004 IF S$(1)=" " THEN LET S$=S$
0005 (2 TO ): GO TO 0001: REM eliminar
0006 primeira palavra se existir
0007 IF S$( TO 2)="O " OR S$( TO
0008 )="A" THEN LET S$=S$(3 TO ): R
0009 EM tirar artigo se existir
0010 IF S$( TO 3)="UM " THEN LET
0011 S$=S$(4 TO ): REM tirar artigo
0012 se existir
0013 IF S$( TO 4)="UMA " THEN LE
0014 T S$=S$(5 TO ): REM tirar artigo
0015 se existir
0016 IF S$(LEN S$ TO )="?" THEN
0017 LET S$=S$( TO LEN S$-1): REM tir
0018 ar sinal de interrogacao

```

```

600 PRINT "SIM"
610 PRINT "NÃO"
620 PRINT "NÃO"
630 PRINT "NÃO"
640 PRINT "NÃO"
650 PRINT "NÃO"
660 PRINT "NÃO"
670 PRINT "NÃO"
680 PRINT "NÃO"
690 PRINT "NÃO"
700 PRINT "NÃO"
710 PRINT "NÃO"
720 PRINT "NÃO"
730 PRINT "NÃO"
740 PRINT "NÃO"
750 PRINT "NÃO"
760 PRINT "NÃO"
770 PRINT "NÃO"
780 PRINT "NÃO"
790 PRINT "NÃO"
800 PRINT "NÃO"
810 PRINT "NÃO"
820 PRINT "NÃO"
830 PRINT "NÃO"
840 PRINT "NÃO"
850 PRINT "NÃO"
860 PRINT "NÃO"
870 PRINT "NÃO"
880 PRINT "NÃO"
890 PRINT "NÃO"
900 PRINT "NÃO"
910 PRINT "NÃO"
920 PRINT "NÃO"
930 PRINT "NÃO"
940 PRINT "NÃO"
950 PRINT "NÃO"
960 PRINT "NÃO"
970 PRINT "NÃO"
980 PRINT "NÃO"
990 PRINT "NÃO"

```

O programa indica ao utilizador aquilo que procura (na linha 630), e se encontra diz-nos em que ponto da tabela foi guardada a informação (linha 660). Se não encontra a segunda palavra informa-nos do facto (linha 680), voltando em seguida ao ciclo principal.

```

700 PRINT "NÃO"
710 PRINT "NÃO"
720 PRINT "NÃO"
730 PRINT "NÃO"
740 PRINT "NÃO"
750 PRINT "NÃO"
760 PRINT "NÃO"
770 PRINT "NÃO"
780 PRINT "NÃO"
790 PRINT "NÃO"
800 PRINT "NÃO"
810 PRINT "NÃO"
820 PRINT "NÃO"
830 PRINT "NÃO"
840 PRINT "NÃO"
850 PRINT "NÃO"
860 PRINT "NÃO"
870 PRINT "NÃO"
880 PRINT "NÃO"
890 PRINT "NÃO"
900 PRINT "NÃO"
910 PRINT "NÃO"
920 PRINT "NÃO"
930 PRINT "NÃO"
940 PRINT "NÃO"
950 PRINT "NÃO"
960 PRINT "NÃO"
970 PRINT "NÃO"
980 PRINT "NÃO"
990 PRINT "NÃO"

```

A secção seguinte do programa tira conclusões. No início, nas linhas 700 a 730; responde SIM se a pergunta foi feita exactamente da mesma forma usada para a entrada de infor-

mações. Isto é, se tiver perguntado É UMA ÁGUIA UMA AVE e anteriormente se tiver indicado expressamente que UMA ÁGUIA É UMA AVE, esta primeira parte do programa descobrirá o facto e responderá SIM.

A secção seguinte: entre 710 e 800, procura descobrir a palavra usando o método que já descrevemos anteriormente, chegando novamente a uma conclusão e imprimindo-a: SIM num dos casos (linha 850) ou NÃO (linha 820) no outro.

```

9010>REM Verificar conteúdo de
frase sobre particular
9020>INK @; INPUT "Tema a Verifi:
9030>INK #
9040>INK ?; LET t=0
9050>WHILE t<N: t=t+1
9060>PRINT "t: ",t; TO LEN (t)=# TH
9070>PRINT "t: ",t; THEN GO TO 940
9080>PRINT "Nao possui da d
9090>GO TO 930#
9100>PRINT "t: ",t;
9110>PRINT "t: ",t; TO LEN (t)=# T
9120>PRINT "t: ",t; THEN GO TO 940#(
k,t)
1000>IF k=# THEN GO TO 1000
1030>GO TO 90

```

Esta última parte do programa é aquela que nos indica aquilo que o programa guardou sob cada título específico.

Em seguida podemos observar a listagem completa de SILOGISMO, facilitando o estudo deste programa pelo leitor.

```

10>REM SILOGISMO
110>REM OKI OF LOGICO: FLESH @; BR
120>PRINT @; "LOGICO: FLESH @; BR";
INK ?
30>GO SUB 1000: REM Inicializa
7
30>PRINT " INPUT #
40>LET t=0
50>PRINT "t: ",t; THEN GO TO 910
60>LET t=t+1 THEN STOP
70>REM Nota: AVE existe um esp
80>@:AVE:AVE:AVE:AVE:AVE:AVE:AVE:
1100>AVE:AVE:AVE:AVE:AVE:AVE:

```

```

700 IF LEN(C) < 3 THEN GO TO 110
710 IF LEN(C) < 3 THEN GO
TO 140
720 IF LEN(C) < 4 THEN GO
TO 140
730 IF LEN(C) < 4 THEN GO
TO 140
740 IF LEN(C) < 4 THEN GO
TO 140
750 IF LEN(C) < 4 THEN GO
TO 140
760 IF LEN(C) < 4 THEN GO
TO 140
770 IF LEN(C) < 4 THEN GO
TO 140
780 IF LEN(C) < 4 THEN GO
TO 140
790 IF LEN(C) < 4 THEN GO
TO 140
800 IF LEN(C) < 4 THEN GO
TO 140
810 IF LEN(C) < 4 THEN GO
TO 140
820 IF LEN(C) < 4 THEN GO
TO 140
830 IF LEN(C) < 4 THEN GO
TO 140
840 IF LEN(C) < 4 THEN GO
TO 140
850 IF LEN(C) < 4 THEN GO
TO 140
860 IF LEN(C) < 4 THEN GO
TO 140
870 IF LEN(C) < 4 THEN GO
TO 140
880 IF LEN(C) < 4 THEN GO
TO 140
890 IF LEN(C) < 4 THEN GO
TO 140
900 IF LEN(C) < 4 THEN GO
TO 140
910 IF LEN(C) < 4 THEN GO
TO 140
920 IF LEN(C) < 4 THEN GO
TO 140
930 IF LEN(C) < 4 THEN GO
TO 140
940 IF LEN(C) < 4 THEN GO
TO 140
950 IF LEN(C) < 4 THEN GO
TO 140
960 IF LEN(C) < 4 THEN GO
TO 140
970 IF LEN(C) < 4 THEN GO
TO 140
980 IF LEN(C) < 4 THEN GO
TO 140
990 IF LEN(C) < 4 THEN GO
TO 140

```



```

6000 PRINT TAB 6; INK 5; "Neo com
6001 PRINT TAB 6; GO TO 60
6002 REM Order sequencia de letras -
6003 #
6004 LET W# = W#(5 TO )
6005 IF LEN W# = 1 THEN GO TO 6008
6006 IF W#(1) = " " THEN LET W# = W#
(6007 TO ) GO TO 6001: REM eliminar
6008 LET W#(1 TO W#) = "O " OR W#(1 TO
) = " " THEN LET W# = W#(2 TO ): R
6009 IF W#(1) = " " THEN LET W# = " "
6010 LET W#(1 TO ) = "UM " THEN LET
W# = W#(2 TO ): REM tirar artigo
6011 IF W#(1) = " " THEN LET W# = " "
6012 IF W#(1 TO ) = " " THEN
LET W# = W#(2 TO LEN W# - 1): REM tir
6013 IF W#(1 TO ) = " " THEN
6014 PRINT TAB 6; INK 5; "Pro
6015 #";
6016 LET X = 0
6017 LET X = X + 1
6018 IF W#(1, X) TO LEN W# = W# TH
6019 PRINT TAB 6; INK 5; " "
6020 IF W#(1, X) = " " THEN GO TO 6025
6021 PRINT TAB 6; INK 5; "Neo end
6022 TO 60"; PRINT TAB 6; W#
6023 GO TO 6000
6024 LET W# = 1
6025 LET W# = W# + 1
6026 IF W#(1, X) = W# THEN INK 5; "
6027 PRINT TAB 6; W#"; INK 7; GO TO
6028
6029 IF W#(1, X) THEN GO TO 6024
6030 LET W# = 1
6031 LET W# = W# + 1
6032 LET W#(1, X)
6033 LET W# = W# + 1
6034 IF W#(1, X) = W# THEN GO TO 60
6035
6036 IF W#(1, X) THEN GO TO 6032
6037 IF W#(1, X) THEN GO TO 6032
6038 PRINT TAB 6; INK 5; "> Neo";
6039 GO TO 6000
6040 LET W# = 1

```



## II PARTE PROCURANDO...

### 4

## ÁRVORES DE DADOS

Nesta parte do livro vamos desenvolver um jogo de tipo damas. Usá-lo-emos para discutir algumas ideias da procura de informação organizada em árvore, permitindo ao computador actuar com um certo grau de inteligência fazendo a procura de dados segundo linhas de opções relacionadas entre si, e escolhendo entre estas a que considera melhor.

Procurando por árvores de opções é vulgar em programas de resolução de problemas. Geralmente usam-se modificações, muitas bastante importantes, como a «poda» da árvore de modo a evitar o estudo dos seus ramos irrelevantes ou a execução deste estudo até uma profundidade desnecessária; estas modificações permitem parar o processo, evitando que demore um tempo excessivo, mas a ideia básica da procura em árvore é ainda fundamental para a resolução de problemas.

### **A que se chama árvore?**

Uma árvore deste tipo cresce do mesmo modo que qualquer outra, com a diferença de ser invertida. Consideremos A, no diagrama seguinte, como ponto de partida da procura executada pelo computador. Os «ramos» (designados por B, C e D) que dela saem representam decisões válidas (ou lances legais, se o programa resolve um jogo). Os ramos mais pequenos que irradiam dos anteriores (E, F, etc.) são implicações do ramo principal.

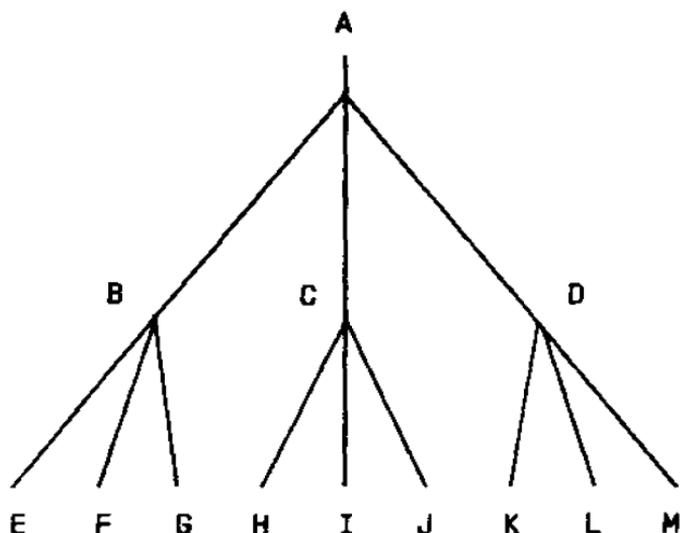


Fig. 8

Se a árvore representa um mecanismo de procura de lances num jogo de xadrez, por exemplo, A representará o movimento de um cavalo. O programa segue todas as implicações desse movimento. E corresponde, digamos, a levar o cavalo a atacar peças adversárias. A resposta E corresponde ao adversário eliminar o cavalo, F pode indicar o apoio à peça ameaçada através de outra, e G pode corresponder ao afastamento da peça atacada. E, F e G são depois subdivididos, em N, O, etc., o que cobre todas as respostas possíveis a cada acção.

É fácil verificar que a procura pode alcançar dimensões astronómicas, a menos que exista algum modo de a controlar. Só num jogo muito simples, como o «Três em Linha», pode o programa examinar todos os ramos de todas as árvores antes de escolher o movimento mais apropriado.

Noutros programas, um ramo pode ser examinado até uma determinada profundidade (mais adiante discutiremos esta «profundidade») em vez de o ser até ao fim, sendo guardado em memória o resultado deste exame.

## Processamento paralelo

Um outro método consistiria em examinar um pouco um dos ramos, voltando depois atrás e passar ao ramo seguinte, e assim por diante; finalmente, voltar aos ramos que parecem mais prometedores e investigá-los mais a fundo. Um ramo, por exemplo, que considerasse a possibilidade de o adversário sacrificar a dama para capturar um peão, não mereceria maior exame. Qualquer ramo que levasse o adversário – na opinião do mecanismo de avaliação do programa – a enfraquecer a sua posição, poderia ser abandonado no momento em que este facto fosse descoberto, sendo o tempo de processamento aplicado em ramos mais prometedores.

Quando se desenvolvem programas de inteligência artificial, vale a pena começar a pensar em termos de árvores de procura, dado que é provável que estas sejam usadas de algum modo. A árvore pode crescer desmesuradamente, em particular se não trabalharmos num domínio bastante restrito – como faremos no programa seguinte, Parte III – ou não temos a certeza quanto aos critérios que permitirão ao programa fazer escolhas.

Desenvolvemos um jogo de tipo damas nesta parte do livro a fim de demonstrar alguns aspectos da procura em árvore, aqui um tanto primitiva. Como é natural, é necessário saber-se jogar para se poder compreender a discussão do jogo. Dentro em pouco seguiremos a execução de um jogo, mas por enquanto vamos concentrar a nossa atenção nos primeiros movimentos.

Vejamos a aparência do tabuleiro no início do jogo:

COMPUTADOR O HUMANO X

```
12345678
- - - - -
O O O O O O
X X X X X X
. . . . .
. . . . .
O O O O O O
X X X X X X
- - - - -
12345678
```

A pontuação (actualmente zero) tanto da máquina como do utilizador é impressa por cima do tabuleiro. Cada jogador começa com oito peças (em vez das 12 normais). As peças do computador encontram-se na parte superior do tabuleiro (os C's), e as do utilizador na parte inferior daquele (os H's). O computador joga de cima para baixo, e o utilizador de baixo para cima.

Os pontos representam os quadrados negros do tabuleiro. As peças movem-se da maneira normal, isto, em diagonal de um quadrado negro para outro. Cada peça encontra-se portanto sobre um ponto negro, que será revelado quando aquela o abandonar. Cada um dos pontos representa uma posição para onde pode ser movida uma peça.

Como já disse, as peças movem-se como é habitual no jogo de damas (Damas Inglesas, entenda-se). As capturas são executadas da forma habitual, saltando por cima da posição adversária para um quadrado vazio que esteja atrás dela. Não se aceitam porém neste jogo saltos múltiplos.

### **Não há «damas» nem movimentos para trás**

O objectivo do jogo consiste em alcançar uma pontuação de 5 antes do adversário. Existem duas maneiras de obter um ponto. Uma, como é óbvio, consiste em capturar uma peça inimiga. A outra consiste em atingir a primeira linha do adversário, isto é, o lado oposto do tabuleiro. No jogo normal de damas, isto resultaria na transformação da peça, o que lhe permitiria voltar atrás e mover-se em qualquer direcção. Nesta adaptação, a peça desaparece do tabuleiro (o que significa que neste jogo não haverá damas propriamente ditas, nem existirão peças em movimento para trás em qualquer situação de jogo).

Se se salta por cima de uma peça adversária e se termina o movimento após a captura na linha traseira, obtêm-se dois pontos em vez de um. No exemplo de jogo que apresentaremos em seguida o leitor poderá observar um exemplo disto. O computador dir-lhe-á quais os movimentos que está a considerar em cada momento, o que facilitará a observação da «inteligência» da máquina. No início, ou seja, na situação que podemos

observar no tabuleiro já apresentado, existem sete movimentos possíveis de abertura. O computador descobre quais os lances legais, imprime-os no visor, executa um deles (os números são formados pelo algarismo do lado do tabuleiro seguido pelo que se encontra em cima ou em baixo):

```

      B B N N O O B B 7 7  B B B B B B B B
      B B N N O O B B 7 7  B B B B B B B B
      B B N N O O B B 7 7  B B B B B B B B
      B B N N O O B B 7 7  B B B B B B B B
      B B N N O O B B 7 7  B B B B B B B B
      B B N N O O B B 7 7  B B B B B B B B
      B B N N O O B B 7 7  B B B B B B B B
      B B N N O O B B 7 7  B B B B B B B B
  
```

Estes números referem-se portanto àqueles que compõem o quadro onde se encontra guardado o estado actual do tabuleiro. Vejamos a correspondência entre as posições deste e os números referidos:

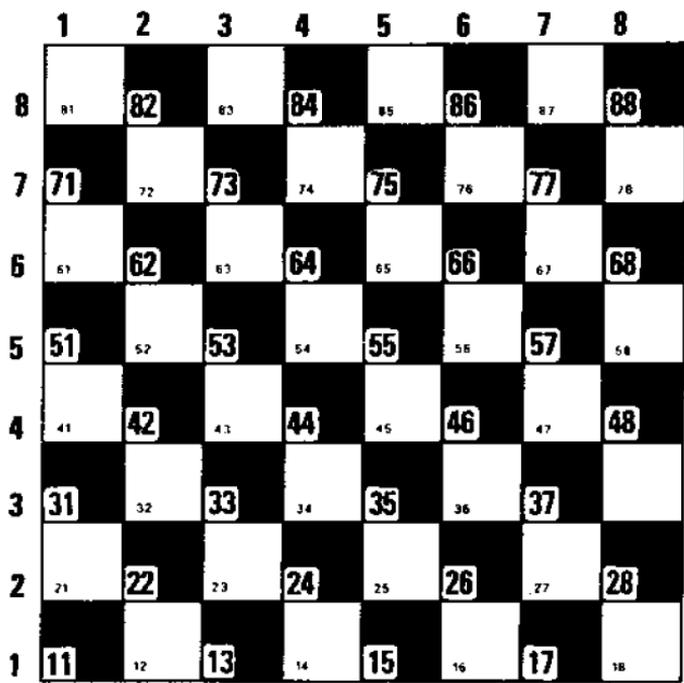


Fig. 9

É fácil ver que a numeração não é consecutiva, nem sequer começando por 1. No entanto, este tabuleiro é muito mais fácil de usar, em termos do computador, do que um outro cujos quadrados negros estivessem numerados entre um e trinta e dois.

Um computador necessita de saber onde se encontram as margens do computador, e são os números «em falta» que lhe dão essa informação. Por exemplo, se tentar passar de 48 para 59, o valor guardado no elemento 59 do quadro (zero neste caso) indicará que esta posição está fora do tabuleiro.

A segunda vantagem, e muito mais importante, deste sistema, refere-se à coerência da especificação dos movimentos, qualquer que seja a posição do tabuleiro onde ocorrem. Vou explicar o que pretendo dizer com isto. Observemos uma lista dos movimentos que o computador considera de início, e notemos as simples relações matemáticas que envolvem a posição de partida e a de chegada:

71 a 62	- 9
73 a 64	- 9
73 a 62	- 11
75 a 66	- 9
75 a 64	- 11
77 a 68	- 9
77 a 66	- 11

A diferença entre a posição de partida e a final é de nove ou onze. E se compararmos os números dados com o tabuleiro, verificaremos que os movimentos para baixo e para a esquerda são sempre menos onze, enquanto os para baixo e para a direita são sempre menos nove.

Este facto é verdadeiro para qualquer posição do tabuleiro. Todos os movimentos que não envolvam capturas devem variar de nove ou onze relativamente ao quadrado inicial. O leitor compreende certamente a vantagem disto; se experimentar um tabuleiro cujas casas estejam numeradas entre um e trinta e dois verificará os graves problemas que daí decorrem.

Mais ainda, o computador pode com este tabuleiro tomar decisões com bastante facilidade. Consideremos o quadrado

onde o computador «está» como sendo indicado por um número X. Se existir uma peça «humana» em X-9, e se X-18 estiver vazio, a máquina sabe que pode capturar saltando para X-18. A sua pontuação será assim aumentada, e X-9 passará a ser um quadrado vazio.

Por outro lado, e é aqui que entra a «inteligência», o computador pode tentar descobrir o que acontecerá depois de cada lance, determinando quais as posições que o utilizador humano poderá tentar ocupar. Se existe uma peça «humana» em X-27, o computador pode considerar – talvez com razão – que o lance humano seguinte consistirá em capturar a peça do computador que nesse momento se encontra em X-18, movendo-se para X-9.

A posição após a captura (não esqueça que o computador estará agora em X-18) encontra-se também potencialmente sob a ameaça de X-25, se X-7 estiver vazia. A sequência do jogo torna-se talvez um pouco confusa a partir deste ponto, pelo que sugiro ao leitor que tente seguir a situação no tabuleiro anteriormente impresso, ou num tabuleiro real.

O computador pode também verificar quando uma das suas peças está ameaçada. Imaginemos, uma vez mais, que o computador está na posição X. O utilizador humano desloca uma peça para  $X - 9$ . Sabe que  $X + 9$  está vazia, pelo que o utilizador pode de facto passar a esta posição no lance seguinte, capturando a peça do computador em X. Pode contrariar-se isto movendo uma peça para  $X + 9$ , ou – se a medida anterior não for possível – deslocando uma peça que ameace  $X + 9$ . Talvez isto persuada o jogador humano a não realizar a captura.

Neste jogo não é obrigatório capturar peças. Pode preferir-se não o fazer, sabendo que é possível obter dois pontos com a mesma peça um pouco depois, noutra captura que termine na linha traseira.

### **Aprofundando...**

A procura por árvore não vai demasiado longe, se bem que permita já um jogo razoável, com uma percentagem satisfatória de vitórias (incluindo a que é usada para demonstrar

o modo de funcionamento do programa nesta parte). Talvez o leitor se sinta tentado a pensar que, se a árvore for construída com mais pormenor e a procura for menos superficial, o programa acabe por jogar de uma forma perfeita.

Este jogo é menos complexo do que o verdadeiro jogo de damas, não admitindo saltos múltiplos nem dobra de damas, não sendo portanto de pôr de lado a hipótese de desenvolver um sistema perfeito. Em última análise, deveria ser possível criar um trajecto que o computador pudesse seguir de forma a jogar bastante bem.

Poderíamos fazê-lo, recorrendo a um método um tanto semelhante ao «computador de caixas de fósforos» apresentado num dos capítulos anteriores. Isto é, poderíamos examinar todos os movimentos possíveis, de qualquer jogo possível, e analisar cada um deles em profundidade. De facto, dispomos de computadores incansáveis à nossa disposição, e podemos dar-lhes o trabalho aborrecido...

Pensem um pouco melhor no assunto. Sabemos (porque o computador nos disse já isso) que a máquina pode realizar um de sete movimentos no início do jogo. A nossa árvore, com A no topo, começará por ter os ramos B, C, D, E, F, G e H, imediatamente no início. O jogador humano também dispõe de sete movimentos possíveis ao principiar o jogo, pelo que cada um dos sete ramos necessitará também de sete sub-ramos (ou «nodos», como se chama aos pontos onde os ramos ligam). Depois de cada jogador ter executado o seu movimento, e mesmo antes de o programa ter começado a analisar as respostas possíveis ao primeiro lance humano, teremos já quarenta e nove direcções possíveis a considerar.

A situação começa então a piorar. Agora que uma das peças foi deslocada da posição inicial, dispomos de dois movimentos possíveis (um em alguns casos, se o primeiro movimento foi executado em qualquer dos lados) e mais seis (talvez o primeiro movimento tenha bloqueado o movimento de uma peça que se encontre ainda na posição de partida). Isto significa que temos agora  $49 \times 8$  ramos para considerar, mesmo antes de o jogador humano ter executado o seu segundo lance.

Uma árvore completa para o jogo de damas conteria cerca de 10 elevado a 40 nodos. Considerando uma velocidade de

três milhões de nodos por segundo (que obrigaria a usar um computador bastante rápido), esta árvore demoraria 10 elevado a 21 anos para ser analisada.

Sugerimos já que um modo de «podar» a árvore consiste em abandonar os ramos menos interessantes (como aqueles que colocam a hipótese de o adversário fazer jogadas perigosas sem necessidade), deixando assim mais tempo livre para os ramos úteis. Foi também sugerido que o computador poderia estudar um pouco um ramo, até uma certa profundidade, tomando nota do resultado a que chegaria, e passando em seguida aos outros ramos, abandonando depois os mais fracos e concentrando-se nos mais prometedores.

Para conseguir isto, temos de conseguir atribuir um valor à posição encontrada. Este valor pode ser numérico (baseado em algo semelhante ao que acontece no programa de damas de Samuel – já discutido anteriormente), ou basear-se num esquema hierárquico que ordene os movimentos escolhidos e decida não seguir a maioria dos ramos. Como poderemos ver dentro em pouco, é este o método utilizado no programa aqui apresentado.

### **Mini-maxing**

Teremos no entanto de observar um pouco melhor as árvores, na nossa tentativa de aperfeiçoar o jogo do computador. O programa utiliza uma forma grosseira de uma técnica designada por «mini-maxing» que nos permite «podar» os ramos que surgem em número cada vez maior.

Para usar esta técnica, no entanto, o computador deve poder atribuir valores numéricos às posições que descobre.

Imagine que o computador dispõe de três opções, e que cada uma delas consiste num movimento de uma peça diferente. O valor atribuído ao movimento pode consistir, em parte, na distância a que a peça ficará do centro após o movimento; no facto de ameaçar uma peça inimiga, imediatamente ou depois de um segundo movimento; no facto de o quadrado para onde o movimento será executado se encontrar ameaçado; no facto de o movimento executar uma captura ou atingir qualquer

outro objectivo (por exemplo, atingir a primeira linha inimiga).

Vejamos então a nossa árvore, com os movimentos B, C e D nas extremidades dos primeiros três ramos, com as pontuações que lhes são atribuídas:

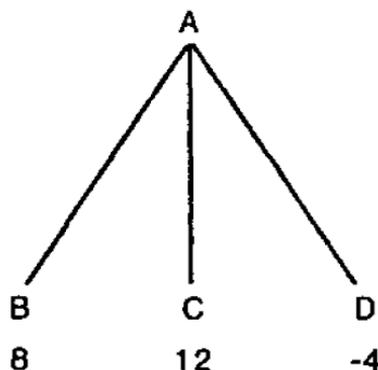


Fig. 10

É fácil ver que C tem o maior valor, pelo que este nodo parece constituir a escolha óbvia. Não se esqueça de que esta pequena árvore se baseia na situação depois de o computador ter executado o seu lance. No entanto, se a máquina procurar avaliar a série seguinte de ramos, tendo em conta e avaliando as possíveis respostas do jogador humano, poderá encontrar o seguinte:

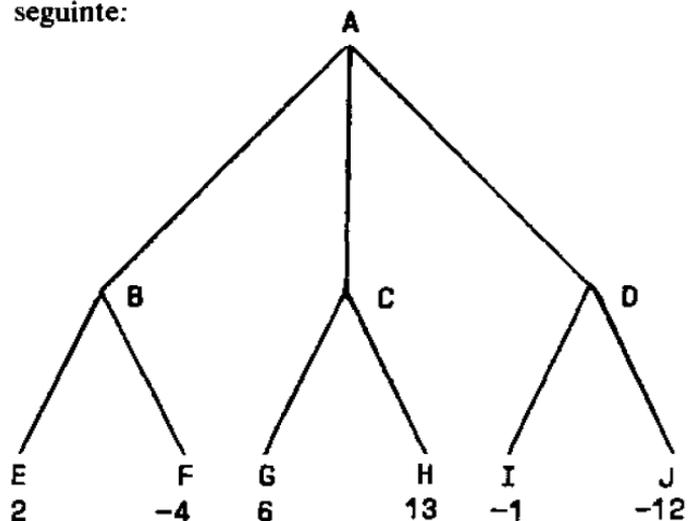


Fig. 11

Os valores aqui atribuídos aos nodos E a J são definidos em função da avaliação das posições do tabuleiro *pele jogador*. O melhor movimento a realizar pelo computador pode ser aquele que dá ao humano a posição mais fraca. A escolha deve portanto ser aquela que dá ao computador a maior pontuação possível, ao mesmo tempo que minimiza a força da posição adversa. É aqui que a expressão «mini-maxing» («mini-maximização») surge.

Considerando que o computador não aprofunda mais o estudo da situação, tentando determinar o estado em que fica após cada um dos movimentos possíveis do adversário (e de determinar talvez as respostas do jogador à sua resposta), convirá talvez escolher o movimento B. Isto deixa-o numa posição relativamente forte (valor 8), se bem que não o deixe na mesma posição em que ficaria se escolhesse o movimento C (valor 12).

O computador parte do princípio de que o jogador fará o melhor movimento nas circunstâncias. Se o computador tivesse jogado C, de modo a obter imediatamente a melhor posição possível, deixaria ao jogador humano a possibilidade de jogar H, terminando com um valor de 13. Em vez disso, jogando B, deixa apenas ao humano - na melhor das hipóteses - a possibilidade de atingir o valor 2, usando o nodo E.

Já disse anteriormente que este programa funciona por atribuição de um valor a cada movimento possível, segundo uma determinada hierarquia. Escolhe os seus movimentos referindo-se a esta hierarquia, o que coloca um valor nos movimentos possíveis pela ordem indicada mais adiante. Executará sempre um movimento que se encontre mais acima na árvore, se tal lhe for possível.

Encontra-se presente um certo grau de «mini-maxing». O programa pensa apenas em termos de vantagem material, isto é, procura continuamente minimizar o número de peças possuídas pelo adversário, e preservar as suas próprias.

Por exemplo, o programa pode observar duas capturas possíveis, uma das quais o exporá subseqüentemente a uma captura adversária. Como é natural, executará o movimento que o deixa em posição mais forte (mantendo ainda em jogo a peça que executa a captura) e ignorará o movimento que

permitirá ao adversário realizar também uma captura.

A hierarquia de movimentos usada neste programa a fim de «podar» a árvore dos movimentos possíveis, evitando a perda de tempo no estudo de ramos inúteis, é a seguinte. São guardados em memória todos os movimentos que concordem com as descrições que se seguem:

- Capturas seguras que ameaçam peças «humanas», e que não expõem outras peças a possível captura.
- Capturas que deixam em completa segurança as peças que as executam.
- Outras capturas.
- Movimentos de protecção a peças ameaçadas.
- Rejeição aleatória dos movimentos citados, se a execução do movimento expuser uma outra peça à captura pelo adversário.
- Movimentos não envolvendo captura para a linha traseira inimiga.
- Movimentos não envolvendo captura que não expõem o computador a qualquer risco.
- Qualquer movimento legal.

Se a máquina encontra quaisquer movimentos de captura, não procura descer mais na análise da árvore. De facto, «corta» automaticamente os ramos com nodos mais baixos, nem sequer os considerando. Isto pode parecer demasiado restritivo, e significa certamente que o programa não é capaz de jogar com qualquer tipo de estratégia global, mas mesmo assim permite obter resultados surpreendentemente bons (auxiliado, evidentemente, pela natureza simples do jogo).

Torna-se assim possível compreender, espero, que esta ordenação hierárquica de movimentos minimiza o número de possibilidades a explorar. Ao examinar o programa, o leitor verificará que primeiro o computador examina o tabuleiro, posição a posição, tentando encontrar capturas possíveis, que são depois guardadas sob a forma «Bom, seguro», «Seguro», ou «Captura».

Se as áreas de armazenamento de dados (áreas dedicadas) se encontram vazias, o computador varre novamente o tabuleiro,

tentando ver se alguma das suas peças se encontra ameaçada por peças «humanas».

Se esta procura não permite determinar um movimento, o nosso computador tenta descobrir se dispõe de alguma peça na segunda linha adversária que possa ser movida para a primeira linha, aumentando assim a pontuação. Possui uma ordem pre-determinada para proceder deste modo, garantindo que no caso de se encontrarem duas peças na segunda linha, ambas podendo ser movidas, o será a que se encontra mais perto do centro, dado que está provavelmente mais sujeita a um ataque do que as peças que se encontram junto aos lados. Esta presunção é obviamente grosseira, mas garante pelo menos que a máquina não desloca a primeira peça que encontra.

Se ainda não foi executado qualquer movimento, o tabuleiro é novamente analisado, procurando-se determinar quaisquer movimentos seguros (isto é, movimentos que não exponham a peça à captura inimiga), sendo tais movimentos imediatamente guardados em memória. Se forem encontrados movimentos nestas condições, o computador escolhe aleatoriamente entre eles.

Se esta procura ainda não permitiu determinar um lance, o computador procura aleatoriamente posições no tabuleiro, tentando descobrir qualquer lance legal. Se não encontra algum, concede o jogo. Vamos examinar brevemente as principais partes da listagem, e identificaremos as subrotinas que executam as tarefas especificadas.

A propósito, o leitor notará que as várias observações do tabuleiro feitas pelo programa parecem ser um desperdício. Não seria possível ao programa analisar o tabuleiro uma só vez? A resposta é evidentemente sim; mas em muitos casos isto representaria um considerável esforço, obrigando a procurar, e armazenar, movimentos que nunca viria a utilizar. Talvez no entanto o leitor queira alterar o programa, ou escrever um novo, de tal modo que esta análise seja executada uma só vez, verificando então os efeitos que isto poderá ter nos tempos de execução.

É óbvio que o sistema hierárquico que determina o valor relativo dos movimentos pode ser combinado, para maior flexibilidade, com uma função de avaliação. Isto pode conseguir efeitos semelhantes aos obtidos por Samuel, como a indicação

do número de peças de cada jogador, o número de peças sob ataque directo ou que controlam o centro.

Uma alternativa ao uso de uma função de avaliação consistiria em o computador guardar todas as posições possíveis no tabuleiro, atribuindo a cada uma delas um determinado valor. Mas esta «solução», tal como a ideia de construir árvores completas cobrindo todas as soluções possíveis do jogo, envolve o uso de números astronomicamente grandes. Existem cerca de 10 elevado a 40 posições possíveis no caso das damas, e mesmo no caso desta variante do jogo o número de posições seria desmedido.

Não existem quaisquer regras simples susceptíveis de serem aplicadas ao desenvolver funções de avaliação durante a escrita de jogos de tabuleiro. Deve definir-se intuitivamente a função, e depois modificá-la tendo em conta os resultados a que conduz na prática. A vantagem de um jogo simples, como o «Três em Linha», é que se torna possível pôr o programa em jogo contra um adversário aleatório ou contra um inteligente. Os resultados dos jogos podem ser usados para modificar automaticamente a função de avaliação, podendo-se também jogar um grande número de jogos com uma versão da função, comparando os resultados com os obtidos com variantes dela. Não é muito simples programar um adversário aleatório que jogue repetidamente contra o programa no caso de jogos mais complexos como o xadrez.

## **Peso dos elementos**

A tarefa é facilitada pelo facto de os elementos que constituem a função de avaliação serem geralmente multiplicados por alguns factores diferentes. A modificação da função de avaliação pode resumir-se então à modificação desses factores, em vez de obrigar ao acrescento ou eliminação de elementos.

Tentarei agora explicar o que disse no último parágrafo com um exemplo concreto. A experiência demonstrou aos jogadores de xadrez que o valor relativo das peças pode ser expresso, de uma maneira grosseira, do seguinte modo:

PEÃO - 1  
 BISPO - 3  
 CAVALO - 3, 5  
 TORRE - 5  
 DAMA - 9  
 REI - infinito

Pode criar-se uma primeira avaliação, somando simplesmente o valor das peças de que se dispõe, e subtraindo o valor das peças adversárias, de modo a obter uma medida da força relativa do seguinte tipo:

$$\begin{aligned}
 \text{Força} = & n \times bP + 3 \times n \times bB + 3,5 \times n \times bC + \\
 & + 5 \times n \times bT + 9 \times n \times bD - (N \times pP + \\
 & + 3 \times N \times pB + 3,5 \times N \times pC + 5 \times N \times pT + \\
 & + 9 \times N \times pD)
 \end{aligned}$$

Com esta avaliação inicial, pode talvez escrever-se um grosseiro jogo de xadrez, desejoso de trocar peças quando a força se torna positiva, e que evita fazê-lo noutras condições. Jogando contra um tal programa, e usando esta função para ajudar a decidir quais os ramos a investigar (usando «mini-maxing»), pode verificar-se se, por exemplo, o papel do Cavalo foi sobrestimado, conduzindo às necessárias correcções. Pode por exemplo diminuir-se o valor dos Cavalos para 3.

O valor da função de avaliação pode ser aumentado se se incorporar o factor «mobilidade» (por exemplo expressa em função do número de movimentos de cada peça). O valor da Torre, por exemplo, pode ser expresso (com  $rm$  igual ao número de movimentos possível) por  $5 \times n \times bT + 3 \times rm$ . A função pode ainda ser mais elaborada se se acrescentar ao valor da peça um número que indique o «valor» do quadrado que ocupa (dando aos quatro quadrados centrais, por exemplo, o valor 8, aos que o rodeiam o valor 6,5, e aos seguintes 4). Pensar nos problemas inerentes à criação da função de avaliação para um jogo tão complexo como o xadrez mostra claramente que esta tarefa não é trivial.

Se está interessado em desenvolver funções de avaliação, talvez queira começar por uma para o programa aqui apresentado, usando-a para modificar o modo como os movimentos são escolhidos. Verificará que mesmo uma função grosseira – se conseguir levar o computador a aplicá-la na prática – melhorará de forma notória o jogo da máquina.

Deveria ser possível, dispondo de um tempo infinito e de um computador de potência igualmente infinita, investigar cada ramo da árvore até atingir o final do jogo. Isto significaria investigar um enorme número de possibilidades, como veremos. Um método mais aceitável consistirá em limitar a profundidade dessa investigação. Consideremos, por exemplo, que resolvemos deliberadamente seguir a árvore durante apenas dois passos, isto é, um movimento da máquina e as possíveis respostas do seu adversário.

O jogo aqui apresentado executa precisamente uma procura deste tipo (mas sem um mini-maxing total), procurando o movimento que lhe dá uma maior vantagem material, partindo do princípio de que o adversário joga o seu melhor lance possível em termos materiais (isto é, que o adversário captura uma peça sempre que possível). Considerando que a função de avaliação é realista, quanto mais profunda for a investigação, quanto maior for o número de passos desta, melhores serão os resultados obtidos pelo programa.

No entanto, obtemos novamente números astronómicos quando aumentamos a profundidade da procura. Se considerarmos, no jogo «Três em Linha», que existem três movimentos possíveis no início do jogo (isto é, um movimento num dos cantos é igual a um movimento em qualquer canto, dado que o primeiro tabuleiro pode ser transformado nos outros por rotação), existem doze posições ao nível de procura em dois passos, e um número próximo de  $12 \times 7$  (próximo, por que nem todos estes jogos seriam executados até ao fim, dado que o empate ou vitória seria óbvio antes de todas as posições serem preenchidas) no nível seguinte.

Noutros jogos, as possibilidades aumentam ainda mais. Uma média de procura a quatro passos em xadrez, por exemplo, deve avaliar cerca de um milhão de possibilidades.

## O Algoritmo Alfa-Beta

Como podemos ter em conta este enorme número de possibilidades, numa tentativa de escrever um programa que jogue razoavelmente bem, mas que não necessite de 10 levantado à potência 40 anos para fazer um movimento? É chegado o momento de introduzir o algoritmo alfa-beta, um auxiliar muito útil para o «corte» de ramos da nossa árvore.

A ideia básica da concepção «alfa-beta» é simples, mas muito potente. Segundo ela, se podermos escolher entre um conjunto de movimentos possíveis – e depois de encontrarmos um que se adapta às nossas necessidades, as quais se podem exprimir por exemplo em termos de melhorar a pontuação produzida pela função de avaliação – não necessitamos de procurar outro movimento nesse conjunto.

O algoritmo alfa-beta é assim designado porque actua simplesmente por referência a dois valores, designados por alfa e beta. O nosso programa procura através de uma árvore, tentando encontrar um bom lance. Alfa é o valor do melhor movimento até então descoberto. À medida que a procura continua, o programa descobre um movimento que produz um valor inferior a alfa. Imediatamente reconhece que não vale a pena seguir esse ramo, porque tal conduziria a um resultado inferior ao melhor obtido até então. Isto significa que o computador pode passar a um novo ramo.

Entretanto, o programa tenta igualmente descobrir as possíveis respostas aos seus próprios movimentos. Se encontra uma resposta que é má do ponto de vista do adversário – pelo que este provavelmente não o executará – não vê qualquer interesse em estudar as situações que podem decorrer dessa resposta. Beta é então o valor que o adversário consegue ao executar a sua melhor resposta ao lance do computador. A procura é interrompida se o ramo conduz a um movimento do opositor que tenda a diminuir o valor de beta, avaliado do ponto de vista do adversário.

O corte desta investigação provocado pela descoberta de um mau movimento para o computador será designado por «alpha-cutoff»; no outro caso, a designação será, como é natural, «beta-cutoff».

Podemos observar uma forma grosseira do funcionamento deste algoritmo, do ponto de vista «alfa», na sequência de acontecimentos que agora apresentamos:

- Medir o valor do tabuleiro actual.
- Descobrir o primeiro movimento.
- Medir o valor do tabuleiro depois desse movimento.
- Descobrir a melhor resposta do adversário, e determinar o valor que o tabuleiro terá depois dela.
- Registrar ambos os valores.
- Descobrir o movimento seguinte, e seguir o mesmo processo.
- Se o novo movimento produz um melhor valor «mini-maxing», eliminar o primeiro, e manter o segundo.
- Continuar a verificar os movimentos possíveis, mantendo sempre um registo daquele que dá a melhor razão mini-maxing.

Actuando deste modo acabamos por obter um movimento único que – para o reduzido estudo efectuado – será o «melhor» a executar.

Note-se que o algoritmo alfa-beta pode ser aplicado em muitas áreas da tomada de decisões, além dos jogos de tabuleiro. Muitos programas inteligentes, ao enfrentarem uma escolha entre um determinado número de opções, seguem um «raciocínio» alfa-beta na tentativa de determinarem a melhor escolha possível.

Voltemos agora ao programa apresentado nesta parte. Talvez o leitor recorde que observámos a posição inicial, e que o computador produziu uma lista dos movimentos que estava a considerar. Os primeiros movimentos que nos forneceu foram os seguintes:

71 para 62	75 para 66
73 para 64	75 para 64
73 para 62	77 para 68
	77 para 66

Todos estes movimentos são definidos pela máquina como «seguros, não-captura», e têm um valor equivalente. Portanto,

o computador escolhe aleatoriamente entre eles, e executa o movimento 71 para 62, como podemos ver:

```

COMPUTADOR @ HUMANO @
 12345678
-----
10000000
,000000
,000000
,.,.,.
,.,.,.
H H H H H
H H H H H
-----
 12345678

```

De facto, os movimentos podem não ter o mesmo valor, dado que interessa desenvolver o jogo no centro do tabuleiro sempre que possível; mas o programa não contém qualquer informação sobre este facto, e portanto pensa (e actua em conformidade) que os movimentos são de facto equivalentes.

A resposta humana é em seguida apresentada, usando o número indicado à esquerda (ou à direita) e o de cima (ou de baixo), por esta ordem; depois o utilizador indica do mesmo modo o quadrado para onde deseja mover a peça:

```

MOVER DE? 24
PARA? 00

```

O tabuleiro é novamente impresso, e o computador revela os movimentos que está a considerar:

```

COMPUTADOR @ HUMANO @
 12345678
-----
10000000
,100000
,000000
,.,.,.
,.,.,.
H H H H H
H H H H H
-----
 12345678

```

PROCESSO LANCE LEGAL ALEATORIO

Alguns movimentos mais adiante, o tabuleiro apresenta-se do seguinte modo (nem todos os movimentos possíveis do computador são indicados na listagem):

COMPUTADOR @ HUMANO @

```

12345678
-----
10 . . 0 0 0 0
7 . 0 0 0 0
4 0 0 0 0 0
1 H . H . .
0 H . H . .
4 H . H . .
1 . H H H H
-----
12345678

```

O computador não pode agora executar quaisquer capturas, e todos os movimentos possíveis podem ser classificados como «bons, seguros». O computador está ainda a jogar por escolha arbitrária entre movimentos igualmente bons (do seu ponto de vista):

COMPUTADOR @ HUMANO @

```

12345678
-----
10 . . 0 . 0
7 . 0 0 0 0
4 0 . 0 0 0
1 H . H . .
0 H . H . .
4 H . H . .
1 . H H H H
-----
12345678

```

MOVIM DE 37  
HUMANO @

COMPUTADOR O HUMANO O

```

12345678
-----
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
-----
12345678
    
```

A posição torna-se agora um pouco mais complexa. As peças do adversário estão a aproximar-se, pelo que surge a possibilidade de executar um movimento para uma posição perigosa. Por exemplo, se o computador fizesse um movimento de 66 para 57, o utilizador humano responderia provavelmente capturando a peça em 57, usando a peça que neste momento se encontra em 48. O mecanismo de procura do programa revela este facto. Não esqueçamos que o computador tenta sistematicamente garantir que a pontuação após cada movimento se mantenha tão boa quanto possível, e que o humano não tenha assim a possibilidade de aumentar a sua própria pontuação. Um movimento de 66 para 57 contrariaria ambos estes objectivos, pelo que só um programa muito mau o executaria - tendo em conta os movimentos alternativos de que dispõe.

Felizmente, este programa pode aperceber-se destes perigos elementares, e executa de facto um movimento de 73 para 62, considerado como «bom, seguro»:

COMPUTADOR O HUMANO O

```

12345678
-----
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
-----
12345678
    
```



## COMPUTADOR @ HUMANO @

```

12345678
-----
00000000
00000000
00000000
00000000
00000000
00000000
00000000
12345678

```

```

MOVIMENTO DE? 24
PRÓXIMO? 00

```

Existem agora poucos movimentos «bons, seguros» para executar, e nestas condições o programa executa um movimento que parece constituir o seu primeiro erro, movendo de 55 para uma situação perigosa, 44. O adversário humano responde com uma captura, e o programa cumprimenta-o:

## COMPUTADOR @ HUMANO @

```

12345678
-----
00000000
00000000
00000000
00000000
00000000
00000000
00000000
12345678

```

```

MOVIMENTO DE? 00
PRÓXIMO? 00
BUENO BARRA:

```

O adversário colocou agora uma peça sua em perigo, um facto que o programa rapidamente apreende. Indica que descobriu esse lance, e executa-o:



COMPUTADOR 1 HUMANO 1

```
      12345678
      - - - - -
      10 . . . . 10
      9 . . . . 9
      8 . . . . 8
      7 . . . . 7
      6 . . . . 6
      5 . . . . 5
      4 . . . . 4
      3 . . . . 3
      2 . . . . 2
      1 . . . . 1
      - - - - -
      12345678
```

MOVER DE? 37  
PARA? 48

**Alguns momentos depois, o programa enfrenta várias possibilidades:**

COMPUTADOR 1 HUMANO 1

```
      12345678
      - - - - -
      10 . . . . 10
      9 . . . . 9
      8 . . . . 8
      7 . . . . 7
      6 . . . . 6
      5 . . . . 5
      4 . . . . 4
      3 . . . . 3
      2 . . . . 2
      1 . . . . 1
      - - - - -
      12345678
```

44 PARA 26 CAPTURANDO EM 35

ESTOU A PENSAR 44 PARA 53

33 PARA 11 CAPTURANDO EM 22

>> CAPTURA DESCOBERTA

CAPTUREI E OCUPEI 11  
NA LINHA TRASEIRA

**O programa escolhe o melhor movimento possível, capturando uma peça inimiga e terminando na primeira linha adver-**

sária (11), ganhando assim dois pontos (e desaparecendo a peça do tabuleiro).

As escolhas enfrentadas pelo jogador humano não são muito agradáveis. A peça em 35 encontra-se sob ameaça de 44, e não parece haver maneira de evitar o perigo. 44 não pode ser capturada, e 35 não pode ser movida. O humano ignora a ameaça dado que nada pode fazer para contrariá-la, e tenta encaminhar uma peça para próximo da primeira linha inimiga (impondo além disso uma pequena ameaça – porque a peça humana não está protegida – sobre a peça do computador em 75):

COMPUTADOR 3 HUMANO 1

```

12345678
-----
1 2 3 4 5 6 7 8
2  . . . . . . . .
3  . . . . . . . .
4  . . . . . . . .
5  . . . . . . . .
6  . . . . . . . .
7  . . . . . . . .
8  . . . . . . . .
-----
12345678

```

MOVIDA DE? 67  
 PRÓXIMO DE? 66

O programa decide não capturar a nova peça em 66, obtendo em vez dela a que está em 35, saltando sobre ela para 26. O ser humano diz então ao computador que captura a peça em 75, pelo que este o cumprimenta:

COMPUTADOR 4 HUMANO 1

```

12345678
-----
1 2 3 4 5 6 7 8
2  . . . . . . . .
3  . . . . . . . .
4  . . . . . . . .
5  . . . . . . . .
6  . . . . . . . .
7  . . . . . . . .
8  . . . . . . . .
-----
12345678

```

```

MOVER DE 7 66
PAPAI? 04
SUJITO 004.

```

O ser humano, obviamente, ganhou dois pontos com este salto, terminando na primeira linha inimiga. O computador possui agora quatro pontos (um menos do que o necessário para a vitória), e o humano três:

```

COMPUTADOR 4 HUMANO 3

```

```

      12345678
      - - - - -
0000 . . . . 0000
0000 . . . . 0000
0000 . 0 0 . 0 000
0004 . . . H H 4000
000H . . . H H 000
0000 . . . 0 0000
0000 . . . . 0000
      12345678

```

Uma das prioridades no interior da hierarquia de movimentos aceite pelo programa consiste em mover uma peça para a primeira linha adversária sempre que possível. Pode fazê-lo agora, e não hesita:

```

MOVO PARA LINHA TRASEIRA DE 25

```

Isto dá obviamente a vitória ao computador:

```

COMPUTADOR 5 HUMANO 3

```

```

      12345678
      - - - - -
0000 . . . . 0000
0000 . . 0 . 0000
0004 . 0 . . 0000
000H . . . H H 0000
0000 . . . . 0000
0000 . . . . 0000
      12345678

```

O JOGO TERMINOU

GANHEI

## Como funciona o programa

Tal como outros programas publicados neste livro, este é construído em torno de um ciclo principal, que é percorrido ciclicamente até ser satisfeita uma condição particular. No interior desse ciclo são chamadas várias subrotinas:

```
10 REM MINI-DAMAS
10 POKER B 3000,2000: BRIGHT 0: F
LAST 0: LINK 7: PAPER 1: BORDER 1
20 GO SUB 2070: REM
inicializar
30 GO SUB 1760: REM imprimir
tabuleiro
40 REM *** ciclo principal ***
50 GO SUB 1000: REM 1000000000
computador
60 GO SUB 1760: REM imprimir
tabuleiro
70 IF c>4 THEN GO TO 120
80 GO SUB 1000: REM aceitar
lance humano
90 GO SUB 1760: REM imprimir
tabuleiro
100 IF b<6 THEN GO TO 50
110 REM *****
```

Como o leitor pode verificar ao observar este ciclo principal, torna-se muito fácil apreender através dele a construção geral do programa. Por outro lado, torna-se fácil descobrir quaisquer erros. Se, por exemplo, o programa não imprimisse correctamente o tabuleiro, seria lógico verificar primeiro a rotina que se inicia na linha 1760, identificada por «impressão do tabuleiro».

A execução passa em primeiro lugar à rotina de inicialização, iniciada na linha 2070:

```
2070 REM **** inicializar ****
2080 REM *****
2090 CLS
2100 RANDOMIZE
2110 DIM a(140): REM tabuleiro
e espaços brancos 14 vezes
```



- G – actua como memória dos movimentos «bons, com captura segura».
- S – como G, mas as capturas aqui guardadas são menos interessantes, sendo apenas definidas como «seguras».
- T – guardam-se aqui as capturas não classificadas em qualquer das categorias anteriores.

As declarações REM identificam as variáveis aqui atribuídas, representando E um quadrado branco vazio, B um quadrado negro vazio (indicado no tabuleiro sob a forma de um ponto), C a peça do computador e H a peça «humana». HS contém a pontuação humana, e CS a pontuação do computador.

As linhas 2210 a 2260 lêem a configuração inicial do tabuleiro, colocando-a no quadro A.

O nosso ciclo principal dá-nos uma indicação do modo como o computador actua a partir deste ponto. Não veremos o modo como o tabuleiro é impresso, nem a forma como os movimentos humanos são aceites, porque não constituem problemas especiais do ponto de vista de programação.

Quando o computador procura o movimento a fazer, segue – como já referimos anteriormente – uma hierarquia de passos bem definida. O programa define três variáveis, que são usadas de cada vez que o programa executa o ciclo, sendo colocadas a zero pelas linhas 220, 230 e 240. As declarações REM explicam-nas:

```

2200 DIM *****
2300 DIM *****
2400 DIM *****
2500 LET *****: REM contar
2600 LET *****: REM contar
2700 LET *****: REM contar
2800 LET *****: REM contar
2900 LET *****: REM contar
3000 LET *****: REM contar
3100 LET *****: REM contar
3200 LET *****: REM contar
3300 LET *****: REM contar
3400 LET *****: REM contar
3500 LET *****: REM contar
3600 LET *****: REM contar
3700 LET *****: REM contar
3800 LET *****: REM contar
3900 LET *****: REM contar
4000 LET *****: REM contar
4100 LET *****: REM contar
4200 LET *****: REM contar
4300 LET *****: REM contar
4400 LET *****: REM contar
4500 LET *****: REM contar
4600 LET *****: REM contar
4700 LET *****: REM contar
4800 LET *****: REM contar
4900 LET *****: REM contar
5000 LET *****: REM contar
5100 LET *****: REM contar
5200 LET *****: REM contar
5300 LET *****: REM contar
5400 LET *****: REM contar
5500 LET *****: REM contar
5600 LET *****: REM contar
5700 LET *****: REM contar
5800 LET *****: REM contar
5900 LET *****: REM contar
6000 LET *****: REM contar
6100 LET *****: REM contar
6200 LET *****: REM contar
6300 LET *****: REM contar
6400 LET *****: REM contar
6500 LET *****: REM contar
6600 LET *****: REM contar
6700 LET *****: REM contar
6800 LET *****: REM contar
6900 LET *****: REM contar
7000 LET *****: REM contar
7100 LET *****: REM contar
7200 LET *****: REM contar
7300 LET *****: REM contar
7400 LET *****: REM contar
7500 LET *****: REM contar
7600 LET *****: REM contar
7700 LET *****: REM contar
7800 LET *****: REM contar
7900 LET *****: REM contar
8000 LET *****: REM contar
8100 LET *****: REM contar
8200 LET *****: REM contar
8300 LET *****: REM contar
8400 LET *****: REM contar
8500 LET *****: REM contar
8600 LET *****: REM contar
8700 LET *****: REM contar
8800 LET *****: REM contar
8900 LET *****: REM contar
9000 LET *****: REM contar
9100 LET *****: REM contar
9200 LET *****: REM contar
9300 LET *****: REM contar
9400 LET *****: REM contar
9500 LET *****: REM contar
9600 LET *****: REM contar
9700 LET *****: REM contar
9800 LET *****: REM contar
9900 LET *****: REM contar

```

As «memórias» são esvaziadas:

```

250 FOR J=1 TO 3
260 LET g(J)=0: REM esvaziar

```

```

memoria de capturas boas e
seguras
0070 LET s(j)=0: REM esvaziar
memoria de capturas seguras
0080 LET t(j)=0: REM esvaziar
memoria de outras capturas
0090 NEXT j

```

Agora, o computador começa o seu primeiro varrimento do tabuleiro, saltando sobre o processo de avaliação (ver a linha 320) se o quadrado avaliado não contém uma das suas peças. Pode valer a pena seguir pormenorizadamente toda a sua sequência de captura. As declarações REM explicam uma vez mais os passos da codificação:

```

0060 FOR J=30 TO 30 STEP -10
0070 FOR K=1 TO 8
0080 IF a(j+k)<>c THEN GO TO 0090
0090 REM eliminar avaliacao se nao
0100 REM aqui peca do computador
0110 REM **capturar % direita**
0120 LET x=j+k-10: LET y=j+k-10:
LET z=j+k-20: LET m=-11
0130 IF a(x)=h AND a(y)=b THEN G
O SUB 700: REM descoberta
capture
0140 REM **capturar % direita**
0150 LET x=j+k-11: LET y=j+k-20:
LET z=j+k-30: LET m=-9
0160 IF a(x)=h AND a(y)=b THEN G
O SUB 700: REM descoberta
capture
0170 NEXT K
0180 NEXT J
0190 IF %safe+%safe+capture=0 T
HEN GO TO 030: REM nao
descoberta capture
0200 REM ** escolher captura a
favor **
0210 PRINT : FOR i=1 TO 10: GOSUB
.0: GOSUB .0: NEXT i: PRINT
TAB 5; ">> CAPTURA DESCOBERTA"
0220 GOSUB 000
0230 IF %safe<>0 THEN GO TO 000
0240 IF %safe<>0 THEN GO TO 070
0250 REM ** escolher em capturas
gerais **
0260 LET move=t(INT (RAND*c captur
e)+1)

```

```

6400 GO TO 640
6500 RM escolher em capturas
6600 RM escolher em lances em
6700 LET move=9*(INT (RAND*66666)+
6800 RM ** executor lance **
6900 LET start=INT (move/166)
7000 LET end=move-166*start
7100 LET i=start-66)
7200 LET i=start-66)
7300 RM ** verificar pontos **
7400 IF start-66>16 THEN EXIT
7500 LET i=start-66)
7600 LET i=start-66)
7700 PRINT "OBRIGADO EM JOGAR O JOGO"
7800 LET i=66
7900 LET i=66
8000 RM ** captura segura **
8100 LET move=9*(INT (RAND*66666)+
8200 GO TO 640
8300 RM ** verificar sequencia
8400 de captura **
8500 RM ** verificar quadrado
8600 de direção **
8700 PRINT "OBRIGADO EM JOGAR O JOGO"
8800 LET i=66
8900 LET i=66
9000 IF i=66 THEN GO TO 620: R
9100 RM guardar como captura seg
9200 segura
9300 RM ** verificar quadrado
9400 de direção **
9500 IF i=66 AND i=66 TH
9600 GO TO 620
9700 RM verificar se fica usa
9800 peça para não executor lance
9900 IF i=66 AND i=66
1000 THEN GO TO 620
1010 RM ** se atingido este ponto
1020 a captura é segura **

```



Se o programa encontrou um movimento «bom, seguro» (ou mais do que um), executa-o, permitindo depois ao adversário executar o seu. Se não o encontrou, mas dispõe de facto de um movimento apenas «seguro», executa-o. Se também não o encontra, executa um movimento de «captura».

Se nenhuma destas eventualidades for possível, o programa passa ao elemento seguinte da sua hierarquia, executando um movimento que proteja uma peça que está ameaçada pelo adversário.

```

10000 IFM *** lance para proteger
10005 IFM ***
10010 LET move=0
10015 LET J=0
10020 LET K=1
10025 IFM a(q)<>c THEN GO TO 1110:
IFM D=0 considerar este quadrado
do. e de peça do computador
10040 IF a(q+10)=b AND a(q-10)=b AN
D a(q+10)=c THEN LET move=100*(q
+100)+a+10
10050 IFM e eliminacao do lance de
o x p de outra peça
10060 IF move<>0 AND a(a-b)=b AND
a(a+b)=b AND AND).0 THEN GO TO
10110
10070 IF a(q+10)=b AND a(q-10)=b AN
D a(q+10)=c THEN LET move=100*(q
+100)+a+10 GO TO 10110
10080 IF a(q+11)=b AND a(q-11)=b
AND a(q+10)=c THEN LET move=100*
(a+100)+a+11
10090 IF move<>0 AND a(a+10)=b AND
a(a+10)=b AND AND).0 THEN GO TO
10110
1110 IF K<=0 THEN LET K=K+1: GO T
O 10020
1120 IF J>10 THEN LET J=J-10: GO
TO 1010

```

Se um tal movimento for descoberto pela linha 1040, a linha seguinte verificará se ele não expõe qualquer outra peça a um perigo. Se assim acontecer, o movimento proposto será rejeitado cerca de 50% das vezes. Isto não é obviamente um mecanismo sofisticado para a realização de uma escolha, mas garante pelo menos que o computador nem sempre execute

cegamente movimentos com o objectivo de proteger uma peça (o que poderia ser descoberto e explorado por um jogador humano), tendendo ainda a tornar cada novo jogo diferente dos anteriores.

A movimentação de uma peça para a primeira linha adversária produz a mesma recompensa que a captura de uma peça inimiga, pelo que o elemento seguinte da hierarquia consiste em executar esse movimento se possível. A rotina que se inicia na linha 1140 verifica o facto. Já expliquei anteriormente o modo como a sequência de avaliação prefere deslocar para a primeira linha inimiga as peças que se encontram mais perto do centro do tabuleiro, em vez das que estão nas extremidades:

```

1140 REM ** nao encontra rump, e
portanto procura lance para "de-
sapear" na linha traseira **
1150 LET move=0
1160 REM lance indesejavel veri-
ficado primeiro, para poder ser
coberto por melhores
1170 IF a(20)=c AND a(11)=b THEN
LET move=20
1180 IF a(20)=c AND a(17)=b THEN
LET move=200
1190 IF a(20)=c AND a(13)=b THEN
LET move=2000
1200 IF a(20)=c AND a(17)=b THEN
LET move=20000
1210 IF a(20)=c AND a(15)=b THEN
LET move=200000
1220 IF a(24)=c AND a(15)=b THEN
LET move=204
1230 IF a(24)=c AND a(13)=b THEN
LET move=204
1240 IF move=0 THEN GO TO 1310
1250 PRINT : PRINT INK 4: BRIGHT
1; "MOVO PARA LINHA TRASEIRA, DE
";move
1260 PULSE 500
1270 LET a(move)=b
1280 LET cs=cs+1
1290 RETURN

```

Se isto não for possível, o programa varre novamente o tabuleiro a fim de encontrar um movimento legal que não envolva qualquer risco. Os movimentos são contados pela variável CMOVE, sendo de facto o lance executado pela linha 1500:





```

0240 LET capture=0: REM contar
0250 OUTRA capture
0300 FOR J=1 TO 3
0310 LET a(J)=0: REM avaliar
0320 FOR K=1 TO 3
0330 LET b(K)=0: REM avaliar
0340 FOR L=1 TO 3
0350 LET c(L)=0: REM avaliar
0360 FOR M=1 TO 3
0370 LET d(M)=0: REM avaliar
0380 FOR N=1 TO 3
0390 LET e(N)=0: REM avaliar
0400 NEXT N
0410 FOR K=0 TO 99 STEP -10
0420 FOR J=1 TO 3
0430 IF a(J+K)>0 THEN GO TO 0500
: REM eliminator avaliar a b c d e f a g
0440 REM ** capturar e computador
0450 LET x=J+K-99: LET y=J+K-100:
LET z=J+K-101: LET m=-11
0460 IF e(x) AND e(y) AND e(z) THEN G
O SUB 700: REM descobrir e
capturar
0470 REM ** capturar e direita **
0480 LET x=J+K-111: LET y=J+K-102:
LET z=J+K-103: LET m=-9
0490 IF e(x) AND e(y) AND e(z) THEN G
O SUB 700: REM descobrir e
capturar
0500 NEXT K
0510 NEXT J
0520 PRINT capture+capture+capture=0 T
HEN GO TO 000: REM nao
descoberto captura
0530 REM ** escolher captura e
funcao **
0540 PRINT : FOR I=1 TO 10: REM
0550 PRINT "0,1,2,3,4,5,6,7,8,9,10: NEXT I: PRINT
TAB(0); ">>> CAPTURA DESCOBERTA"
0560 PRINT 000
0570 IF capture<>0 THEN GO TO 0500
0580 IF capture<>0 THEN GO TO 0530
0590 REM ** escolher as capturas
gerais **
0600 LET move=(INT (RAND*capture
e)+1)
0610 GO TO 0540
0620 REM escolher as capturas
boas
0630 REM escolher as lances em
memoria

```

```

0300 LET MOVE=0 (INT (RND*(99999)) +
1)
0400 REM *** computador lance ***
0500 LET INT=(MOVE/100)
0600 LET START=100
0700 LET CONT=INT
0800 LET CONT=INT-100
0900 LET CONT=INT+1
1000 REM *** verificacao pontuacao
1100 IF INT=100 THEN PRINT "PARABENS!"
1200 IF INT<10 THEN PRINT "MUITO RUIM!"
1300 REM
1400 LET a=(start-R*ed)+b
1500 LET b=(a+1)*c
1600 LET INT=INT+1
1700 LET CONT=CONT-1
1800 IF CONT=0 THEN PRINT "FIM DO JOGO!"
1900 LET CONT=INT
2000 REM *** captura segura ***
2100 LET MOVE=(INT (RND*(99999)) +
1)
2200 GO TO 540
2300 REM *** verificacao seguranca
2400 IF INT=100 THEN PRINT "PARABENS!"
2500 IF INT<10 THEN PRINT "MUITO RUIM!"
2600 LET a=(a+1)*c
2700 LET b=(a+1)*c
2800 IF INT=100 THEN PRINT "PARABENS!"
2900 IF INT<10 THEN PRINT "MUITO RUIM!"
3000 REM
3100 LET a=(a+1)*c
3200 LET b=(a+1)*c
3300 IF INT=100 THEN PRINT "PARABENS!"
3400 IF INT<10 THEN PRINT "MUITO RUIM!"
3500 REM
3600 LET a=(a+1)*c
3700 LET b=(a+1)*c
3800 IF INT=100 THEN PRINT "PARABENS!"
3900 IF INT<10 THEN PRINT "MUITO RUIM!"
4000 REM
4100 LET a=(a+1)*c
4200 LET b=(a+1)*c
4300 IF INT=100 THEN PRINT "PARABENS!"
4400 IF INT<10 THEN PRINT "MUITO RUIM!"
4500 REM
4600 LET a=(a+1)*c
4700 LET b=(a+1)*c
4800 IF INT=100 THEN PRINT "PARABENS!"
4900 IF INT<10 THEN PRINT "MUITO RUIM!"
5000 REM
5100 LET a=(a+1)*c
5200 LET b=(a+1)*c
5300 IF INT=100 THEN PRINT "PARABENS!"
5400 IF INT<10 THEN PRINT "MUITO RUIM!"
5500 REM
5600 LET a=(a+1)*c
5700 LET b=(a+1)*c
5800 IF INT=100 THEN PRINT "PARABENS!"
5900 IF INT<10 THEN PRINT "MUITO RUIM!"
6000 REM
6100 LET a=(a+1)*c
6200 LET b=(a+1)*c
6300 IF INT=100 THEN PRINT "PARABENS!"
6400 IF INT<10 THEN PRINT "MUITO RUIM!"
6500 REM
6600 LET a=(a+1)*c
6700 LET b=(a+1)*c
6800 IF INT=100 THEN PRINT "PARABENS!"
6900 IF INT<10 THEN PRINT "MUITO RUIM!"
7000 REM
7100 LET a=(a+1)*c
7200 LET b=(a+1)*c
7300 IF INT=100 THEN PRINT "PARABENS!"
7400 IF INT<10 THEN PRINT "MUITO RUIM!"
7500 REM
7600 LET a=(a+1)*c
7700 LET b=(a+1)*c
7800 IF INT=100 THEN PRINT "PARABENS!"
7900 IF INT<10 THEN PRINT "MUITO RUIM!"
8000 REM
8100 LET a=(a+1)*c
8200 LET b=(a+1)*c
8300 IF INT=100 THEN PRINT "PARABENS!"
8400 IF INT<10 THEN PRINT "MUITO RUIM!"
8500 REM
8600 LET a=(a+1)*c
8700 LET b=(a+1)*c
8800 IF INT=100 THEN PRINT "PARABENS!"
8900 IF INT<10 THEN PRINT "MUITO RUIM!"
9000 REM
9100 LET a=(a+1)*c
9200 LET b=(a+1)*c
9300 IF INT=100 THEN PRINT "PARABENS!"
9400 IF INT<10 THEN PRINT "MUITO RUIM!"
9500 REM
9600 LET a=(a+1)*c
9700 LET b=(a+1)*c
9800 IF INT=100 THEN PRINT "PARABENS!"
9900 IF INT<10 THEN PRINT "MUITO RUIM!"

```

```

0000 REM *** Verificador se pode
0001 ser chamado "lance" ***
0040 LET check=0: safe
0080 IF G+M#<1 THEN RETURN
0090 IF a(G+M)=b AND a(G-120+M)
<>b AND a(G+M#M)=b THEN LET safe=
safe+1
0070 IF check=safe THEN RETURN
: REM *** este lance nao e
considerado bom ***
0000 REM *** guardar lance
seguro em bom ***
0000 PRINT "MOTOU A PENHA " J+k
: PRINT "M+M0+J+k
0000 LET a(safe)=100*(J+k)+M0+M
0010 RETURN
0000 REM guardar captura nao
segura
0000 LET capture=capture+1
0040 PRINT "MOTOU A PENHA " J+k
: PRINT "M+M0+J+k
0000 LET t(capture)=100*(J+k)+
M0+M
0060 RETURN
0070 REM *****
0000 REM *** (este programa proteger
o programa de cada um) ***
10000 LET move=0
1010 LET k=1
1020 LET a=J+k
1030 IF a(Q)<>c THEN GO TO 1010:
REM nao considerar este quadra-
do, esse jogo do computador
1040 IF a(Q+10)=b AND a(Q-10)=b AN
D a(Q+10)=c THEN LET move=100*(a
+10)+a+10
1050 REM eliminacao do lance se
expor outro peao
1060 IF move<>0 AND a(Q-10)=b AND
a(Q+10)=b AND AND>.0 THEN GO TO
1010
1070 IF a(Q+10)=b AND a(Q-10)=b AN
D a(Q+20)=c THEN LET move=100*(a
+20)+a+10: GO TO 1010
1080 IF a(Q+11)=b AND a(Q-11)=b
AND a(Q+22)=c THEN LET move=100*
(a+22)+a+11
1090 IF move<>0 AND a(Q+2)=b AND
a(Q+22)=b AND AND>.0 THEN GO TO
1010

```

```

1110 IF K<8 THEN LET K=K+1: GO T
O 1020
1120 IF J>10 THEN LET J=J-10: GO
TO 1010
1130 REM *****
1140 REM ** nao encontra rumo, e
portanto procura lance para "des-
separar" na linha traseira **
1150 LET move=0
1160 REM lance indesejavel veri-
ficado primeiro, para poder ser
coberto por melhores
1170 IF a(20)=c AND a(11)=b THEN
LET move=20
1180 IF a(20)=c AND a(17)=b THEN
LET move=20
1190 IF a(20)=c AND a(13)=b THEN
LET move=20
1200 IF a(20)=c AND a(17)=b THEN
LET move=20
1210 IF a(20)=c AND a(15)=b THEN
LET move=20
1220 IF a(24)=c AND a(15)=b THEN
LET move=24
1230 IF a(24)=c AND a(13)=b THEN
LET move=24
1240 IF move=0 THEN GO TO 1010
1250 PRINT : PRINT INK 4; BRIGHT
1; "MOVO PARA LINHA TRASEIRA, DE
";move
1260 PAUSE 500
1270 LET a(move)=b
1280 LET cs=cs+1
1290 RETURN
1300 REM *****
1310 REM lances seguros sem
cobertura
1320 LET cmove=0: REM *** des-
coberte contagem de lances ***
1330 FOR J=00 TO 30 STEP -10
1340 FOR J=1 TO 0
1350 IF a(J+k)>c THEN GO TO 145
0
1360 LET X=J+k-9: LET y=J+k-18:
LET z=J+k-20
1370 LET q=J+k+2
1380 IF a(x)<b THEN GO TO 1460
1390 IF a(y)=b OR a(z)=b AND a(q
)=b THEN GO TO 1460
1400 GO SUB 1060

```

```

1410 LET x=j+k-11: LET y=j+k-22:
  LET z=j+k-20
1420 LET q=j+k-2
1430 IF a(x)<>b THEN GO TO 1460
1440 IF a(y)=h OR a(z)=h AND a(q)
  =b THEN GO TO 1460
1450 GO SUB 1000
1460 NEXT k
1470 NEXT j
1480 IF cmove=0 THEN GO TO 1500
1490 REM *** executor lance ***
1500 LET move=t(INT (RAND*cmove)+
  1)
1510 LET start=INT (move/100)
1520 LET ed=move-100*start
1530 LET a(start)=b
1540 LET a(ed)=c
1550 RETURN
1560 REM ** guardar lances **
1570 LET cmove=cmove+1
1580 PRINT "PENSO EM "(j+k)" PAR
  A ";x
1590 PAUSE 400
1600 LET t(cmove)=100*(j+k)+x
1610 RETURN
1620 REM *****
1630 REM lance aleatorio ***
capture
1640 INK 6: PRINT "PROCURO LANCE
  LEGAL ALABATORIO"
1650 LET l=0
1660 LET l=l+1
1670 LET j=10*INT (RAND*8+1)
1680 LET k=INT (RAND*8+1)
1690 IF a(j+k)=c THEN GO TO 1720
1700 IF l<200 THEN GO TO 1660
1710 PRINT : FOR i=1 TO 4: BEEP
  .5,i: BEEP .2,i+10: BEEP .3,i+5:
  NEXT i: PRINT "CONCEDO O JOGO"
  STOP
1720 IF a(j+k-9)=b THEN LET move
  =100*(j+k)+j+k-9: GO TO 1510
1730 IF a(j+k-11)=b THEN LET mov
  e=100*(j+k)+j+k-11: GO TO 1510
1740 GO TO 1700
1750 REM *****
1760 REM imprimir tabuleiro
1770 CLS
1780 PRINT
1790 PRINT INK 6: BRIGHT 1:"COMP

```

```

UTADOR "): FLASH 1);(c); FLASH 2)";
HUMANO "): FLASH 1);(s)
1000 PRINT
1010 PRINT "          103455678 "
1020 PRINT "          -----"
1030 FOR J=80 TO 10 STEP -10
1040 PRINT "          "J/10);
1050 FOR K=1 TO 8
1060 INK 4: PRINT PAPER 5; INK 8
;CHR$(a(J+K));
1070 NEXT K
1075 INK 7
1080 PRINT J/10
1090 NEXT J
1095 PRINT "          -----"
1010 PRINT "          103455678 "
1020 PRINT
1030 RETURN
1040 REM *****
1050 REM * acelerar tempo humano*
1060 INPUT "MOUSE DE start"
1070 IF a(start)<h THEN GO TO 10
00
1000 INPUT "VIRA "a
1010 PRINT "): DIGIT 1);(a "a
a); para "a: IF a(a)<b OR
a(b) (start+a)>11 AND a(start+
a)/2)>c THEN GO TO 1000
2000 LET a(start)=b
2010 LET a(a)=a
2020 IF a(b) (start-a)>11 THEN L
ET a(start+a)/2)=b: LET a(a)+
1: REM .67: REM .2: REM .1:
40 PRINT INK 7; GO RIGHT 1; REM
0: FLASH 1;" MUITO BOM. "
2000 IF a>99 THEN LET a(a)=
LET a(a)+1: PRINT "BOM UM BOM
01"
2040 PRUSE 100
2050 RETURN
2060 REM *****
2070 REM *****
2080 REM *****
2090 CLS
2100 RANDOMIZE
2110 DIM a(110): REM tabuleiro
e as peças brancas "a volta
2120 DIM a(3): REM capturas
brancas
2130 DIM s(3): REM capturas

```



## O VALOR DOS JOGOS

Afirmava-se nos primeiros tempos da investigação em inteligência artificial, que a programação de jogos não tinha interesse. Foi sugerido que o esforço aplicado nos algoritmos do jogo de xadrez, por exemplo, poderia ser melhor aplicado em esquemas que demonstrassem teoremas matemáticos ou em programas que seguissem o modo (tanto quanto este era conhecido na época) como o cérebro humano funciona.

Mas o modo como um cérebro atinge uma solução para um problema complexo – como o representado por uma situação de meio jogo em xadrez – manteve fascínio contínuo. Muito antes de os computadores existirem tal como hoje os conhecemos, já se pensava no modo de escrever um programa de xadrez.

Em 1949, Claude Shannon (cujo trabalho com relés e no campo da lógica já foi discutido anteriormente neste livro), ao trabalhar para a Bell Telephone Laboratories, apresentou um artigo importante numa Convenção em Nova Iorque. Era intitulado *Programming a Computer for Playing Chess*. O valor do artigo transcendia bastante a sua importância histórica como primeira obra publicada sobre o tema. Um número significativo dos conceitos discutidos por Shannon nesse texto são ainda usados em programas actuais de jogo de xadrez.

Mais ainda, Shannon compreendeu que no caso de poderem ser resolvidos os problemas da programação de um computador para jogar xadrez, a experiência acumulada teria bastante valor para o desenvolvimento de programas noutros campos de igual complexidade. Listou alguns destes: a concepção de circuitos

electrónicos, resolução de situações complexas de comutação telefónica, tradução de línguas e problemas de educação lógica.

Os que criticavam a atenção dada à programação de jogos não compreendiam a situação. Qualquer avanço do conhecimento da inteligência artificial constitui potencialmente uma fonte de informação muito útil noutras áreas de aplicação daquela. Na parte deste livro dedicada ao programa «Três em Linha» encontramos um exemplo, não muito significativo, de um programa que aprende por si mesmo a jogar melhor. Mas esta ideia de aprendizagem pela máquina é em si extremamente importante.

## **A complexidade do mundo real**

Existem muitas situações que podem ser consideradas como o produto de uma série extraordinária de factores. É excessivo o número de factores que conduziram à situação actual para permitir compreendê-la completamente. E se a situação se altera (como acontece no mundo real) a capacidade do homem para acompanhar essas alterações e tomar as decisões mais apropriadas vê-se bastante limitada.

É aqui que os computadores que executam jogos podem tornar-se úteis. A perícia ganha na escrita de uma função de avaliação para o jogo de xadrez (uma função de avaliação determina a força ou fraqueza de um dos lados num jogo, em função de um certo número de factores, incluindo o número de peças em jogo, a natureza destas e respectivas posições, das posições que podem atacar, etc.), pode obviamente ser aplicada na produção de uma função de avaliação que sugerisse os melhores passos a executar para resolver problemas como a eliminação de restos nucleares, etc.

Consideremos a situação quando o reactor nuclear de Three Mile Island teve uma falha de funcionamento. O número de variáveis a considerar encontrava-se para além da capacidade dos operadores humanos, como o Relatório da Comissão Malone sobre o acidente indicava:

«... o operador foi bombardeado por números, luzes de aviso, etc., ao ponto de a detecção de qualquer situação de erro e a definição das medidas correctas a tomar se terem tornado impossíveis...»

Um computador que pudesse isolar os aspectos importantes e sugerir um tipo de actuação apropriado seria evidentemente valioso numa situação deste tipo.

Parece então provável que os conhecimentos ganhos com o desenvolvimento de programas como o do jogo do xadrez possam dar bons resultados noutras áreas da inteligência artificial.

Os avanços verificados nem sempre são os previsíveis. Por exemplo, os programas de xadrez têm sido escritos de forma a (a) emular o modo como os seres humanos jogam xadrez; (b) tentar simplesmente jogar tão bem quanto possível. Verificou-se que os programas que tentam emular o comportamento humano não conseguem, no seu conjunto, dar tão bons resultados como os que permitem às máquinas funcionarem da maneira mais adequada a elas próprias.

Podem tirar-se daqui duas conclusões. Uma primeira é que a tentativa de copiar os esquemas de funcionamento humano numa máquina pode não ser o melhor caminho a seguir para atingir os níveis mais elevados de inteligência artificial. A segunda é que, a partir da tentativa de construir um programa que se comporte como um ser humano, podemos ganhar algumas concepções válidas sobre o modo como a mente humana de facto funciona.

## **Outros jogos, outras lições**

Como é óbvio, o xadrez não foi o único jogo investigado nos primeiros tempos da inteligência artificial. Por exemplo, as damas e o «Três em Linha» receberam também bastante atenção.

Já discutimos anteriormente o trabalho de Arthur Samuel no desenvolvimento de um jogo de damas capaz de aprender. Samuel não compreendia os problemas envolvidos na escrita

de um programa deste tipo quando começou. Disse a Pamela McCorduck (em *Machines Who Think*, San Francisco: W.H. Freeman and Co., 1979; pp. 148, 149) que o seu programa de damas começou em 1949 quando – depois de trabalhar para a Bell – foi ensinar para a Universidade do Illinois.

Decidiu que a Universidade necessitava de um computador, mas a comissão directiva da Universidade não conseguiu obter os fundos necessários para tal. Samuel concluiu que a única maneira de conseguir uma máquina consistiria em utilizar o dinheiro disponível para a fazerem eles próprios. Pensou que, se conseguisse fazer algo de espectacular com a primeira máquina que pensavam construir, a publicidade conseguida ajudaria a atrair fundos governamentais. Samuel afirma ter pensado que as damas era um jogo bastante trivial, que seria fácil de programar. Depois de o programa estar escrito, utilizá-lo-iam para derrotar o campeão mundial da época num campeonato realizado numa cidade vizinha, Kankakee, o que facilitaria em sua opinião a aquisição dos fundos.

A grandeza desta tarefa depressa se tornou evidente. No momento em que o campeonato se realizou, nem sequer a máquina estava pronta.

Samuel conta que pensou nas damas porque sabia que outros grupos trabalhavam em jogos de xadrez. Em comparação com o xadrez, considerava o jogo de damas bastante simples. Mas, como vimos na secção que refere este jogo, mesmo a programação de um computador para o jogo «Três em Linha» apresenta várias dificuldades.

Se «Três em Linha» não é simples, pensemos num jogo como o Go. Foi muito o esforço aplicado durante toda a história da inteligência artificial na concepção de jogos de xadrez, mas pouco o foi no Go.

Existem três razões para este facto. Uma é puramente cultural. A maior parte dos ocidentais não jogam este jogo, mas quase todos nós conhecemos o xadrez. A segunda razão é histórica. Aqueles que primeiro se dedicaram a este campo, como Turing e Shannon, mostraram que o xadrez era uma área que valia a pena explorar. E a terceira razão, apontada por J. A. Campbell (em «Go», contribuição sua para *Computer Game-Playing, Theory and Practice*, editado por M. A. Bramer,

Chichester, West Sussex, Ellis Horwood, Ltd., 1983; p. 136) é que se verificou ser extremamente difícil escrever um programa que consiga jogar mesmo ao nível de um muito mau jogador humano.

De facto, se o leitor quiser enfrentar um verdadeiro desafio, pode evidentemente tentar resolver pelo menos alguns aspectos do jogo. Em vez do tradicional tabuleiro de  $19 \times 19$  posições, pode por exemplo considerar um tabuleiro de  $7 \times 7$ , ou limitar os movimentos possíveis. Tal como Othello foi criado acrescentando uma limitação na abertura ao Reversi, pode criar-se uma espécie de «mini-Go» com um tabuleiro pequeno, e certas restrições ao jogo.

Enquanto os jogos como o Othello – onde os valores relativos dos diversos quadrados do tabuleiro podem ser facilmente tabelados – podem responder bem às técnicas de procura «por grosso» típicas de um computador, foi já dito que o mesmo não acontece no caso do «Go». De facto, este jogo pode vir a substituir o xadrez como teste final do valor da inteligência artificial (ver David Brown, *Seeing is Believing*, op. cit., p. 177).

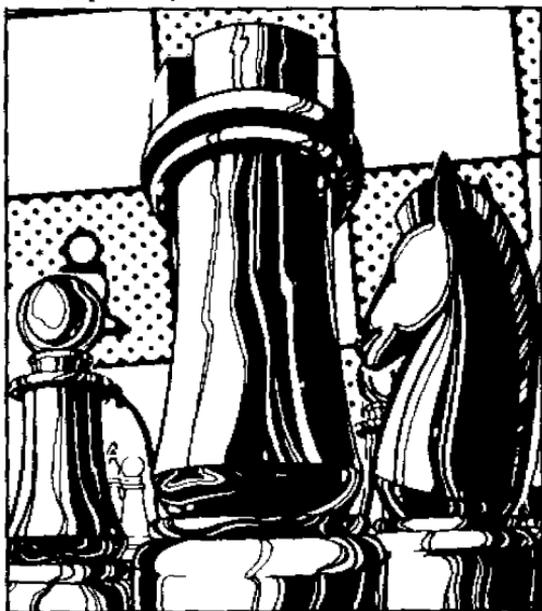


Fig. 12

### III PARTE FALANDO...

#### 6

### COMPREENDER A LINGUAGEM NATURAL

Existem poucas dúvidas quanto ao facto de a capacidade dos computadores para compreenderem a «linguagem natural» (isto é, a linguagem habitual que usamos para a comunicação humana) vir a ser o factor que permitirá em última análise avaliar a inteligência dos computadores ou falta dela.

A incapacidade dos computadores para conversarem na mesma língua que nós forma uma barreira entre estas máquinas e nós próprios. E uma tal barreira conduz-nos a não aceitar a existência no computador de qualquer medida de inteligência.

Neste campo foram produzidos dois ou três programas «clássicos», e nesta parte do livro examinaremos programas que vão permitir ao leitor ter pelo menos uma ideia do interesse que provocaram. Os programas em causa foram SHRDLU (de que apresentamos uma réplica adiante) e ELIZA (cujo equivalente neste livro, uma das mais completas já feitas em Basic, é chamada DOUTOR).

No SHRDLU original, um *robot* manipulava blocos coloridos e outras formas geométricas, em resposta a ordens dadas em linguagem natural. Conseguia manter uma boa conversação sobre aquilo que estava a fazer, o porquê de fazê-lo, e o que fizera no passado.

ELIZA, uma imitação de uma psiquiatra, foi tão eficaz que o seu criador conta ter recebido telefonemas angustiados de pessoas desesperadas para terem acesso ao programa para resolverem problemas da sua vida.

Além das nossas adaptações destes dois programas, observaremos ainda os problemas e as potencialidades da tradução por máquina. Incluímos um programa simples nesta parte, capaz de traduzir frases para «inglês», e que basta para ilustrar o

tipo de soluções que os computadores podem atingir quando tentam tratar não uma, mas duas linguagens naturais.

HANSHAN, o último programa desta parte sobre tratamento da linguagem, cria poemas aleatoriamente. Trata-se de um programa de nível bastante reduzido quando comparado com os outros aqui apresentados, e um que – poder-se-á dizer – em pouco evidencia a potência do computador onde é executado. No entanto, se o leitor tivesse lido as últimas linhas há cerca de 30 anos, com um autor afirmando que conseguia fazer poesia com uma máquina de baixo custo, afirmando ainda por cima que se tratava de um acontecimento insignificante, teria ficado espantado. Há trinta anos teria parecido um acontecimento de importância esmagadora.

No entanto, alguns dos resultados produzidos pelos programas contidos nesta parte do livro fornecem de facto resultados espantosos. Antes de entrarmos na discussão do funcionamento dos programas, convirá falar um pouco dos problemas que impedem uma comunicação perfeita entre o homem e a máquina na linguagem natural daquele.

### **«Parsing» da linguagem**

O termo «parsing» descreve a divisão das frases em elementos que um computador possa manipular. O campo da linguística em computador investiga tradicionalmente as formas de dividir as frases de tal modo que seja revelado o papel das várias partes destas relativamente à sua sintaxe. Isto é evidentemente feito na esperança de levar a máquina a ter uma compreensão mínima de cada frase.

No entanto, desenvolve-se cada vez mais o interesse pela procura do significado na frase, num âmbito muito mais vasto (o do conjunto de referências a que nós próprios recorreremos ao tentarmos compreender uma frase). Como é óbvio, à medida que a investigação baseada na estrutura sintática se desenvolve, a tendência para um método baseado num «conceito do mundo» aumenta.

É óbvia a razão de ser disto. Queremos ser capazes de falar com os computadores nos nossos próprios termos, em

vez de sermos limitados por linguagens «metálicas». Quando falamos sobre um tema que nos interessa, a amigos com interesses semelhantes, podemos partir de um conhecimento de fundo partilhado por todos. Do mesmo modo, gostaríamos de poder falar com os computadores a partir de uma mínima base de conhecimento comum.

Consideremos por exemplo que estamos a dirigir uma empresa mineira. Possuímos um programa de computador que nos ajuda a procurar minérios preciosos (e de facto existe pelo menos um programa nestas condições, chamado PROSPECTOR). Gostaríamos de poder falar com a máquina usando as palavras e frases que utilizamos geralmente quando falamos sobre este tema com os nossos colegas de trabalho.

A situação reduz-se a dar ao computador uma «concepção do mundo» que lhe permita interpretar as entradas em linguagem natural, usando os conhecimentos da máquina para levá-la a compreender o que dizemos por comparação.

O leitor descobrirá, nesta parte do livro, que as únicas demonstrações convincentes da comunicação em «linguagem natural» que lhe podemos apresentar envolvem «concepções do mundo extremamente restritas. No primeiro programa que se segue, por exemplo, o mundo consiste simplesmente num espaço bidimensional, no qual o nosso computador manipula quatro blocos coloridos. No entanto, os resultados obtidos pela máquina neste universo tão limitado são já muito interessantes, mesmo tendo em conta que não consegue atingir o brilhantismo de SHRDLU, o programa em que se inspira.

SHRDLU poderia por exemplo responder a frases do tipo **DESCOBRIR UM BLOCO MAIS ALTO DO QUE AQUELE QUE SEGURAS E PÔ-LO NA CAIXA**. No nosso programa equivalente, apesar de usar um universo mais limitado do que o de SHRDLU, já consegue no entanto funcionar de modo aceitável.

O programa, como já disse, deve manipular quatro blocos. Pode dizer-nos onde estão, indicando um bloco específico ou descrevendo toda a situação, e pode deslocá-los de um lado para o outro. No exemplo de execução que precede a listagem do programa, existe um bloco verde por cima de outro, amarelo. Pedi ao computador que colocasse o bloco vermelho sobre o



importante que funcione bem. Se, como verificaremos até certo ponto nesta parte sobre tratamento da linguagem, o computador pode usar a linguagem de forma eficaz, *como se de facto compreendesse o que ouve e diz*, isto bastará em muitas situações.

Os programas «dedicados» (que discutiremos em detalhe na parte correspondente do livro) podem trazer novas descobertas, e ajudar os humanos a resolverem problemas difíceis. O lado pragmático da inteligência artificial tende agora a assumir uma perspectiva do tipo «a inteligência é o que faz». Se existe um comportamento inteligente – mesmo que num domínio bastante restrito – partimos do princípio que o programa de facto compreende o que está a acontecer. Passemos portanto às questões importantes, como o aumento da inteligência aparente da máquina.

## **Problemas**

Os investigadores em inteligência artificial defrontam-se com um certo número de problemas importantes na tentativa de resolverem os mistérios do processamento da linguagem natural. O enorme número de palavras de qualquer língua falada, e a vastíssima quantidade de modos de combinação destas palavras, constitui obviamente o principal obstáculo. Muitas frases no interior de uma frase são ambíguas. A partir de conhecimentos anteriores, podemos em geral eliminar a ambiguidade de uma frase e compreender o seu significado. A ambiguidade é muitas vezes uma característica inerente à fala – talvez mais do que na comunicação escrita – e de facto a palavra dita é muitas vezes incompleta e quase completamente não estruturada.

Cada nova tarefa atribuída a um computador aumenta o tempo de processamento. Um sistema de linguagem natural não deve só por si exigir tanto tempo que o processo se torne inútil em termos de utilidade para o homem. Se o computador necessita de uma semana para «compreender» um parágrafo, não vamos certamente perder muito tempo a investigar a sua capacidade de comunicação conosco.

## Sintaxe e semântica

São estes os dois polos da investigação no campo do «parsing» da linguagem. Não se excluem necessariamente. São usados para enfrentar os problemas que existem mesmo no uso normal da linguagem. A simples descoberta de quem é o «ele» implícito na frase seguinte pode demorar um momento:

### O HOMEM QUE ESTAVA COM PEDRO DISSE QUE ESTAVA CANSADO

Se esta frase for lida no vazio, como o leitor acabou de fazer, não existe qualquer indicação de quem é a pessoa cansada, se bem que pessoalmente me pareça mais provável ser o homem que estava com Pedro.

Qualquer sistema de «parsing» da língua natural deve ser capaz de tratar problemas como este. Margaret Boden (em *Artificial Intelligence and Natural Man*, Hassocks, Sussex, Harvester Press, 1977, p. 112) dá o delicioso nome de «Problema do Arcebispo» à dificuldade de definir automaticamente o significado de tais palavras. A origem desse nome encontra-se em *Alice no País das Maravilhas*:

Mesmo Stigand, o patriótico arcebispo de Centerbury, achou conveniente -

"Achou o que?", disse o pato.

"Achou isso", respondeu o rato, aborrecido. "Debe certamente o que 'isso' significa?"

"Sei o que 'isso' significa", disse o pato, "quando acho alguma coisa. O problema é" - que achou o arcebispo?"

Vejamos um pouco melhor a frase, e como poderia ser dividida de forma a ser analisada (em seguida examinaremos o importante problema de saber como se define a «compreensão» da frase).

Eis a frase:

## O HOMEM VELHO E MAGRO ESTÁ POR BAIXO DO CARVALHO

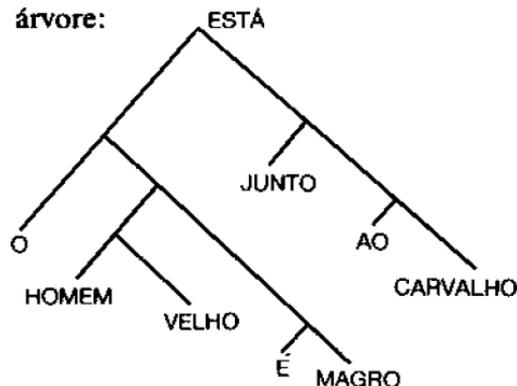
Podemos observar esta frase sintaticamente (colocando cada elemento sintático da estrutura entre parêntesis), do seguinte modo:

[[ [ O [ [ [ HOMEM ] [ VELHO ] ] ] [ E [ MAGRO ] ] ] ] ] ]  
 ESTÁ [ [ POR BAIXO ] [ DO [ CARVALHO ] ] ]

Observe cuidadosamente este exemplo, tente compreender a divisão feita, e talvez obtenha uma ideia razoável dos vários elementos que são ligados entre si. Por exemplo, as palavras **HOMEM** e **VELHO** estão ligadas separadamente, sob a forma [ **HOMEM** ] [ **VELHO** ] mas pertencem simultaneamente a um grupo maior: [ [ **HOMEM** ] [ **VELHO** ] ].

O adjectivo **MAGRO** modifica o substantivo, do mesmo modo que **VELHO**, estando portanto ligado de forma semelhante, sob a forma [ [ [ **HOMEM** ] [ **VELHO** ] ] [ **E** [ **MAGRO** ] ] ] mas esta forma apresenta uma ligação mais forte entre **VELHO** e **HOMEM** do que entre **MAGRO** e **HOMEM**. Existe ainda uma outra ligação que contém toda a frase. Além desta, existe uma outra ligação envolvendo a proposição; reconhecendo o facto de esta servir apenas para modificar a palavra que se segue, não estando de facto ligada às palavras que se encontram de ambos os lados, mas apenas a seguinte, só é construída esta última ligação.

Podemos exprimir a estrutura sintática da nossa frase sob a forma da seguinte árvore:



**Fig. 13**

Se conseguirmos levar um computador a dividir uma frase da forma indicada, tornando-se capaz de reconhecer as partes do discurso em cada ramo da árvore, ou os pares ligados na nossa frase-exemplo, estaremos obviamente a caminho de um certo grau de «compreensão».

Isto conduz-nos de novo à questão que já levantei um pouco atrás. Que significa, no contexto dos computadores, a palavra «compreender»? J. Klir e M. Valach (em *Cybernetic Modelling*, Londres; Iliffe Books, 1965) sugeriram que a compreensão de uma mensagem falada é geralmente vista com algo divisível em três partes:

1. Um modo de «ouvir» a mensagem
2. Um modo de responder a essa mensagem
3. Um método de determinação da possibilidade de interpretar a resposta 2) como mostrando ou não a ocorrência de uma compreensão.

Poderiam existir diversos métodos para determinar se existe ou não compreensão de termos escritos, segundo afirma Geoff Simons (em *Are Computers Alive?*, Brighton, Sussex; The Harvester Press, 1983, p. 129). Entre eles inclui-se supor que ocorreu tal compreensão quando o computador consegue responder correctamente às perguntas, ou notando o facto de a máquina poder realizar ligações inteligentes entre a sua própria base de conhecimento anterior e a informação que recebe ao «ler».

## BLOCOS DE PALAVRAS

Por vezes, um computador que «fala» em português natural (ou numa aproximação deste) pode produzir um efeito um tanto estranho. No programa que se segue, uma versão simplificada do famoso programa conhecido por SHRDLU (que discutiremos um pouco adiante), o computador manipula uma série de blocos coloridos, seguindo as nossas instruções, e diz-nos – de vez em quando – como estão dispostos os blocos relativamente uns aos outros.

Os blocos não existem de facto, como é óbvio, a não ser sob a forma de elementos electrónicos no cérebro do computador. No entanto, podemos ver uma representação desses blocos no visor, e esta representação altera-se à medida que a máquina desloca os blocos.

Como o leitor já terá certamente compreendido, é geralmente mais fácil obter uma demonstração convincente da inteligência da máquina quando o computador actua num domínio bastante restrito. O domínio em causa, dos blocos de brinquedo, é muito usado em experiências de inteligência artificial porque permite obter um certo grau de interacção e manipulação, como veremos.

No universo do computador existem apenas quatro blocos. As suas cores são vermelho, verde, amarelo e azul. Por vezes estas cores serão designadas por E (vermelho), V (verde), M (amarelo) e A (azul).

Quando o programa se inicia, vemos o seguinte no visor da televisão:

```

* * * * *
* * * * *
* * * * *
* * * * *
* E M B U *
```

Nesta situação o leitor está a observar os blocos «de frente». O universo referido é basicamente bidimensional. Se bem que seja possível deslocar os blocos, e colocá-los uns por cima dos outros, não se pode colocá-los «atrás» de outros, ou «à frente» deles. Os pontos são invisíveis para o computador. Existem apenas para benefício do utilizador, e mostram as posições que os blocos podem ocupar.

Se bem que dispunhamos de um número de frases muito limitado para comunicar com o programa, a conversação pode já ser relativamente agradável. Por outro lado, estas possibilidades conversacionais constituem já uma demonstração fácil de explicar (e suficientemente impressionante) de inteligência artificial.

Depois de os blocos terem surgido no visor, vemos um sinal de interrogação «?», indicando que o programa espera uma instrução nossa. Podemos pedir ao computador que descreva o seu mundo:

```

? DIGA-ME O QUE VE
COMECANDO PELA DIREITA
...UM ESPACO E DEPOIS
OS MEUS SENSORES REGISTAM O
BLOCO VERDE
E O BLOCO AZUL
OS MEUS SENSORES REGISTAM O
BLOCO ENCARNADO
FINALMENTE ...UM ESPACO
```

Depois de nos ter dado uma ideia geral do universo tal como o vê (indicando cada ponto como tratando-se de um «espaço»), podemos pedir à máquina que defina a localização de blocos específicos no interior desse universo:

```

* * * * *
* * * * *
* * * * *
* * * * *
* E M B U *
```

```
? ONDE ESTA' O BLOCO AMARELO
```

```
> DEIXE-ME VER <
```

```
ESTA' NA POSICAO 3 A PARTIR  
DA ESQUERDA  
O BLOCO ENCORADO ESTA'  
IMEDIATAMENTE A SUA ESQUERDA
```

```
VEJO O BLOCO AZUL A SUA  
DIREITA. TOCANDO NELE
```

```
NAO EXISTE NADA ACIMA
```

Como é óbvio, a simples indicação destas posições não é nada de especial, mesmo tendo em conta – como podemos verificar – que o programa está a responder-nos usando uma versão razoável de português (...). Vamos agora, no entanto, obrigar o computador a actuar sobre os elementos do seu universo restrito:

```
: : : : :  
: : : : :  
: : : : :  
: : : : :  
: E M A O :
```

```
? POR O BLOCO VERDE SOBRE O  
BLOCO AMARELO
```

```
> OK
```

```
ESTOU A MOVER O BLOCO VERDE PARA  
CIMA DO BLOCO AMARELO
```

```
: : : : :  
: E M A : :  
:
```

```
? ONDE ESTA' O BLOCO AMARELO
```

```
> DEIXE-ME VER... <
```

```
ESTA' NA POSICAO 3 A PARTIR DA  
ESQUERDA
```

O BLOCO VERMELHO ESTÁ  
IMEDIATAMENTE À SUA ESQUERDA

VEJO O BLOCO AZUL À SUA  
DIREITA, TOCANDO NELE

POR CIMA ESTÁ O BLOCO VERDE

NAO EXISTE NADA POR CIMA

É muito fácil colocar um bloco sobre outro quando este não tem nada por cima. Mas é muito mais complicado, e obriga já a uma codificação mais complexa, fazê-lo quando o segundo bloco já tem outro por cima. A situação é tornada ainda mais complicada quando existe um quarto bloco por cima do primeiro, que deve ser retirado antes de se poder «pegar» neste:

```
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .
```

? POR O BLOCO VERMELHO SOBRE O  
BLOCO AMARELO

COMPREENDO  
MOVO AGORA O BLOCO VERDE

MOVO-O AGORA PARA A LINHA 4  
MOVO AGORA O BLOCO VERMELHO  
PARA JUNTO DO BLOCO AMARELO

Um pouco mais tarde, os blocos apresentam-se assim:

```
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .  
. . . . .
```

? DIGA O QUE VE

COMEÇANDO PELA DIREITA  
...UM BLOCO VERDE EM CIMA  
...UM BLOCO VERDE EM CIMA  
...UM BLOCO VERDE EM CIMA

```
OS MEUS SENSORES REGISTRAM O
BLOCO AMARELO
```

```
M O BLOCO VERMELHO
..UM POR BAIXO..
UM O BLOCO AZUL
..UM POR BAIXO..
UM O BLOCO VERDE
FINALMENTE ...UM ESPAÇO
```

Fazemos agora um pedido ainda mais exigente:

```
. . . . .
. UM . . . .
. UM . . . .
. < M . . . .
```

```
O POR O BLOCO AZUL POR CIMA DO
BLOCO AMARELO
```

```
. OK
DEVO MOVER O BLOCO VERMELHO
NOVO O ADORA A LINHA 5
NOVO ADORA O BLOCO AZUL
PARA JUNTO DO BLOCO AMARELO
```

```
>> CARREGUE NUMA TECLA <<
```

O programa comporta-se de um modo exemplar. Parece compreender as nossas ordens, e actua de acordo com elas. Dir-se-ia que sabe o que faz, e que pode portanto assumir os resultados das suas acções.

## Como funciona?

Como se obtém este resultado? Qual é o segredo deste impressionante comportamento? Tudo se resume a executar uma divisão das frases que seja capaz de isolar as partes importantes das nossas ordens, a manipular os elementos num quadro bidimensional de acordo com essas indicações e, finalmente, a imprimir o quadro de tal forma que indique a situação de uma forma fácil de compreender.

«Explicar» o que se passa deste modo não diminui eviden-

temente a impressão causada pela aparente inteligência do programa, se bem que a versão aqui usada seja apenas uma «criatura» primitiva quando comparada ao seu progenitor, SHRDLU.

Este programa, que foi terminado em 1971 por Terry Winograd, não só manipulava blocos, como ainda os mostrava no visor, a cores, criando o mundo e interagutando com ele. Vejamos o modo como Terry «fala» com SHRDLU (as saídas do computador são apresentadas uma vez mais em maiúsculas, precedidas por um sinal >):

```
Pegar no bloco vermelho
> OK
Encontrar um bloco mais alto do
que o actual, e coloca-lo na
caixa.
> OK
Que contém a caixa?
> A PIRAMIDE AZUL E A CAIXA
AZUL
O que suporta a piramide?
> A CAIXA
Existe um bloco grande atrás da
piramide?
> SIM, TRES: UM BLOCO VERMELHO,
UM CUBO VERDE, E O CUBO AZUL
```

Como se pode verificar, trata-se de um esforço notável, que de resto levou os investigadores a considerarem a obra de Winograd como uma das mais importantes no desenvolvimento de programas de linguagem natural. A nossa versão é de facto muito mais pobre.

No entanto, o nosso programa é ainda capaz de executar uma manipulação relativamente complexa, apesar de não ir tão longe como SHRDLU. Demonstraremos o que acabamos de dizer voltando a ele e pedindo-lhe que realize alguns malabarismos mais difíceis.

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

O bloco amarelo encontra-se, como podemos ver, por baixo do azul. Pediremos agora ao programa que inverta a posição destes dois blocos.

```
? POR O BLOCO AMARELO SOBRE O
BLOCO AZUL
```

```
COMPREENDO
DEVO MOVER O BLOCO AZUL
MOVO-O PARA A LINHA 4
MOVO AGORA O BLOCO AMARELO
PARA JUNTO DO BLOCO AZUL
```

```

* * * * *
* * * * *
* * * * *
* * * * *
* O * A * *

```

Agora construiremos uma pequena torre com os blocos; vejamos como o programa resolve a situação:

```
? POR O BLOCO VERMELHO SOBRE O
BLOCO AMARELO
```

```
COMPREENDO
```

```
ESTOU A MOVER O BLOCO VERMELHO
PARA JUNTO DO BLOCO AMARELO
```

```

* * * * *
* * * * *
* * * * *
* * * * *
* O * A * *

```

Dizemos em seguida ao computador que queremos retirar o bloco inferior, e colocá-lo sobre aquele que neste momento se encontra isolado. Note que o programa foi escrito de modo a garantir que quando o «alvo» se encontra livre, não seja coberto enquanto o outro bloco é libertado:

```
? POR O BLOCO AZUL SOBRE O
BLOCO VERDE
```

```
> OK
```

```
DEVO MOVER O BLOCO VERMELHO
MOVO-O PARA A LINHA 5
```

```

DEVO MOVER O BLOCO AMARELO
MOVO-O PARA A LINHA 5
MOVO AGORA O BLOCO AZUL
PARA JUNTO DO BLOCO VERDE

```

```

. . . . .
. . . . .
. C . . . M .
. C . . . M .
. C . . . M .

```

Temos agora duas torres pequenas, uma com o bloco azul por cima do verde, e a outra com o amarelo por cima do vermelho. Pediremos em seguida ao computador que manipule ambos os blocos que se encontram neste momento por baixo de outros. Como se pode ver, estamos a submeter o programa a uma série de testes de complexidade crescente:

```

? POR O BLOCO VERMELHO SOBRE O
BLOCO VERDE

```

```

: OK

```

```

DEVO MOVER O BLOCO AMARELO
MOVO-O PARA A LINHA 5
MOVO AGORA O BLOCO AZUL

```

```

MOVO-O PARA A LINHA 5
MOVO AGORA O BLOCO VERMELHO
PARA JUNTO DO BLOCO VERDE

```

```

. . . . .
. . . . .
. C . . . M .
. C . . . M .
. C . . . M .

```

O computador realizou a tarefa sem dificuldade. Usemos agora o bloco verde (por enquanto coberto pelo vermelho), colocando-o sobre o azul, começando assim a criar o que virá a ser uma torre com quatro blocos:

```

? POR O BLOCO VERDE SOBRE O
BLOCO AZUL

```

```

COMPREENDO
DEVO MOVER O BLOCO VERMELHO

```

```
MOVO-O PARA A LINHA 3
MOVO AGORA O BLOCO VERDE
PARA JUNTO DO BLOCO AZUL
```

```

. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
```

```
? POR O BLOCO VERMELHO SOBRE O
BLOCO AZUL
```

```
COMPREENDO
DEVO MOVER O BLOCO VERDE
```

```
MOVO-O PARA A LINHA 2
MOVO AGORA O BLOCO VERMELHO
PARA JUNTO DO BLOCO AZUL
```

```

. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
```

```
? POR O BLOCO VERDE SOBRE O
BLOCO VERMELHO
```

```
COMPREENDO
```

```
MOVO AGORA O BLOCO VERDE
PARA JUNTO DO BLOCO VERMELHO
```

```

. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
```

Chegamos agora a um teste bastante difícil - e que de resto pensei originalmente que o programa não saberia resolver. É certo que eu incorporara uma rotina geral de manipulação dos blocos, mas não tinha a certeza de que esta rotina pudesse tratar situações como esta. Trata-se de «pegar» num bloco (o azul), que se encontra perto da base da torre, removendo



**Como o programa esteve a trabalhar bem, resolvi dar-lhe um pequeno descanso:**

```

. . . . .
. . . . .
. . . . .
. . . . .
. . . . .

```

? BARALHAR OS BLOCOS UM POUCO

É BOM TER UMA OPORTUNIDADE DE FAZER O QUE QUERO!!

```

. . . . .
. . . . .
. . . . .
. . . . .
. . . . .

```

Vejamos agora os elementos importantes do programa.

### Módulos do programa

Tal como acontece noutros programas deste livro; este inicia-se por uma chamada de uma subrotina no final do programa que inicializa as variáveis usadas:

```

0470 REM inicializar
0480 CLS
0490 RANDOMIZE
0500 DIM a(5,5)
0510 FOR x=1 TO 5
0520 FOR y=1 TO 5
0530 LET a(x,y)=45
0540 NEXT y
0550 NEXT x
0560 LET a(1,2)=CODE "E": REM
0570 LET a(1,3)=CODE "M": REM
0580 LET a(1,4)=CODE "A": REM
0590 LET a(1,5)=CODE "V": REM
0600 VERDE
0610 RETURN

```

Como se pode verificar (linha 2500), usa-se um quadro de cinco por seis para guardar o «universo». Encontra-se inicialmente preenchido (linhas 2510 a 2550) pelo valor 46, isto é, pelo código ASCII dos pontos que são usados para indicar um espaço vazio. A posição de partida dos blocos é definida pelas linhas 2560 a 2590. Podemos verificar aqui que o programa atribui uma letra da cor (A para azul, etc.) ao bloco correspondente. Não existe nada de complicado nesta primeira subrotina.

Se bem que a subrotina de inicialização seja invocada apenas uma vez em cada utilização, uma outra, a relativa ao nome da cor, é chamada sempre que o computador se pretende referir a um bloco:

```

2400 DEM ## nome da cor ##
2410 IF q=CODE "B" THEN PRINT "U
2420 ELSE
2430 IF q=CODE "M" THEN PRINT "A
2440 ELSE
2450 IF q=CODE "R" THEN PRINT "A
2460 ELSE
2470 IF q=CODE "V" THEN PRINT "U
2480
2490 RETURN

```

Esta subrotina substitui a letra inicial pelo nome completo da cor correspondente. Ambas estas subrotinas encontram-se no final do programa.

Voltando ao início deste, encontramos uma curta secção de código que imprime a imagem dos blocos. Poderia tratar-se de uma subrotina, mas como é necessária sempre que o programa executa o ciclo principal, pareceu lógico incorporá-la aqui.

```

30 DEM ## imprimir vista ##
40 INK 4: CLS : PRINT : PRINT
50 FOR X=5 TO 1 STEP -1
60 PRINT TAB 5;
70 FOR Y=1 TO 6
80 PRINT INK 0; PAPER 3; CHR$
a(x,y);
90 NEXT Y
100 PRINT
110 NEXT X
120 INK 7: PRINT : PRINT

```

A linha 50 mostra que a imagem é impressa em posição invertida, sendo a quinta linha impressa antes da quarta, etc.,

e a primeira no fundo do visor. A razão de ser disto é que facilita a manipulação dos blocos pelo programa. Este sabe que deve procurar números maiores a fim de verificar se existe um bloco por cima daquele que está a considerar.

Não teria sido difícil fazer exactamente o contrário (quanto menor o número, mais elevada a posição do bloco), mas pareceu-me ser uma complicação desnecessária.

A secção de código seguinte aceita as instruções do utilizador, e a partir delas determina qual a subrotina a chamar para executar a ordem:

```
130 INPUT a#: LET a#=a#+"  
135 BEEP .1,2: BEEP .2,10: BEEP  
.2,0  
140 PRINT a#: PRINT  
150 IF LEN a#=15 THEN STOP : RE  
M TERMINAR CARREGARDO ENTER  
160 IF a#( TO 10) = "ONDE ESTA"  
THEN GO SUB 240  
170 IF a#( TO 10) = "DIGA O QUE"  
THEN GO SUB 1000  
180 IF a#( TO 0) = "BARALHAR" THE  
N GO SUB 1200  
190 IF a#( TO 5) = "POR O" THEN G  
O SUB 1500  
200 PRINT : PRINT : PRINT ">> O  
arregue numa tecla <<": PAUSE 3e  
4  
210 GO TO 40
```

A construção de um programa de inteligência artificial conduz-nos a compreendermos rapidamente a complexidade do funcionamento inteligente. Este programa actua sobre um universo muito restrito, e reage apenas às situações que foram especificamente previstas (se bem que, como vimos no exemplo de execução, consiga reagir a uma situação para a qual não pensei tê-lo preparado). Apesar das limitações, o programa exige uma codificação extensa, com uma secção dedicada a cada tipo de investigação, e à aceitação de cada ordem.

Consideremos, por exemplo, a rotina que determina a localização de um dado bloco. Primeiramente o programa deve verificar se desejamos um bloco que se encontre no universo que conhece. Faz isto extraindo a letra — na linha 260 — que

identifica o bloco, e verifica – na linha 270 – se este bloco é um dos quatro que é capaz de reconhecer:

```

240 REM "Onde está o bloco?"
250 LIMT = 0
260 FOR I = 1 TO 4
270 IF B#(I TO 2) = "AM" THEN LET
B# = "M": GO TO 300
280 IF B#(I TO 4) = "VERM" OR B#(
TO 4) = "E" THEN LET B# = "E": GO TO
300
290 IF B#(I TO 2) = "AZ" THEN LET
B# = "R": GO TO 300
300 IF B#(I TO 4) = "VERD" THEN LE
T B# = "V": GO TO 300

```

Se tivermos pedido, por exemplo, um «bloco cor-de-rosa», o programa utiliza a rotina seguinte (280 a 310) para escolher aleatoriamente uma resposta, antes de voltar ao programa principal:

```

300 IF RAND > .7 THEN GO TO 300
300 PRINT "perdao, nao tenho in
formacoes sobre isso": GO TO 3
10
300 PRINT "Não possui dados par
a responder"
310 RETURN

```

Se passa este obstáculo, começa a procurar o bloco. Não usamos ciclos FOR/NEXT nesta secção, dado que queremos tornar o programa capaz de abandonar a procura em qualquer momento. Um programa que sai arbitrariamente de ciclos FOR/NEXT não pode ser considerado bem construído (e muitos computadores enchem completamente a memória com grande rapidez se se acumularem muitos endereços de retorno no seu «stack»). Esta parte do programa fornece-lhe as primeiras informações necessárias para a localização do bloco.

```

320 LET m = CODE B#
340 PRINT TAB (6) "Deixe-me ver
"
350 LET x = 5
360 LET y = 1
370 IF a(x,y) = m THEN GO TO 410
380 IF y < 6 THEN LET y = y + 1: GO T
O 370

```

```

390 IF X>1 THEN LET X=X-1: GO T
O 350
400 GO TO 280
410 IF X>1 THEN GO TO 910: REM
POR cima de outro
420 IF Y>1 THEN GO TO 630: REM
nao 'a esquerda

```

As declarações REM no resto desta secção explicam o que cada linha faz:

```

440 REM *** 'a esquerda ***
450 PRINT "ESTA 'A ESQUERDA"
460 IF a(1,2)=46 THEN PRINT "NA
O EXISTE NADA IMEDIATAMENTE 'ASU
A DIREITA": GO TO 790
470 LET a=a(1,2)
480 PRINT
490 PRINT "PERTO DELE VEJO O BL
COO "
500 GO SUB 2400
510 GO TO 790
520 IF X<6 THEN GO TO 650
530 REM *****
540 REM *** 'a direita ***
550 PRINT
570 PRINT "ESTA' DO LADO DIREIT
O "
580 IF a(1,5)=45 THEN PRINT "NA
O EXISTE NADA IMEDIATAMENTE 'AES
QUERDA": GO TO 790
590 PRINT "'A ESQUERDA VEJO O B
LOCO "
600 LET a=a(1,5)
610 GO SUB 2400
620 GO TO 790
630 REM *****
640 REM *** centro ***
650 PRINT
670 PRINT "ESTA' NA POSICAO "G
" 'A PARTIR", "OU ESQUERDA"
680 IF a(x,y-1)=46 THEN PRINT "
NDO EXISTE NADA IMEDIATAMENTE 'A
ESQUERDA": GO TO 730
690 LET a=a(x,y-1)
700 PRINT "O BLOCO "
710 GO SUB 2400
720 PRINT "ESTA": PRINT "IMEDI
ATAMENTE 'A SUA ESQUERDA"
730 IF a(x,y-1)=46 THEN PRINT "

```

```

NÃO EXISTE NADA IMEDIATAMENTE "A
RESOLVER": GO TO 700
740 LET a=a(x,y+1)
760 PRINT : PRINT "VEJO O BLOCO
"
780 GO SUB 2400
770 PRINT "A DIREITA, TOCANDO
NOME"
7900 REM *****
7900 REM *** Digite o cimo? ***
8000 PRINT
810 LET P=X
820 IF X=5 THEN GO TO 910
830 IF a(x+1,y)=48 THEN PRINT "
NÃO ESTA NADA ACIMA": GO TO 910
840 PRINT : PRINT "ACIMA ESTA
O BLOCO"
850 LET a=a(x+1,y)
860 GO SUB 2400
870 LET X=X+1
880 GO TO 820
8900 REM *****
9010 REM ** por cima de outro **
9020 IF a(x,y) <> 48 THEN LET X=P
9100 PRINT
9200 IF X=1 THEN GO TO 910
9300 PRINT "ESTA"
9400 PRINT "POR CIMA DO BLOCO"
950 LET a=a(x-1,y)
960 GO SUB 2400
1000 LET X=X-1
1010 IF X<1 THEN GO TO 910
1020 GO TO 900
1030 RETURN

```

A rotina que se segue é chamada quando se pretende que o programa diga o que vê. Trata-se de uma rotina muito mais simples do que aquela que localiza um bloco específico:

```

1040 REM *****
1050 REM "Diga o que"
1060 PRINT "COMEÇANDO PELA DIREI
TA"
1070 LET y=5
1080 LET X=5
1090 IF a(x,y) <> 48 THEN GO TO 11
1100 IF y=1 AND x=1 THEN PRINT "
FINALMENTE"
1110 IF X=1 AND a(x,y)=48 THEN P

```

```

PRINT "...UM ESPACO ": IF Y>1 TH
EN PRINT "E DEPOIS"
1120 IF X>1 THEN LET X=X-1: GO T
O 1000
1130 IF Y>1 THEN LET Y=Y-1: GO T
O 1000
1140 RETURN
1150 LET I=INT (RND*2)
1160 IF I=0 THEN PRINT "E' O BLO
CO ": GO TO 1100
1170 IF I=1 THEN PRINT "OS MEUS
SENSEORS REGISTRAM O "BLOCO ":
GO TO 1100
1180 PRINT "VEJO O BLOCO ":
1190 LET A=B(X,Y)
1200 GO SUB R400
1210 IF X=1 THEN GO TO 1100
1220 LET X=X-1
1230 PRINT "... E POR BAIXO..."
1240 GO TO 1100
1250 RETURN

```

A rotina que «baralha» os blocos é também bastante simples:

```

1270 REM *****
1280 REM Baralhar os blocos
1290 PRINT
1300 IF RND>.5 THEN INK 3: PRINT
TAB 7:"CA' E' TEMPO": GO TO 130
0
1310 INK 5: PRINT "E' BOM TER UM
B OPORTUNIDADE DE": PRINT TAB 4)
"7 ANOS O QUE QUERO!"
1320 INK 7: FOR X=1 TO 5
1330 FOR Y=1 TO 5
1340 LET B(X,Y)=40
1350 NEXT Y
1360 NEXT X
1370 LET C=INT (RND*5)+1
1380 LET D=INT (RND*5)+1
1390 IF C<D THEN GO TO 1300
1400 LET C=INT (RND*5+1)
1410 IF C<D OR C=D THEN GO T
O 1300
1420 LET C=INT (RND*5+1)
1430 LET D=INT (RND*5+1) OR D=C
THEN GO TO 1300
1440 LET B(1,C)=0
1450 LET B(1,D)=0
1460 LET B(2,C)=0
1470 LET B(1,D)=71
1480 RETURN

```

Finalmente, atingimos a rotina que produz os resultados mais impressionantes, a que coloca um bloco sobre outro. Tal como acontece na primeira rotina importante que examinámos, as declarações REM dão aqui as explicações necessárias:

```

1500 REM "Por o bloco sobre o...
"
1510 IF RAND>.5 THEN PRINT TAB 5;
"COMPREENDO": GO TO 1530
1520 PRINT TAB 8;"> OK"
1530 LET b#=#(0): REM bloco obj
ect
1540 IF b#="E" THEN LET l=#6
1550 IF b#="A" THEN LET l=#7
1560 IF b#="U" THEN LET l=#8
1570 IF b#="M" THEN LET l=#9
1580 LET c#=#(l)
1590 LET b=#(0) b#
1600 LET c=#(0) c#
1610 LET f(0)=c
1620 REM * descobrir bloco b# *
1630 LET x=5
1640 LET y=1
1650 IF a(x,y)=b THEN GO TO 1740
1660 IF y<8 THEN LET y=y+1: GO T
O 1650
1670 IF x>1 THEN LET x=x-1: GO T
O 1640
1680 PRINT "NAO ENCONTRO O BLOCO
"
1690 LET a=b
1700 GO SUB 2400
1720 FOR t=1 TO 200: NEXT t
1730 RETURN
1740 LET r=x: LET s=y
1750 REM bloco objecto esta' em
r,s
1760 REM bloco alvo isolado?
1770 IF a(r+1,s)=45 THEN GO TO 1
920: REM "sim"
1780 IF a(r+2,s)=45 THEN LET tas
k=1: GO TO 1890
1790 LET task=3: IF a(r+3,s)=45
THEN LET task=2
1800 FOR w=task TO 1 STEP -1
1810 PRINT "OBUO MOVER O BLOCO"
1820 LET a=a(r+w,s)
1830 GO SUB 2400
1840 LET d=#(INT (RAND*5))+1
1850 LET a=a OR a(1,d)=c OR a(2

```

```

,de)=c OR a(3,de)=c THEN GO TO 1
1800
1870 PRINT "MOVO-O PARA A LINHA
"/de
1880 LET l=1
1890 IF a(l,de)=46 THEN LET a(l,
de)=a(r+w,s): LET a(r+w,s)=46: G
O TO 1910
1900 LET l=l+1: GO TO 1890
1910 NEXT w
1920 REM bloco-alvo em r,s, agor
e isolado
1930 REM Bloco-objecto isolado?
1940 REM Encontrar bloco-objecto
1950 LET x=5
1960 LET y=1
1970 IF a(x,y)=c THEN GO TO 2070
1980 IF y<6 THEN LET y=y+1: GO T
O 1970
1990 IF x>1 THEN LET x=x-1: GO T
O 1970
2000 PRINT "NAO ENCONTRO O BLOCO
"
2010 LET q=c
2020 GO SUB 2400
2040 FOR j=1 TO 200: NEXT j
2050 RETURN
2060 REM ** C encontrado **
2070 LET t=x: LET u=y: REM
posicao de C
2080 IF a(t+1,u)=46 THEN GO TO 2
090
2090 IF a(t+2,u)=46 THEN LET tas
k=1: GO TO 2110
2100 IF a(t+3,u)=46 THEN LET tas
k=2
2110 LET de=INT (RAND*6)+1
2120 IF de=u OR de=s THEN GO TO
2110
2130 FOR w=task TO 1 STEP -1
2140 PRINT "AGORA MOVO O BLOCO "
?
2150 LET q=a(t+w,u)
2160 GO SUB 2400
2180 PRINT
2190 PRINT "ESTOU A MOVE-LO PARA
A LINHA "de
2200 LET l=1
2210 IF a(l,de)=46 THEN LET a(l,
de)=a(t+w,u): LET a(t+w,u)=46: G
O TO 2200

```

```

22220 LET I=I+1: GO TO 2210
22300 NEXT #
22400 PRINT Bloco-objecto isolado
22500 PRINT *** FIM DO MOVIMENTO ***
22600 PRINT "ESTOU A MOVER O BLOCO
O "
2270 LET a=a(r,s): LET z=b(r,s)
2280 GO SUB 2400
2290 PRINT " PARA JUNTO DO BLOCO
"
2310 IF a(t,u)=45 THEN LET a(t,u)
=710
2320 LET a=a(t,u)
2330 GO SUB 2400
2350 LET a(r,s)=45
2360 LET a(t+1,u)=z
2370 FOR J=1 TO 200: NEXT J
2380 RETURN

```

Como é natural, o programa deve saber resolver todas as situações que encontra. Depois da listagem completa do nosso programa, apresentamos novos exemplos da conversação de Winograd com SHRDLU, o que servirá para dar ao leitor algumas ideias sobre a forma de desenvolver o programa aqui proposto. Mantendo-o estruturado da mesma forma em que agora se encontra, o leitor poderia verificar que é possível aumentar a complexidade sem nos perdermos no meio da codificação.

A única informação adicional necessária é o formato de entradas requerido pelo programa. É possível fazer quatro perguntas, indicadas a seguir (e este programa espera-as em maiúsculas, se bem que o leitor possa facilmente alterar este facto):

ONDE ESTÁ O BLOCO cor?  
DIZ-ME O QUE VÊS.  
BARALHA OS BLOCOS.  
PÓE O BLOCO cor SOBRE O BLOCO cor.

Pode abandonar-se o programa em qualquer momento (como se verifica na linha 150) carregando muito simplesmente em ENTER, quando o programa nos faz uma pergunta.

Vejamos agora a listagem completa:

```

100 REM BLOCOS
110 TITLE: PAPER 1: BORDER 1:
INK 7: BRIGHT 1
20 GO SUB 2470: REM initialize
7
30 REM ** imprimir visto **
40 INK 4: CLR: PRINT: PRINT
50 FOR X=5 TO 1 STEP -1
60 PRINT TAB 0;
70 FOR Y=1 TO 5
80 PRINT INK 0; PAPER 0;CHR$(
  (X,Y))
90 NEXT Y
100 PRINT
110 NEXT X
120 INK 7: PRINT: PRINT
130 BEEP 1,10: FOR I=1 TO 5: BE
EP .1,1: NEXT I
130 INPUT a$: LET a#=#a#+
135 BEEP .1,2: BEEP .2,10: BEEP
.2,3
140 PRINT a$: PRINT
150 IF LEN a#=16 THEN STOP . BE
K terminate correçdo Enter
160 IF a#( TO 16)="ONDE ESTÁ?"
TIME 600 GUB 240
170 IF a#( TO 16)="DIGA O QUE"
TIME 600 GUB 180
180 IF a#( TO 3)="BARALHEIRO" THE
N 600 GUB 120
190 IF a#( TO 5)="POR O" THEN G
O 300
200 PRINT: PRINT: PRINT ">> C
orreçdo BUSE tecla <": PAUSE 30
4
210 GO TO 40
220 REM *****
230 REM "Onde está o bloco?"
240 LIMIT 0=0
250 IF a#( TO )
260 G#( TO 2)="DM" THEN LET
a#="M": GO TO 330
270 IF a#( TO 4)="URDM" OR a#(
TO 4)="E" THEN LET a#="E": GO TO
330
280 IF a#( TO 2)="RZ" THEN LET
a#="R": GO TO 330
290 IF a#( TO 4)="URRD" THEN LE
T a#="U": GO TO 330

```

```

2000 IF AND>.7 THEN GO TO 300
2000 PRINT "Parabens, nao tenho in
formacoes sobre isso": GO TO 0
10
300 PRINT "Nao possuo dados par
a responder"
310 RETURN
320 REM *****
330 LET #=00000000#
340 PRINT TAB 0;"> Deixe-me ver
...
350 LET X=0
360 LET Y=1
370 IF a(X,Y)=# THEN GO TO 410
380 IF X<6 THEN LET Y=Y+1: GO T
0 370
390 IF X>1 THEN LET X=X-1: GO T
0 380
400 GO TO 200
410 IF X>1 THEN GO TO 010: REM
Por cima de outro
420 IF Y>1 THEN GO TO 500: REM
Nao a esquerda
430 REM *****
440 REM *** a esquerda ***
450 PRINT "ESTA A ESQUERDA"
460 IF a(1,2)=46 THEN PRINT "NA
EXISTE NADA IMEDIATAMENTE "ASU
A DIREITA": GO TO 700
470 LET q=a(1,2)
480 PRINT
490 PRINT "PERTO DELE VEJO O BL
000 "
500 GO SUB 2400
510 GO TO 700
520 IF X<6 THEN GO TO 650
530 REM *****
540 REM *** a direita ***
550 PRINT
570 PRINT "ESTA DO LADO DIREIT
0"
580 IF a(1,5)=46 THEN PRINT "NA
EXISTE NADA IMEDIATAMENTE "AES
QUERDA": GO TO 700
590 PRINT "A ESQUERDA VEJO O S
LOCO "
600 LET a=a(1,5)
610 GO SUB 2400
620 GO TO 700
630 REM *****
640 REM *** centro ***

```

```

660 PRINT
670 PRINT "ESTA NA POSICAO "
700 PRINT "A PARTIR", "DA ESCURAO"
680 IF  $a(x,4)=48$  THEN PRINT "
NAO EXISTE NADA IMEDIAMENTE A
ESCURAO": GO TO 730
690 LET  $a = a(x,4-1)$ 
700 PRINT "O BLOCO "
710 GO SUB 2400
720 PRINT "ESTA": PRINT "IMEDI
AMENTE A SUB ESCURAO"
730 IF  $a(x,4-1)=48$  THEN PRINT "
NAO EXISTE NADA IMEDIAMENTE A
ESCURAO": GO TO 790
740 LET  $a = a(x,4+1)$ 
750 PRINT : PRINT "VELO O BLOCO
"
760 GO SUB 2400
770 PRINT "A DIREITA, TOCANDO
NELA"
780 GOTO *****
790 GOTO *****
800 PRINT *****
810 LET  $p = x$ 
820 IF  $x=0$  THEN GO TO 910
830 IF  $a(x+1,4)=48$  THEN PRINT "
NAO EXISTE NADA ACIMA": GO TO 810
840 PRINT : PRINT "ACIMA ESTA"
O BLOCO "
850 LET  $a = a(x+1,4)$ 
860 GO SUB 2400
870 LET  $x = x+1$ 
880 GO TO 820
890 GOTO *****
900 IF  $p < 0$  THEN LET  $x = p$ 
910 PRINT
920 IF  $x=1$  THEN GO TO 910
930 PRINT "ESTA":
940 PRINT "OP CIMA DO BLOCO "
950 LET  $a = a(x-1,4)$ 
960 GO SUB 2400
970 LET  $x = x-1$ 
980 IF  $x=0$  THEN GO TO 910
990 GO TO 820
1000 GOTO *****
1010 PRINT *****
1020 PRINT "COMANDO PELA DIREI
DA"

```

```

1070 LET Y=5
1080 LET X=5
1090 IF B(X,Y) <> 46 THEN GO TO 11
1100 IF Y=1 AND X=1 THEN PRINT "
FINAL PRINT"
1110 IF X=1 AND B(X,Y) = 46 THEN P
PRINT " "
1120 IF Y > 1 TH
EN PRINT " "
1130 IF X > 1 THEN LET X=X-1: GO T
O 1090
1140 IF Y > 1 THEN LET Y=Y-1: GO T
O 1090
1150 RETURN
1160 LET I=INT (RND*2)
1170 IF I=0 THEN PRINT "0" O BLO
OO "
GO TO 1190
1180 IF I=1 THEN PRINT "05 MEUS
06 BLOOO "
GO TO 1190
1190 PRINT "0600 O BLOOO "
1200 LET B(X,Y)
1210 IF X=1 THEN GO TO 1160
1220 IF Y=1 THEN GO TO 1160
1230 LET X=X-1
1240 LET Y=Y-1
1250 GO TO 1160
1260 RETURN
1270 *****
1280 PRINT "06 BLOOO"
1290 IF B(X,Y) THEN INK 0: PRINT
"06" TEMPO": GO TO 1300
1310 PRINT "06 BLOOO"
1320 PRINT "06 BLOOO"
1330 PRINT "06 BLOOO"
1340 LET X=1 TO 5
1350 LET Y=1 TO 5
1360 IF B(X,Y) = 46
1370 PRINT "X"
1380 IF B(X,Y) THEN (RND*5)+1
1390 IF B(X,Y) THEN (RND*5)+1
1400 IF B(X,Y) THEN GO TO 1380
1410 IF B(X,Y) THEN (RND*5)+1
1420 IF B(X,Y) THEN GO T
O 1380
1430 IF B(X,Y) THEN (RND*5)+1
1440 IF B(X,Y) THEN (RND*5)+1
1450 IF B(X,Y) THEN (RND*5)+1
1460 IF B(X,Y) THEN (RND*5)+1
1470 IF B(X,Y) THEN (RND*5)+1
1480 IF B(X,Y) THEN (RND*5)+1
1490 IF B(X,Y) THEN (RND*5)+1
1500 IF B(X,Y) THEN (RND*5)+1
1510 IF B(X,Y) THEN (RND*5)+1
1520 IF B(X,Y) THEN (RND*5)+1
1530 IF B(X,Y) THEN (RND*5)+1
1540 IF B(X,Y) THEN (RND*5)+1
1550 IF B(X,Y) THEN (RND*5)+1
1560 IF B(X,Y) THEN (RND*5)+1
1570 IF B(X,Y) THEN (RND*5)+1
1580 IF B(X,Y) THEN (RND*5)+1
1590 IF B(X,Y) THEN (RND*5)+1
1600 IF B(X,Y) THEN (RND*5)+1
1610 IF B(X,Y) THEN (RND*5)+1
1620 IF B(X,Y) THEN (RND*5)+1
1630 IF B(X,Y) THEN (RND*5)+1
1640 IF B(X,Y) THEN (RND*5)+1
1650 IF B(X,Y) THEN (RND*5)+1
1660 IF B(X,Y) THEN (RND*5)+1
1670 IF B(X,Y) THEN (RND*5)+1
1680 IF B(X,Y) THEN (RND*5)+1
1690 IF B(X,Y) THEN (RND*5)+1
1700 IF B(X,Y) THEN (RND*5)+1
1710 IF B(X,Y) THEN (RND*5)+1
1720 IF B(X,Y) THEN (RND*5)+1
1730 IF B(X,Y) THEN (RND*5)+1
1740 IF B(X,Y) THEN (RND*5)+1
1750 IF B(X,Y) THEN (RND*5)+1
1760 IF B(X,Y) THEN (RND*5)+1
1770 IF B(X,Y) THEN (RND*5)+1
1780 IF B(X,Y) THEN (RND*5)+1
1790 IF B(X,Y) THEN (RND*5)+1
1800 IF B(X,Y) THEN (RND*5)+1
1810 IF B(X,Y) THEN (RND*5)+1
1820 IF B(X,Y) THEN (RND*5)+1
1830 IF B(X,Y) THEN (RND*5)+1
1840 IF B(X,Y) THEN (RND*5)+1
1850 IF B(X,Y) THEN (RND*5)+1
1860 IF B(X,Y) THEN (RND*5)+1
1870 IF B(X,Y) THEN (RND*5)+1
1880 IF B(X,Y) THEN (RND*5)+1
1890 IF B(X,Y) THEN (RND*5)+1
1900 IF B(X,Y) THEN (RND*5)+1
1910 IF B(X,Y) THEN (RND*5)+1
1920 IF B(X,Y) THEN (RND*5)+1
1930 IF B(X,Y) THEN (RND*5)+1
1940 IF B(X,Y) THEN (RND*5)+1
1950 IF B(X,Y) THEN (RND*5)+1
1960 IF B(X,Y) THEN (RND*5)+1
1970 IF B(X,Y) THEN (RND*5)+1
1980 IF B(X,Y) THEN (RND*5)+1
1990 IF B(X,Y) THEN (RND*5)+1
2000 IF B(X,Y) THEN (RND*5)+1

```



```

1850 IF de=s OR a(1,de)=c OR a(2
,de)=c OR a(3,de)=c THEN GO TO 1
8950
1870 PRINT "MOVO-O PARA A LINHA
...de
1880 LET l=1
1890 IF a(l,de)=46 THEN LET a(l,
de)=a(t+w,s): LET a(t+w,s)=46: G
O TO 1910
1900 LET l=l+1: GO TO 1890
1910 NEXT #
1920 REM bloco-alvo em r,s, egor
e isolado
1930 REM bloco-objecto isolado?
1940 REM encontrar bloco-objecto
1950 LET x=s
1960 LET y=1
1970 IF a(x,y)=c THEN GO TO 2070
1980 IF y<6 THEN LET y=y+1: GO T
O 1970
1990 IF x>1 THEN LET x=x-1: GO T
O 1960
2000 PRINT "NAO ENCONTRO O BLOCO
...
2010 LET q=c
2020 GO SUB 2400
2040 FOR j=1 TO 200: NEXT j
2050 RETURN
2060 REM ** c encontrado **
2070 LET t=x: LET u=y: REM
posicao de c
2080 IF a(t+1,u)=46 THEN GO TO 2
0800
2090 IF a(t+2,u)=46 THEN LET tas
k=1: GO TO 2110
2100 IF a(t+3,u)=46 THEN LET tas
k=2
2110 LET de=INT (RND*6)+1
2120 IF de=u OR de=s THEN GO TO
2110
2130 FOR w=task TO 1 STEP -1
2140 PRINT "AGORA MOVO O BLOCO "
...
2150 LET a=b(t+w,u)
2160 GO SUB 2400
2180 PRINT
2190 PRINT "ESTOU A MOVE-LO PARA
A LINHA "de
2200 LET l=1
2210 IF a(l,de)=46 THEN LET a(l,
de)=a(t+w,u): LET a(t+w,u)=46: G

```

```

0 TO 2200
2220 LET l=l+1: GO TO 2210
2230 NEXT #
2240 REM Bloco-objeto isolado
2250 REM *** Fazer movimento ***
2260 PRINT "ESTOU A HOVER O BLOC
0 "
2270 LET a=a(r,s): LET z=a(r,s)
2280 GO SUB 2400
2290 PRINT " PARA JUNTO DO BLOCO
...
2310 IF a(t,u)=46 THEN LET a(t,u)
)=f10g
2320 LET a=a(t,u)
2330 GO SUB 2400
2340 LET a(r,s)=46
2350 LET a(t+1,u)=z
2370 FOR J=1 TO 200: NEXT J
2380 RETURN
2390 REM *****
2400 REM ** nome de cor **
2410 IF a=CODE "E" THEN PRINT "V
MARRELO
2420 IF a=CODE "M" THEN PRINT "A
MARRELO "
2430 IF a=CODE "A" THEN PRINT "A
ZUL "
2440 IF a=CODE "U" THEN PRINT "U
VERDE "
2450 RETURN
2460 REM *****
2470 REM inicializar
2480 CLR
2490 RANDOMIZE
2500 DIM a(5,5)
2510 FOR x=1 TO 5
2520 FOR y=1 TO 5
2530 LET a(x,y)=46
2540 NEXT y
2550 NEXT x
2560 LET a(1,2)=CODE "E": REM
bloco encarnado
2570 LET a(1,3)=CODE "M": REM
bloco amarelo
2580 LET a(1,4)=CODE "A": REM
bloco azul
2590 LET a(1,5)=CODE "U": REM
bloco verde
2600 RETURN

```

## Melhoramento do programa

Se bem que o nosso programa dê já resultados interessantes na sua forma actual, é possível desenvolvê-lo de muitas maneiras. Como já vimos o programa tem uma forma modular, sendo dedicada uma única subrotina a cada tipo de comando dado. Nestas condições, para aumentar as capacidades do programa, pode aumentar-se a secção de reconhecimento do vocabulário incluída no início, e usá-la para enviar a execução para novas subrotinas.

Podemos começar-se, por exemplo, por dar ao programa a capacidade de deslocar um bloco para a direita, ou para a esquerda, em vez de o colocar simplesmente «por cima» de outro bloco. Depois de se ter conseguido fazê-lo (verificando se o computador é ainda capaz de rejeitar pedidos «impossíveis», como tentar colocar um bloco à direita de um outro que já se encontra na periferia do «universo» aceite), poderá tentar acrescentar um quinto bloco, ou talvez uma outra forma. Finalmente, pode criar o seu próprio programa completamente diferente, que seja capaz de deslocar blocos e outros elementos em três dimensões, dando sentido a expressões como «à frente de» ou «atrás de».

Para obter algumas ideias do que poderá fazer, observamos agora alguns exemplos de conversação com SHRDLU, o programa escrito por Terry Winograd. O curto trecho de conversação já apresentado anteriormente foi extraído de uma execução mais longa, que apresentaremos a seguir. Antes do mais, porém, vamos apresentar um esboço da situação discutida:

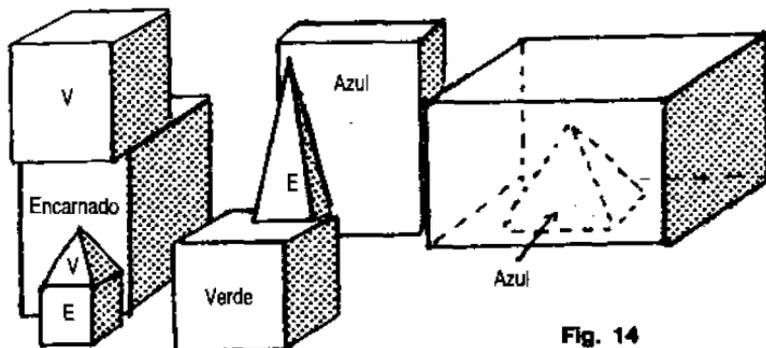


Fig. 14

## Continuemos então a conversação:

Pegue no bloco vermelho grande.  
> OK  
Agora na pirâmide.  
NÃO É A QUE PIRÂMIDE SE REFERE  
Descubra o bloco mais alto do  
que aquele que segura e ponha-o  
na caixa.  
> NÃO ENTENDO O BLOCO QUE É  
MAIS ALTO DO QUE AQUELE QUE  
SEGURA  
> OK  
Que contém a caixa?  
> A PIRÂMIDE AZUL E O BLOCO AZUL  
Onde está apoiada a pirâmide?  
> NA CAIXA  
Quantos blocos estão na caixa?  
> QUATRO  
É pelo menos um deles menos  
largo do que aquele que lhe  
deba ser seguro?  
> SIM, O CUBO VERMELHO  
Pode-se em alguma coisa?  
> SIM, NA CAIXA  
Pode-se pegá-lo em blocos?  
> NÃO  
Pode uma pirâmide apoiar-se num  
bloco?  
> SIM  
Pode uma pirâmide apoiar-se numa  
pirâmide?  
> NÃO  
Pode uma pirâmide sobre outra.  
> NÃO  
A pirâmide azul é minha.  
> COMO NÃO

Por muito impressionante que seja, este diálogo torna-se quase insignificante quando comparado com o que se segue, que implica uma manipulação lógica que mesmo os seres humanos considerariam difícil seguir. O segmento que se segue ocorreu imediatamente depois do anterior:

Possuo todos os blocos que não  
são vermelhos, mas não possuo  
nada que suporte uma pirâmide

>COMPREENDO  
 Possui a caixa?  
 >NAO  
 Possui alguma coisa dentro da caixa?  
 >SIM, DUAS COISAS: O BLOCO AZUL E A PIRAMIDE AZUL  
 Pode juntar ambos os blocos vermelhos e um cubo verde ou uma piramide?  
 >OK  
 Que cubo se encontra sobre a mesa?  
 >O CUBO VERDE QUE SUPORTA A PIRAMIDE VERMELHO  
 Existe um bloco grande por tras da piramide?  
 >SIM, TRES BLOCOS: UM BLOCO VERMELHO GRANDE, UM CUBO VERDE GRANDE E O BLOCO AZUL  
 Ponha o bloco pequeno junto do cubo verde que suporta uma piramide  
 >OK  
 Ponha a piramide mais pequena sobre ele  
 >OK  
 Quantas coisas estao sobre cubos verdes?  
 >NAO SEI BEM O QUE RESPONDER POR "SOBRE" NA FRASE "SOBRE CUBOS VERMELHOS"  
 >SIGNIFICADO:  
 1 - DIRECTAMENTE NA SUPERFICIE  
 2 - EM QUALQUER SITIO SOBRE?  
 2  
 Tocou em alguma piramide antes de por o bloco verde sobre o cubo pequeno?  
 >SIM, A PIRAMIDE VERDE  
 Quando pegou nela?  
 >QUANDO TENTAVA A ACUMULAR O CUBO VERMELHO, UM BLOCO VERMELHO GRANDE E UM GRANDE CUBO VERDE  
 Porque?  
 >PARA ME LIBERTAR DELA  
 Porque fez isso?  
 >PARA LIBERTAR O CUBO VERMELHO  
 Porque libertou esse cubo?  
 >PARA O POR SOBRE UM CUBO VERDE

GRANDE  
PORQUE FEZ ISSO?  
> PARA BOM CUMULADO O COBO VERMELHO,  
UM BLOCO VERMELHO GRANDE E UM  
COBO VERDE GRANDE  
PORQUE FEZ ISSO?  
> PORQUE ME PEDIU.

## O DOUTOR ESTÁ

Nenhum estudo do «processamento em linguagem natural» ficaria completo sem uma discussão de ELIZA, o mais famoso programa deste tipo (e, em certa época, o mais famoso programa de computador).

Joseph Weizenbaum, professor de Ciência Informática no Massachusetts Institute of Technology, criou ELIZA em 1964-6. O programa foi concebido para representar o papel de uma psicoterapeuta da escola Carl Rogers (Weizenbaum insiste em que a ideia era parodiar estes praticantes). Os psicoterapeutas desta escola recusam-se a conduzir a conversação com o doente, preferindo em vez disso reflectir as suas observações sobre eles próprios, de tal modo que em qualquer momento a conversação depende apenas do próprio doente.

A reacção ao programa – depois de ter sido tornado público – começou por impressionar, e acabou por preocupar Weizenbaum. Como resultado da corrente de pensamento produzida pelo choque a que deu origem (o que discutiremos dentro em pouco), Weizenbaum abandonou o Instituto durante dois anos a fim de pensar nas mais latas implicações da experiência ELIZA, e durante esse tempo escreveu a importante obra *Computer Power and Human Reason* (São Francisco, W. H. Freeman and Co., 1976). Este livro contém bastante material com o qual não concordo, mas esta observação de modo algum diminui a sua importância. Independentemente das minhas opiniões, trata-se de uma obra que aconselho o leitor a estudar se estiver seriamente interessado na inteligência artificial, e nas relações e dependências que vemos desenvolverem-se entre o homem e a máquina.

Usa-se o programa de Weizenbaum escrevendo a nossa parte do «diálogo», à qual o programa responde usando as observações que contém em memória. Vejamos agora uma parte de uma conversação típica com ELIZA:

>> TENHO MEDO DE FALAR EM  
PUBLICO

PENSA QUE E' NORMAL TER MEDO DE  
FALAR EM PUBLICO

-----

>> ACHO QUE MUITAS PESSOAS TEM  
ACHA QUE SIM?

-----

>> ACHO

PARECE TER A CERTEZA DISSO

-----

>> FALAR COM UM COMPUTADOR E'  
MUITO ESTRANHO

ESTA A PENSAR EM MIM

Weizenbaum escolheu o nome ELIZA para o seu programa porque, segundo disse, tal como a Eliza de Pigmaleão, podia ser ensinado a falar cada vez melhor. O programa ELIZA original foi escrito em duas partes. A primeira analisava a entrada do utilizador, e a segunda era um «argumento». Eram concebidos diferentes «argumentos» para diferentes tópicos, e DOUTOR (o nome da nossa versão deste programa) era um deles (outros suportavam discussões sobre navios antigos, propriedade de terras, cotações de moedas, etc.).

O argumento DOUTOR foi o primeiro experimentado por Weizenbaum. O programa tornou-se bem conhecido no Instituto de Tecnologia de Massachusetts porque constituía uma forma bastante eficaz de demonstrar a potência de um computador (não esqueçamos que tudo isto se passou há bastante tempo, quando as pessoas ainda não compravam computadores em tabacarias).

Weizenbaum acabou por publicar dados sobre o programa ELIZA na imprensa especializada («ELIZA – A Computer Program for the Study of Natural Language Communication Between Man and Machine», *Communications of the Association for Computing Machinery*, vol. 9, n.º 1, Janeiro de 1965, págs. 36 a 45) e dentro em pouco diversas versões do programa – baseadas na sua descrição – estavam em execução noutras instituições americanas.

Weizenbaum indica que houve três acontecimentos diferentes que o «chocaram», à medida que ELIZA se tornava conhecido. Em primeiro lugar, sentiu-se horrorizado (e parece-me difícil compreender porquê) ao descobrir que as pessoas depressa se deixavam ligar emocionalmente ao programa.

Afirma que mesmo a sua secretária, que trabalhara com ele no desenvolvimento do programa durante muitos meses, e que portanto se encontrava em óptima posição para compreender que se tratava apenas de um programa, começou a reagir a ele emocionalmente. Numa ocasião, a secretária começou a utilizar o programa e ao fim de apenas algumas frases ficou embaraçada e misteriosa. Pediu então a Weizenbaum se se importava de abandonar a sala enquanto ela continuava a «conversação».

Em certa ocasião Weizenbaum sugeriu que fosse ligada uma impressora ao computador de modo a obter uma transcrição das conversas com ELIZA. A ideia foi mal recebida, porque significaria tornar públicas conversações muito íntimas.

Weizenbaum deixou-se incomodar pela forma como as pessoas se identificavam com o programa, atribuindo-lhe uma personalidade, e partilhando com ele os seus pensamentos mais íntimos. Afirmou nunca ter previsto esse tipo de reacção a um programa relativamente simples.

## O «amigo russo»

Pamela McCorduck, no seu interessante livro *Machines Who Think* (São Francisco, W. H. Freeman and Co., 1979) também se refere aos efeitos que o programa pode ter. Recorda que a primeira vez que viu ELIZA foi no Stanford Computation Center, onde um cientista russo, reconhecido internacionalmente, estava a realizar uma visita.

O cientista sentou-se junto a um computador onde corria uma versão do programa escrito por um dos colegas de Weizenbaum, Kenneth Colby (de quem voltaremos a falar dentro em pouco). McCorduck afirma ter observado, embaraçada, enquanto o cientista – em resposta à frase FALE-ME DA SUA FAMÍLIA – começava a discutir com alguma profundidade certas preocupações pessoais, obviamente esquecendo-se daqueles que o rodeavam.

Weizenbaum verificou que alguns dos acessos ao programa, através de terminais em regime de time-sharing espalhados pela universidade, muitas vezes continuavam durante uma hora ou mais, entrando pela noite. Recebeu telefonemas de pessoas que desesperadamente desejavam aceder ao programa mesmo que por pouco tempo, a fim de resolverem os seus problemas.

Colby, que mencionámos há pouco, encontrara Weizenbaum algum tempo antes, em Stanford. Professor de Psiquiatria na UCLA, Colby interessava-se pela inteligência artificial. Pensava que as descobertas realizadas neste campo poderiam vir a aclarar certos aspectos do pensamento humano (em particular no tocante a comportamentos neuróticos). Antes de ter sido publicado o texto de Weizenbaum sobre ELIZA, o próprio Colby publicou uma nota sobre o assunto no *Journal of Nervous and Mental Diseases*.

Os dois homens afastaram-se pouco mais tarde, principalmente porque Weizenbaum discordava das afirmações de Colby segundo o qual o programa poderia ter efeitos terapêuticos; além disso parece que Colby não creditava a Weizenbaum o trabalho original de desenvolvimento de ELIZA.

Colby e dois dos seus colegas sugeriram que uma versão melhorada de DOUTOR poderia ter um uso terapêutico genuíno. Pensavam que essa versão poderia ser colocada à disposição

de hospitais mentais onde houvesse falta de pessoal, de tal modo que os doentes pudessem aceder ao programa (através de sistemas funcionando em time-sharing). Weizenbaum ficou horrorizado com esta perspectiva. Afirmou que pensava ser vital, como ponto de partida para o auxílio a pessoas que dele necessitassem para resolução dos seus problemas, um reconhecimento obviamente «humano» destes.

### **Perspectivas muito chocantes para Weizenbaum**

Weizenbaum ficou chocado com a perspectiva de um psiquiatra poder pensar que o processo de tratamento podia ser substituído por uma técnica puramente mecânica. Um tal pensamento nunca lhe surgira na mente. Mais ainda, mesmo que pudesse ser feito, não deveria sê-lo. Existem certas áreas onde as máquinas nunca devem ser aceites, afirmava Weizenbaum, mesmo que pudessem de facto aceder a elas.

Colby não se deixou abater pela reacção de Weizenbaum. Agradava-lhe a possibilidade de esta técnica se revelar eficaz. Mais ainda, defendia a sua opinião afirmando que só os leigos poderiam confundir psicoterapia com casamento. O que interessava era a existência de uma relação útil entre o terapeuta e o doente, afirmava.

Mais concretamente, Colby atacava Weizenbaum pela afirmação deste de existirem áreas onde o computador não deveria ser empregue em nenhuma circunstância. Porque não? – perguntava Colby. Apenas porque Weizenbaum o diz? Será que Weizenbaum acredita que ajudar pessoas usando um computador é pior do que deixá-las sofrer? E não deverá um terapeuta explorar todos os possíveis instrumentos, para o caso de serem eficazes?

A opinião de Colby foi mais ou menos apoiada por Carl Sagan, a quem agrada a disponibilidade de um programa como ELIZA – por alguns dólares em cada sessão – em áreas construídas especialmente.

E é aqui que ocorre o «terceiro choque» de Weizenbaum. Deixou-se primeiro impressionar pela identificação com e a inequívoca antropomorfização do programa. Depois alarmou-se

com as sugestões de que ELIZA poderia substituir, ou auxiliar, terapeutas humanos. O seu terceiro choque ocorreu ao observar que muitas pessoas vieram a acreditar que de algum modo o programa era importante na medida em que demonstrava a existência de uma solução real para os problemas da compreensão da linguagem natural pelas máquinas. Com efeito, no texto original sobre o programa, Weizenbaum preocupara-se em insistir no facto de ser impossível encontrar uma solução geral para o problema.

Já afirmei anteriormente que não concordava com tudo o que vem dito no livro de *Weizenbaum Computer Power and Human Reason*. Um dos aspectos de que discordo é que «existem certas coisas que nunca deverão ser feitas pelas máquinas». John McCarthy (1976, «An Unreasonable Book», em *Three Reviews of J. Weizenbaum's Computer Power and Human Reason*, Memo AIM-291, Stanford AI Laboratory, Novembro), avança o ponto de vista de que se existem funções que o computador não deve ser ensinado a fazer, estas coisas não devem sequer ser feitas de modo algum, tanto por uma pessoa como por uma máquina.

Outras pessoas concordam com esta opinião. No livro *Artificial Reality* (1983: Addison-Wesley Publishing Co., Reading MA, p. 168), Myron W. Krueger sugere que o horror de Weizenbaum pelo uso do seu programa para efeitos terapêuticos não tem qualquer fundo real.

No entanto, independentemente destas opiniões sobre a tese de Weizenbaum, e do valor do livro (já afirmei que em minha opinião o leitor deveria lê-lo, quanto mais não seja para compreender melhor este debate), não há dúvida de que ELIZA, tal como DOCTOR, demonstraram ser uma companhia bastante divertida. Dentro em pouco o leitor terá oportunidade de verificar este facto.

Além de apreciar o programa, o leitor poderá ainda avaliar até que ponto sugere ou não a existência de inteligência na máquina que o executa. Adrian Berry (em *The Super-Intelligent Machine*, Londres, Jonathan Cape, Ltd., 1983, p. 63) conclui que ELIZA (e PARRY, um programa que foi concebido para mimar um doente paranóico) constitui um muito mau advogado das possibilidades da verdadeira inteligência artificial.

O leitor verificará que as suas próprias ideias se modificarão à medida que usar o programa. Quando o DOUTOR produz uma observação particularmente inspirada ou apropriada, sentirá que isso indica de facto uma inteligência, pelo menos a um nível reduzido. Outras vezes, descobrirá que DOUTOR é simplesmente incapaz de perceber o que você lhe diz. Sentirá que ninguém seria incapaz de fazer perguntas tão estúpidas em situações tão descabidas (VEIO TER COMIGO PORQUE EU ESTAVA MORTO? é uma das perguntas feitas numa das versões do programa, segundo menciona Berry no livro já citado) ou de criar inocentemente respostas soberbas como se exemplifica em seguida, a partir de uma execução do programa que apresentamos neste livro:

```
Acho que este a fugir das minhas
perguntas
PARECE-LHE MUITAS VEZES QUE
FUJO DAS SUAS PERGUNTAS
Sim quando falo com um
computador
As coisas parecem acontecer
mais ou menos no momento certo
```

O que pode ser feito em resposta a uma tal afirmação além de fazer um brinde a Joseph W., progenitor de ELIZA? Será talvez chegado o momento de voltarmos ao computador.

## O programa

Agora que sabemos em que circunstâncias ELIZA foi escrito, e as reacções que suscitou, vejamos o tipo de resultados que pode produzir. A entrada humana é precedida por um sinal >? sendo cada observação humana e respectiva resposta isoladas por uma linha de traços:

```
O DOUTOR ESTA'...
POR FAVOR PREPARE-SE

BEMVINDO A OUTRA SESSAO
COM O DOUTOR...

OLA'...
```

-----  
>? OLÁ! DOUTOR

OLÁ!... ACALME-SE, E FALEMOS DA  
SUA SITUAÇÃO

-----  
>? TENHO UM PROBLEMA GRAVE  
EM QUE PODEREI AJUDÁ-LO

-----  
>? TEM A VER COM O MEU PROFESSOR  
MALUCO

TEM A CERTEZA DE QUE TEM A VER  
COM O SEU PROFESSOR MALUCO

-----  
>? SIM TENHO A CERTEZA  
PARECE MUITO CERTO...

-----  
TENHO PENSADO NO ASSUNTO DIA E  
NOITE

FALOU-ME NO SEU PROFESSOR MALUCO  
FALA-ME MAIS SOBRE O ASSUNTO

-----  
>? TUDO COMEÇOU QUANDO EU TINHA  
DEZ ANOS DE IDADE

MUITO INTERESSANTE

-----  
AGORA TENHO MEDO DE IR À ESCOLA

VEIO FALAR COMIGO PORQUE TEM  
MEDO DE IR A ESCOLA AGORA

-----  
ACHO QUE SIM

MAS NAO TEM A CERTEZA DISSO

-----  
>? TALVEZ HAJA OUTRAS RAZOES

ISSO E MUITO INTERESSANTE

-----  
>? TAMBEM TENHO PROBLEMAS COM O  
MEU IRMAO LOUCO

FALE-ME MAIS DISSO

-----  
ACHO QUE ELE ME ODEIA

DUVIDA DE QUE ELE O ODEIA

-----  
>? ACHO QUE ESTA A FUGIR DAS  
MINHAS PERGUNTAS

PARCE-LHE MUITAS VEZES QUE FUGO  
DAS SUAS PERGUNTAS

-----  
>? SIM QUANDO FALO COM UM  
COMPUTADOR

AS COISAS PARECEM ACONTECER MAIS  
OU MENOS NO MOMENTO CERTO

>? ISSO E' UMA RESPOSTA ESTUPIDA  
DE UM COMPUTADOR

QUE ASPECTO DOS COMPUTADORES LHE  
INTERESSA MAIS

-----

>? AJUDAR-ME NOS MEUS PROBLEMAS  
FALE-ME SOBRE ISSO

-----

>? QUERO BATER NO MEU IRMAO  
QUE MELHORARIA NA SUA VIDA SE  
FIZESSE ISSO

-----

>? SENTIR-ME BEM  
EXPLORE UM POUCO ESSA FRASE

-----

>? SINTO-ME INFERIOR QUANDO ELE  
ME ABESQUINHA  
GOSTA DE SE SENTIR INFERIOR  
QUANDO ELE O ABESQUINHA

-----

CLARO QUE NAO GOSTO  
PENSA HAVER ALGUMA RELACAO ENTRE  
ISSO E O SEU IRMAO

-----

>? A RELACAO E' OBVIA  
FALE-ME SOBRE ISSO

>? DESEJO AUXILIO CONTRA OS MEUS  
DECIOS

QUE SIGNIFICARIAM PARA SE TER  
AUXILIO CONTRA OS SEUS DECIOS

-----

>? ACIO QUE ME SENTIRIA MAIS  
FORTE

DUVIDA DE QUE SE SENTIRIA MAIS  
FORTE

-----

>? APETECE-ME SEMPRE FUGIR

NAO SEJA: 'SEMPRE' UM POUCO  
FORTE

-----

>? TALVEZ SEJA

PORQUE NAO VOLTAMOS ATRAS E  
DISCUTIMOS MELHOR OS SEUS  
DECIOS

-----

Este exemplo de execução é impressionante, e depois de o leitor usar o programa algumas vezes descobrirá certamente o tipo de frases e palavras a que melhor responde. Constitui por outro lado uma boa demonstração para aqueles que não estão habituados a computadores.

### Como funciona

A estrutura do programa é, na sua essência, simples. O computador aceita uma cadeia de entrada do utilizador, percorre-

-a, procurando as frases capazes de desencadear uma resposta (como PENSO ou PORQUE NÃO POSSO) guardada em memória. Se não encontra nenhuma, procurará palavras que para ele sejam significativas (por exemplo COMPUTADOR), escolhendo do mesmo modo uma das três respostas pré-programadas para o efeito.

Se não encontra qualquer palavra, procura descobrir se dispõe de alguma frase do próprio utilizador, guardada numa das entradas anteriores, e se assim acontecer, constrói uma pergunta que a utilize. Se tudo isto falhar, escolherá entre um conjunto de respostas (NÃO COMPREENDO BEM O QUE DIZ, etc.), concebida meramente para manter a conversação.

```
FALOU-ME 'A POUCO DA SUA ATITUDE  
FALE-ME SOBRE ISSO
```

```
-----  
>? E' UMA PREOCUPACAO CONSTANTE
```

```
QUE CERTEZA TEM EM SER UMA  
PREOCUPACAO CONSTANTE PARA SI
```

```
-----  
>? PORQUE DE FACTO ASSIM E'
```

```
QUE OUTRAS RAZOES LHE OCBREM
```

### **A base de dados de resposta**

A parte mais importante do programa, e aquela que pode ser mais divertido modificar ou acrescentar, é o enorme conjunto de declarações DATA no final (o leitor verificará que a última declaração deste tipo contém uma série de asteriscos. Estes servem para completar a quantidade de dados esperada pela instrução READ, e para lhe permitir acrescentar alguns novos dados de sua autoria sem ser forçado a contar os que já existem para fazer concordar a instrução READ e as declarações DATA. As declarações DIM dimensionam os quadros alfanuméricos



```

0050 DATA "QUE OUTRAS RAZOES HE
0060 OCCORREM?"
0080 DATA "QUE MAIS EXPLICA ISSO
0090 ="

```

De maior interesse são as frases a usar no início das respostas do computador, sendo o equilíbrio da resposta produzido pela entrada inicial do utilizador (depois de terem sido realizadas as trocas de «conjugação» necessárias):

```

1460 REM * produzir palavras*
1470 DATA "RESPOSTA *"
1470 DATA "PRECISO"
1480 DATA "PORQUE PRECISO*"
1490 DATA "SERIA DE FACTO VARIU
0000 DATA "TIVESSE*"
1500 DATA "TEM A CERTEZA DE SE
1510 DATA "ESTOU"
1520 DATA "VEIO FALAR COMIGO DE
0000 DATA "
1430 DATA "DESDE QUANDO ESTÁ"
1470 DATA "PENSA QUE É NORMA E
1470 DATA "
1490 DATA "E"
1500 DATA "PENSA QUE É*"
1500 DATA "EM QUE CIRCUNSTANCIA
1510 DATA "
1540 DATA "PODERIA ACONTECER DE
1550 DATA "

```

É fácil ver que cada uma das frases que fazem parte da resposta termina com um asterisco, usado pelo computador como uma indicação de que parte da entrada original deve ser modificada a fim de completar a frase.

Vejamos como tudo isto funciona na prática. Sonhamos que o utilizador escreveu:

### QUERO MOSTRAR-LHE A VERDADE

O DOUTOR varre o conteúdo do quadro C\$, descobre o elemento que contém «QUERO». Os elementos equivalentes dos quadros D\$, E\$ e F\$ contêm as porções iniciais das respostas apropriadas, como se pode ver:

```

1750 DATA "DESEJO"
1760 DATA "QUE SIGNIFICARIA DE
1770 DATA "

```

1770 DATA "PORQUE DESEJA\*"  
1780 DATA "QUE MELHORARIA NA SUA  
VIDA SE TIVESSE\*"

O DOUTOR produz então um número aleatório entre um e três, imprimindo o elemento D\$ se for um, E\$ se for dois, e F\$ se for três (depois de verificar previamente se termina por um asterisco, anotando o facto e retirando este antes de imprimir). Consideremos que o computador escolheu a resposta D\$. A sua resposta, até agora, é:

### PORQUE QUER

Em seguida, o computador analisa o equilíbrio da frase do utilizador (o material que se segue a QUERO), tratando-o exactamente do mesmo modo (usando o mesmo código, aliás) a que recorreremos no programa de tradução mais adiante, trocando os pronomes, formas verbais, etc. Imprime cada palavra à medida que a processa, deixando a palavra do utilizador sem qualquer modificação se não houver necessidade de trocas na secção de conjugação.

A frase original...

### QUERO MOSTRAR-LHE A VERDADE

seria portanto transformada em...

### PORQUE QUER MOSTRAR-ME A VERDADE

É apenas isto que o programa faz, mas basta para criar um efeito notável.

Se o programa não consegue descobrir uma frase para combinar com parte da entrada do utilizador, o DOUTOR procura uma palavra nessas condições em qualquer ponto da entrada (e não apenas no princípio, como quando procura frases). Entre estas palavras incluem-se COMPUTADOR e AMIGOS, produzindo resultados como os seguintes:

PORQUE SE REFERE AO TEMA DOS AMIGOS?  
POR FAVOR FALE-ME MAIS DOS SEUS AMIGOS

e...

QUAIS OS ASPECTOS DOS COMPUTADORES  
QUE MAIS LHE INTERESSAM  
ESTÁ A PENSAR EM MIM EM PARTICULAR?

## A estrutura do programa

O programa inicia-se, tal como outros deste livro, pela chamada de uma subrotina de inicialização (que se inicia em 1140). Depois disto, o programa imprime uma linha em branco, e em seguida a linha tracejada e uma nova linha em branco.

```
10 REM DOUTOR
15 BRIGHT 1: FLASH 0: PAPER 1:
INK 7: BRIGHT 0: BORDER 1: INVE
RSE 0: CLS
20 GO SUB 1140: REM INICIALIZA
30 PRINT : PRINT INK 6: BRIGHT
1:
-----
40 POKE 23650,200: PRINT ">":
INPUT X$: PRINT X$
50 IF X$="" THEN STOP: REM DE
SISTIR CARREGANDO EM ENTER
60 PRINT
70 IF X$=Z$ THEN PRINT "POR FA
VOR NÃO SE REPITA": GO TO 30
80 LET Z$=X$: LET X$=X$+" "
90 IF X$(1 TO 5)="ADEUS" THEN P
RINT "OK, ATÉ À PROXIMA...": S
TOP
```

A linha 40 imprime o símbolo «>», aceitando em seguida a entrada do utilizador. Se a entrada for uma cadeia vazia (isto é, o utilizador limitou-se a carregar em ENTER, sem escrever qualquer letra), o programa termina.

A linha 60 imprime uma linha em branco, e a linha 70 compara esta entrada (X\$) com a dada da vez anterior (Z\$), e se verificar que são iguais imprime «POR FAVOR NÃO SE REPITA», voltando à linha 30 para uma nova entrada. A linha 80 atribui a nova entrada a Z\$, para verificação da vez seguinte. Se a entrada contém inicialmente ADEUS o computador despede-se e pára a execução.

Tendo sobrevivido a tudo isto, o programa começa a «pensar»:

```
100 REM ** procurar frases que
produzem respostas no início da
entrada **
```

```

110 LET I=0
120 LET I=I+1
130 LET IN=CODE C#(I,1)+40
140 IF IN X#( TO IN-1)=C#(I,2 TO I
N) THEN GO TO 360: REM descobert
2 frase
150 IF L&&K THEN GO TO 120

```

Usando os elementos do quadro C\$, varre a primeira porção da entrada, procurando encontrar algo semelhante. Se assim acontecer, a acção passa para a linha 360 onde é executada a comparação entre a frase de entrada e o resto da entrada do utilizador:

```

360 REM ** User frase descobert
2 do inicio da entrada **
370 LET t=INT (RND*3)+1
380 IF t=1 THEN LET q#=#(I)
390 IF t=2 THEN LET q#=#(I)
400 IF t=3 THEN LET q#=#(I)
410 REM ** verificar se termo
360 teste de q#, e se e o mesmo
2 teste de entrada **
420 IF q#=#(I) THEN GO TO 1 STEP -1: IF q
#(I) = " " THEN NEXT I
430 IF q#(I) = "*" THEN LET frase =
1: LET q#=#( TO I-1)
440 IF INT q#(I) = " "
450 IF q#(I) THEN GO TO 360: RE
M ** preciso mais material
460 REM ** user equilibrio de e
2 teste de **
470 IF t=LEN X# TO 1 STEP -1:
IF X#(I) = " THEN NEXT I
480 LET q#=#( TO I)
490 IF X#(I) = "+X#(I+1 TO LEN X
#)+ "
490 IF *****
490 IF *****
500 IF *****
510 IF *****
520 LET q#=#(I)
530 LET q#=#(I)
540 IF q#=#(I) THEN GO TO 360
550 IF X#(I) = " THEN GO TO 370
560 GO TO 360
570 LET X=#+1
580 LET q#=#+1

```

```

0100 LET Y=Y+1
0110 IF LEN X#<X+Y THEN LET X#=X
#+
0120 IF X#(X+Y)="*" THEN LET A#=
X#(X TO X+Y-1): GO TO 030
0130 IF X+Y>255 THEN GO TO 530
0140 GO TO 0100
0150 LET M#=#
0160 LET M#=#M#+1
0170 IF A#="MEU" AND K#="" THEN
LET K#=#(X+0 TO LEN X#-4)
0180 IF A#=#(M#,2 TO CODE A#(M#
+1)-40) THEN PRINT B#(M#,2 TO CO
DE B#(M#,1)-40); " "; GO TO 030
0190 IF M#<K# THEN GO TO 040
0200 PRINT A#; " ";
0210 GO TO 030

```

A parte que se segue à linha 370, até à 400, escolhe um dos três inícios de resposta, entre D\$(n), E\$(n) e F\$(n). A linha 420 passa a zero uma «flag», usando em seguida a linha 430 para verificar se a frase escolhida termina por um asterisco (indicativo, como o leitor recordará, que se trata apenas de uma resposta parcial, necessitando de mais material obtido na entrada do utilizador).

Se encontra um asterisco no final, a flag é passada a um e a parte final da linha 430 retira o último carácter da frase.

A linha 440 imprime a frase escolhida. Se FLAG é ainda igual a zero (linha 450), o programa volta à linha 30 para obter a entrada seguinte. Se não, o DOUTOR deve verificar o equilíbrio da entrada do utilizador, realizando as mudanças de conjugação necessárias (usando, como já disse, o mesmo código empregue no programa de tradução apresentado adiante).

Por outro lado, como se indica na linha 500, o programa procura a palavra MEU para definir K\$, a flag intitulada «my-flag». Se encontra aquela palavra na entrada (como acontece na frase TEM A VER COM O MEU PROFESSOR MALUCO) e MYFLAG não tem qualquer valor, utilizará o equilíbrio da entrada do utilizador a partir da palavra MEU e atribuirá a K\$ - neste caso - a expressão PROFESSOR MALUCO. Mais tarde, se o DOUTOR não consegue encontrar uma frase ou palavra que provoque uma resposta, poderá usar K\$ juntamente com outras frases (como HÁ POUCO FALOU DO SEU PRO-

FESSOR MALUCO) para manter a conversação. O efeito deste pequeno truque sobre os utilizadores é notável.

Se não tiver sido descoberta qualquer frase susceptível de provocar uma resposta, o computador procura uma palavra nessas condições, usando a seguinte parte do programa:

```

160 REM ** Programa vem aqui se
descobre frase **
170 REM ** procurar palavras-ch
ave na entrada **
180 LET X#="" "+X#+ "
190 LET M=LEN X#
200 LET L=0
210 LET L=L+1
220 IF L=#-1 THEN GO TO 260: RE
M
230 H=ENCONTRA
240 IF X#(L)=H " THEN GO TO 250
250 GO TO 210
260 LET X=L+1
270 LET U=0
280 LET U=U+1
290 IF X#(X+U)=" " THEN LET A#=
X#(XTOX+U): GO TO 300
300 GO TO 270
310 LET A=A+1
320 IF A#(1 TO LEN A#)=C#(0,2 TO
CODE C#(0,1)-40) THEN GO TO 310
: REM ** descobre a palavra-chave
330 IF A# THEN GO TO 310
340 GO TO 210

```

### Uso de myflag

Se não encontra nada, o computador volta à MYFLAG (se existir) ou usa uma frase aleatória (no caso de MYFLAG ser uma cadeia vazia):

```

350 REM *respostas aleatorias*
** sem chave **
360 LET t=INT (RAND*8)+1
370 IF t=1 THEN PRINT "QUE LHE
SUGERE ISSO?": GO TO 30
375 IF t=2 THEN PRINT "NAO TENH
O A CERTEZA DE ESTAR A COMPREEN
DER BEM": GO TO 30
380 IF t=3 THEN PRINT "POR FAVO
R ESCLAREÇA-ME MELHOR": GO TO 30

```



foram apenas pensadas para manter a conversação.

Agora que o leitor conhece o funcionamento do programa, convirá apresentar a listagem completa deste:

```
10 REM DOUTOR
15 BRIGHT 1: FLASH 0: PAPER 1:
INK 7: BRIGHT 0: BORDER 1: INVE
RSE 0: OLS
20 GO SUB 1140: REM INICIALIZA
R
30 PRINT : PRINT INK 6: BRIGHT
17:"-----"
-----"
40 POKE 20656,200: PRINT ">"):
INPUT X$: PRINT X$
50 IF X$="" THEN STOP : REM DE
SISTIR CARREGANDO EM ENTER
60 PRINT
70 IF X#=Z$ THEN PRINT "POR FA
VOR NÃO SE REPITA": GO TO 30
80 LET Z$=X$: LET X$=X$+" "
"
90 IF X$( TO 5)="ADEUS" THEN P
RINT "OK, ATE" "A PROXIMA...": S
TOP
100 REM ** procurar frases que
produzem respostas no inicio de
entrada **
110 LET L=0
120 LET L=L+1
130 LET Ln=CODE c$(L,1)-48
140 IF X$( TO Ln+1)=c$(L,2 TO L
n) THEN GO TO 360: REM descobert
a frase
150 IF L&K THEN GO TO 120
160 REM ** Programa vem aqui se
descobre frase **
170 REM ** procurar palavras-ch
ave na entrada **
180 LET X$=" "+X$+" "
190 LET M=LEN X$
200 LET L=0
210 LET L=L+1
220 IF L=M-1 THEN GO TO 360: RE
M não encontra
230 IF X$(L)= " " THEN GO TO 250
240 GO TO 210
250 LET X=L+1
260 LET M=0
270 LET C=C+1
```







```

1270 IF S#(K,2)="#" THEN GO TO
1280
1290 GO TO 1250
1300 LET K=0
1310 LET K=K+1
1320 PRINT C$(K),d$(K),e$(K),f$(K)
1331 FOR I=10 TO 1 STEP -1: IF C
$(K,I)="#" THEN NEXT I
1340 LET C$(K)=CHR$(I+41)+C$(K)
TO 140
1350 IF f$(K,1)="#" THEN GO TO 1
340
1360 GO TO 1300
1370 GOTO
1380 PRINT "SERVINDO A OUTRA SES
1390
1400 PRINT "COM O DOUTOR..."
1410 PRINT "OLA..."
1420 RETURN
1430 PRINT *****
1440 PRINT *****
1450 PRINT *****
1460 PRINT *****
1470 PRINT *****
1480 PRINT *****
1490 PRINT *****
1500 PRINT *****
1510 PRINT *****
1520 PRINT *****
1530 PRINT *****
1540 PRINT *****
1550 PRINT *****
1560 PRINT *****
1570 PRINT *****
1580 PRINT *****
1590 PRINT *****
1600 PRINT *****
1610 PRINT *****
1620 PRINT *****
1630 PRINT *****
1640 PRINT *****
1650 PRINT *****
1660 PRINT *****
1670 PRINT *****
1680 PRINT *****
1690 PRINT *****
1700 PRINT *****
1710 PRINT *****
1720 PRINT *****
1730 PRINT *****
1740 PRINT *****
1750 PRINT *****
1760 PRINT *****
1770 PRINT *****
1780 PRINT *****
1790 PRINT *****
1800 PRINT *****
1810 PRINT *****
1820 PRINT *****
1830 PRINT *****
1840 PRINT *****
1850 PRINT *****
1860 PRINT *****
1870 PRINT *****
1880 PRINT *****
1890 PRINT *****
1900 PRINT *****
1910 PRINT *****
1920 PRINT *****
1930 PRINT *****
1940 PRINT *****
1950 PRINT *****
1960 PRINT *****
1970 PRINT *****
1980 PRINT *****
1990 PRINT *****
2000 PRINT *****

```

16000 DATA "PORQUE LHE INTERESSA  
SE ESTOU QUANDO\*"  
16110 DATA "PREFERIRIA QUE EU NAO  
\*"  
16200 DATA "TALVEZ POR VEZES VOCE  
CULPE QUANDO ESTOU\*"  
16300 DATA "NAO POSSO"  
16400 DATA "COMO SABE QUE NAO POD  
M\*"  
16500 DATA "MAS TENTOU?"  
16600 DATA "TALVEZ AGORA POSSA\*"  
16700 DATA "ESTOU"  
16800 DATA "VEIO FALAR COMIGO POR  
QUE ESTO\*"  
16900 DATA "DESSA QUANDO ESTA\*"  
17000 DATA "PENSA QUE E' NORMAL E  
ESTAR\*"  
17100 DATA "SOU"  
17200 DATA "GOSTA DE SER\*"  
17300 DATA "PORQUE ME DIZ QUE E' \*"  
\*"  
17400 DATA "PORQUE E' \*"  
17500 DATA "COMO EUO"  
17600 DATA "QUE SIGNIFICARIA PARA  
M\*"  
17700 DATA "PORQUE OBARBARIA\*"  
17800 DATA "QUE MELHORIA NA SUA  
VIDA\*"  
17900 DATA "O QUE"  
18000 DATA "COMO SE PARESENTA?"  
18100 DATA "COMO PODEMOS AJUDA-LO  
COMO ESTA PARESENTA?"  
18200 DATA "COMO LHE PARECE?"  
18300 DATA "COMO PODERIA RESOLVER  
O PROBLEMA?"  
18400 DATA "SERIA MELHOR RESPONDE  
MIGO"  
18500 DATA "QUEM SE ESTA DE FACTO  
PREGUNTAR?"  
18600 DATA "QUEM?"  
18700 DATA "MUITAS VEZES ME  
PREGUNTAM?"  
18800 DATA "MUITAS VEZES O AVALIA  
MIGO"  
18900 DATA "QUEM LHE PARECE\*"  
19000 DATA "QUEM"  
19100 DATA "E' UMA PERGUNTA ESTUP  
IDA\*"  
19200 DATA "PRECISA MESMO DE SABE  
CNO\*"

1940 DATA "QUE SIGNIFICARIA PARA  
 SI EU "DIZER-LHE ONDE?"  
 1950 DATA "QUANDO"  
 1960 DATA "COMO POSSO SABER QUAN  
 DO?"  
 1970 DATA "O MOMENTO NAO DEVE SE  
 R DISCUTIDO"  
 1980 DATA "AS COISAS ACONTECEM M  
 AIS OU "MENOS NO MOMENTO CERT  
 O"  
 1990 DATA "PORQUE"  
 2000 DATA "PORQUE NAO ME DIZ POR  
 QUE RAZAO  
 \*"  
 2010 DATA "QUE ME TERA' DITO QUE  
 ME PERMITA  
 DIZER-LHE PORQUE?"  
 2020 DATA "DESEJA VERDADEIRAMENT  
 E SABER  
 PORQUE?"  
 2030 DATA "PORQUE?"  
 2040 DATA "E' ESSA A VERDADEIRA  
 RAZAO?"  
 2050 DATA "QUE OUTRAS RAZOES LHE  
 OCORREM?"  
 2060 DATA "QUE MAIS EXPLICA ISSO  
 ?"  
 2070 DATA "PERDOO"  
 2080 DATA "EM QUE OUTRAS CIRCUNS  
 TANCIAS SE  
 DESOULPA?"  
 2090 DATA "MUITAS VEZES NAO E' N  
 ECESSARIA  
 QUALQUER DESOULPA"  
 2100 DATA "QUE SENTE QUANDO PEDE  
 DESOULPA?"  
 2110 DATA "OLA"  
 2120 DATA "OLA"... PRAZER EM VE-  
 LO!"  
 2130 DATA "VIVA,... ESTOU SATISFE  
 ITO POR  
 VE-LO HOJE"  
 2140 DATA "OLA"... APETECIA-ME C  
 ONVERSAR  
 CONSIGO"  
 2150 DATA "VIVAI!"  
 2160 DATA "COMO ESTA'? AGRADA-ME  
 VE-LO"  
 2170 DATA "AINDA BEM QUE APARECE  
 C..."  
 2180 DATA "OLA"... ACALME-SE, E  
 TALEMOS DA  
 SUA SITUACAO"  
 2190 DATA "TALVEZ"  
 2200 DATA "PARECE UM POUCCO HESIT  
 ANTE"  
 2210 DATA "ESTA' MUITO INDECISO,  
 .."

2200 DATA "EM QUE OUTRAS SITUACO  
 2205 DE SENTENCAO INDECISO?"  
 2210 DATA "NAO"  
 2215 DATA "PORQUE E' TAO PESSIMI  
 2220 STA QUANTO A ISSO?"  
 2225 DATA "ESTA' A DIZER ISSO SO  
 2230 PARA SER NEGATIVO?"  
 2235 DATA "PARECE-ME MUITO FORCA  
 2240 DO, QUE LHE SUGERE ISSO?"  
 2245 DATA "SEMPRE"  
 2250 DATA "POR FAVOR, DE-ME UM E  
 2255 XEMPLO ESPECIFICO"  
 2260 DATA "NAO E' SEMPRE UM POU  
 2265 O FORTE?"  
 2270 DATA "QUANDO?"  
 2275 DATA "PENSO"  
 2280 DATA "DUVIDA DE QUE\*"  
 2285 DATA "PENSA DE FACTO ASSIM?  
  
 2340 DATA "MAS NAO TEM A CERTEZA  
 2345 DE\*"  
 2350 DATA "AMIGO"  
 2355 DATA "PORQUE REFERE O PROBL  
 2360 EMA DOS AMIGOS?"  
 2365 DATA "QUAL E' A SUA MELHOR  
 2370 RECORDACAO DE UM AMIGO?"  
 2375 DATA "POR FAVOR DIGA-ME MAI  
 2380 S SOBRE ESSA AMIZADE..."  
 2385 DATA "AMIGOS"  
 2390 DATA "DE QUE MODO O AFECTAM  
 2395 AS REACOES DOS SEUS AMI  
 2400 GOS?"  
 2405 DATA "O QUE O LEVOU A FALAR  
 2410 DOS AMIGOS AGORA?"  
 2415 DATA "DE QUE MODO O INFERIO  
 2420 RIZAM OS AMIGOS?"  
 2425 DATA "SIM"  
 2430 DATA "O QUE ESTA' POR TRAS  
 2435 DESSE CERTEZA?"  
 2440 DATA "ESTA' MUITO CONVICTO,  
 2445 ."  
 2450 DATA "PARECE TER UMA CERTEZ  
 2455 A QUANTO A ISSO"  
 2460 DATA "COMPUTADOR"  
 2465 DATA "QUE SENTE AO FALAR-ME  
 2470 ASSIM?"  
 2475 DATA "ESTA' A PENSAR EM MIM  
 2480 ."  
 2485 DATA "QUAL O ASPECTO DOS CO  
 2490 MPUTADORES QUE MAIS LHE INTERESS  
 2495 A?"

2510 DATA "SERÁ"  
 2520 DATA "PENSA QUE É\*"  
 2530 DATA "EM QUE CIRCUNSTANCIAS  
 PODERIA\*"  
 2540 DATA "PODERIA ACONTECER QUE  
 \*"  
 2550 DATA "É"  
 2560 DATA "QUE CERTEZA TEM DE QU  
 E"  
 2570 DATA "TEM A CERTEZA DE QUE  
 E"  
 2580 DATA "QUE SENTIRIA SE EU LH  
 A DISSÉSSE QUE PROVAVELMENTE NAO  
 E\*"  
 2590 DATA "PODERÁ"  
 2600 DATA "QUE O LEVA A PENSAR Q  
 UE NAO  
 CONSIGO\*"  
 2610 DATA "NAO LHE PARECE QUE PO  
 DERIA\*"  
 2620 DATA "TALVEZ GOSTASSE DE PO  
 DERA\*"  
 2630 DATA "PODEREI"  
 2640 DATA "TALVEZ NAO QUEIRA DE  
 FACTO\*"  
 2650 DATA "QUER SER CAPAZ DE\*"  
 2660 DATA "DUVIDO DISSO"  
 2670 DATA "VOCE É"  
 2680 DATA "PORQUE PENSA QUE SOU\*  
 2690 DATA "AGRADA-LHE PENSAR QUE  
 SOU\*"  
 2700 DATA "TALVEZ LHE AGRADASSE  
 SER\*"  
 2710 DATA "TU ES"  
 2720 DATA "PORQUE PENSA QUE SOU\*  
 2730 DATA "AGRADA-LHE PENSAR QUE  
 SOU\*"  
 2740 DATA "PORQUE DIZ QUE SOU\*"  
 2750 DATA "NAO SOU"  
 2760 DATA "NAO É? "  
 2770 DATA "PORQUE NAO É\*"  
 2780 DATA "QUER SER\*"  
 2790 DATA "SINTO"  
 2800 DATA "FALE-ME MAIS DO QUE S  
 ENTE"  
 2810 DATA "SENTE MUITAS VEZES IS  
 SO? "  
 2820 DATA "GOSTA DE SENTIR\*"  
 2830 DATA "EU SINTO"  
 2840 DATA "EXPLOREMOS UM POUCC E

285A AFIRMACAO"  
 2850 DATA "SENTE-SE ASSIM MUITAS  
 VEZES?"  
 2860 DATA "QUE EMOCOES PROVOCA E  
 N SI ESSE SENTIMENTO?"  
 2870 DATA "TENHO"  
 2880 DATA "PORQUE ME DIZ QUE TEM  
 \*"  
 2890 DATA "E' OBVIO QUE TEM\*"  
 2900 DATA "COMO O PODEREI AJUDAR  
 NISSO?"  
 2910 DATA "GOSTARIA"  
 2920 DATA "PODE-ME EXPLICAR PORO  
 LE GOSTARIA\*"  
 2930 DATA "A QUEM MAIS DISSE QUE  
 GOSTARIA \*"  
 2940 DATA "QUE CERTEZA TEM DE QU  
 E GOSTARIA \*"  
 2950 DATA "HAVERA \*"  
 2960 DATA "CLARO QUE HA \*"  
 2970 DATA "E' PROVAVEL QUE HAJA\*  
 2980 DATA "GOSTARIA QUE HOUVESSE  
 \*"  
 2990 DATA "O MEU"  
 3000 DATA "O SEU\*"  
 3010 DATA "COMPREENDO, O SEU\*"  
 3020 DATA "QUE SIGNIFICA PARA SI  
 QUE O SEU \*"  
 3030 DATA "UOOS"  
 3040 DATA "ESTA SESSAO E' PARA A  
 AJUDA' -LO A SI; NAO PARA DISCUTIR  
 A MINHA PESSOA!"  
 3050 DATA "QUE O LEVOU A DIZER I  
 3060 SOBRE MIM?"  
 3070 DATA "ESTOU A TIRAR NOTAS E  
 3080 SOBRE TUDO ISSO PARA RESOLVER A  
 3090 SUA SITUACAO"  
 3070 DATA "ODEIO"  
 3080 DATA "PORQUE ODEIA\*"  
 3090 DATA "HA' QUANTO TEMPO ODEI  
 A\*"  
 3095 DATA "TODOS OS SEUS AMIGOS  
 ODEIAM \*"  
 3100 DATA " \*", " \*", " \*", " \*"

## TRADUÇÃO MECÂNICA

Ao pensar-se em algumas das possibilidades que adviriam da capacidade das máquinas compreenderem e processarem a linguagem natural, dir-se-ia que os computadores poderiam ser muito úteis para a tradução das línguas humanas. Desde o início dos anos cinquenta que esta esperança é acalentada, e fizeram-se já grandes progressos neste campo.

Existem actualmente mais de vinte sistemas de tradução mecânica (Machine Translation – MT) em uso no mundo. Mas, ao contrário do que geralmente se pensa, estes sistemas não funcionam aceitando um documento numa língua por uma abertura e apresentando noutra uma tradução automática do mesmo documento. O funcionamento destas máquinas de tradução é mais subtil, e mais complexo. De facto, existem vários subcampos neste domínio da tradução mecânica.

### **Os seres humanos ainda são úteis**

Se bem que, nos primeiros tempos da construção destes sistemas, tivesse sido aceite (talvez sem pensar muito no assunto) que os tradutores humanos acabariam por tornar-se redundantes à medida que as máquinas se tornassem mais capazes, os investigadores acabaram por aceitar que por enquanto (e no futuro próximo) o papel dos tradutores humanos é vital. Os especialistas neste campo falam agora de «pré-tradução mecânica», equivalendo os documentos produzidos pelos sistemas

MT a simples esboços grosseiros depois utilizados pelos seres humanos para a criação de obras definitivas.

Existem diferentes formas de pensar a tradução mecânica actualmente em uso. Entre elas incluem-se as que deram origem a sistemas construídos com o objectivo de traduzirem documentos escritos numa versão limitada da linguagem natural, ou documentos montados de modo a facilitar o manuseamento pela máquina. A Xerox possui um sistema deste tipo, designado por SYSTRAN. Observaremos dentro em pouco um exemplo das saídas produzidas pelo SYSTRAN (usado para documentos da CEE).

Um outro método permite ao utilizador modificar o sistema de acordo com as suas necessidades, dando-lhe um vocabulário adaptado aos temas específicos em causa. Este sistema, chamado CULT, é vulgarmente usado em Hong-Kong para traduzir revistas chinesas de matemáticas. A saída das máquinas é vendida a bibliotecas em todo o mundo.

Quando nós, os leigos, pensamos em tradução mecânica, pensamos em máquinas onde se põe inglês num lado e sai português pelo outro, e de facto é este o objectivo final daqueles que desenvolvem estes sistemas. Está-se no entanto ainda muito longe disso. Porém, o sistema SYSTRAN – mencionado há pouco como capaz de trabalhar com documentos escritos numa «sub-linguagem» ou previamente montados – pode ser usado para qualquer documento. O êxito obtido varia de documento para documento.

Muitos documentos passam por uma fase de montagem prévia antes de serem apresentados à máquina. São então eliminadas as ambiguidades potenciais, assim como outros aspectos do texto que poderiam iludir a máquina. Muitos documentos (aliás a maior parte) devem depois ser novamente trabalhados, corrigindo a saída da máquina. Nesta fase procuram-se erros de tradução, e corrige-se a sintaxe.

Alguns documentos não necessitam desta correcção final. Para certos fins, a saída directa do sistema pode ser suficiente.

A tradução mecânica pode também ser executada com o auxílio de um tradutor humano, que intervirá durante a execução da tradução.

Como podemos concluir do que foi dito, o papel dos

humanos é ainda vital no processo de tradução. E não há qualquer indicação de que isto se venha a alterar num futuro próximo. As máquinas podem fazer uma tradução bastante grosseira, necessitando depois de ser «polida» por seres humanos.

Vejamus um exemplo de tradução por máquina. Trata-se de um documento da CEE traduzido de francês para inglês pelo sistema SYSTRAN em 1981.

Eis o início do documento em francês:

«Application de la micrologique au controle des opérations de production.

But de la recherche:

Perfectionner les appareillages existants de sorte que les préposés soient débarassés des tâches dans lesquelles leur jugement n'intervient pas.

Application au central de télésurveillance d'engins sur pneus.»

A máquina respondeu com a seguinte tradução:

«Application of micrological to the control of the production operations.

Aim of research:

To improve existing equipments so that the officials debarasses tasks in which their judgement does not intervene.

Application to the exchange of telesurveillance of equipment on tyres.»

Se bem que esta tradução seja bastante grosseira, consegue-se ter uma ideia razoável do significado das frases. O «debarasses» que sobrevive na tradução inglesa deve-se, de facto, a um erro de escrita no original francês (que deveria ser «debarasses», palavra que em princípio a máquina deveria traduzir).

Depois da correcção humana, o documento assumiu a seguinte forma:

«Application of micrology to the monitoring of production operations.

Aim of the research:

To perfect existing apparatus so that staff can be relieved of tasks where no judgement is required.

Application to the remote monitoring station for trackless vehicles.»

Achei fascinante seguir a evolução do documento. Exceptuando a última linha, a versão final inglesa não é muito diferente do original SYSTRAN.

Nem todo o documento foi traduzido com êxito. A pessoa que corrigiu o documento tratou drasticamente o texto que se seguia, reduzindo a saída MT a uma sombra.

Vejam os que a máquina imprimiu:

«It publishes station and day reports indicating the duration and the importance relative of the periods devoted by each instrument supervised to the various possible activities: evacuation of the products, transport of equipment, maintenance, station service... as well as the number of evacuated coal cups.»

Trata-se de um texto desastroso, com frases como «the importance relative of the periods» mostrando claramente a origem francesa do texto.

Depois de corrigido, o texto foi transformado no seguinte:

«It publishes shift and day reports indicating the duration and the relative portion of time spent by each vehicle recorded on the various possible tasks: coal clearance, materials transport, maintenance, refuelling points... as well as the number of coal buckets carried.»

Finalmente, antes de passarmos à criação do nosso programa «tradutor», é interessante notar que a vasta maioria dos documentos que actualmente usam a MT não são literários. A tradução de obras literárias constitui um campo inteiramente diferente, e no que se refere à tradução mecânica neste campo, encontra-se ainda na sua infância.

## «Portugalês»

O programa que se segue, usando um vocabulário muito reduzido, aceita uma entrada em inglês, e dá-nos uma estranha mistura de palavras nessa língua e em português. As palavras mais simples e óbvias são traduzidas para português, e as difíceis são deixadas em inglês (esta técnica permitiria produzir, por exemplo, frases como SOU UM HOMEM EXASPERATED...) A revista Punch possui uma coluna regular chamada «Let's talk Franglais» que mostra os resultados mais curiosos da tradução do inglês para francês usando este método.

O programa aqui apresentado não foi concebido de forma muito séria. É no entanto suficiente para indicar alguns dos problemas da tradução mecânica. Se fosse mais completo, dispondo de um vocabulário mais extenso, poderia ser usado para produzir documentos grosseiros em português (ou em inglês, se se trocarem as variáveis), os quais poderiam posteriormente ser corrigidos. Se o programa fosse usado num campo com um vocabulário bastante especializado, poderia realizar um trabalho bastante aceitável, se bem que não permita realizar qualquer juízo sobre as partes mais complexas de cada frase.

Talvez o leitor pense que a afirmação de que este programa poderia ser usado seriamente, com um vocabulário mais extenso, para produzir uma tradução aceitável, é perfeitamente irreal. É certo que as saídas produzidas pelo programa facilmente sugerem essa conclusão. No entanto, se pensar um pouco nas potencialidades deste programa, verificará que é possível melhorá-lo bastante.

Vejamos um exemplo das saídas produzidas pelo programa:

```
? OLA! MEU BOM AMIGO  
-> HELLO MY GOOD FRIEND  
  
? QUERO COMER UM BIFE COM BATATA  
S  
-> QUERO COMER A STEAK WITH FR  
IES  
  
? TODOS PENSAM QUE O BOM POLICIA  
E UM FANTASTICO DETECTIVE  
-> EVERYBODY PENSAM QUE THE GO  
OD POLICEMAN IS A AMAZING DETECT  
IVE
```

```

? SE VOCE VIRAR 'A DIREITA EM PA
RIS ACABA NA PLANICIE
-> SE YOU VIRAR 'A RIGHT EM PA
RIS ACABA NA PLAIN

```

Como pode verificar, apresentamos aqui ao programa certas palavras que têm em conta o vocabulário extremamente reduzido que o programa emprega:

```

? QUERO ALGUNS CIGARROS PARA POR
  ATRAS DA PORTA EM CASA
-> WANT SOME CIGARETTES TO POR
  BEHIND DA DOOR EM HOUSE

```

```

? PRECISO DO VINHO PORQUE TENHO
FRIO
-> PRECISO DO WINE PORQUE HAVE
  COLD

```

```

? QUANDO TOCAR MUSICA QUERO CANTAR
E BOENAR COM O BARCO AO SOL
-> WHEN TOCAR MUSICA WANT CANTA
  R AND BOENAR WITH THE BAR AO SUN

```

```

? PEIXE E MOSTARDA NAO DAO BEM
-> FISH AND MUSTARD NO DAD BEM

```

## Estrutura do programa

O programa é muito simples de compreender. Começa (como é habitual) por chamar uma subrotina que se encontra no final e inicializa as variáveis.

```

100 REM ** TRADUZIR **
110 FOR I=0 TO 255: REM
120 GO SUB 9999: REM inicializa
130 REM
140 FLASH 0: BRIGHT 1: PAPER 1:
150 H2K 7: BRIGHT 0: BORDER 1: CLR
160 REM ** INICIALIZAR **
170 CLR
180 DIM e$(100,15): REM palavra
190 DIM f$(100,20): REM palavra
200 DIM count=0
210 REM count=count+1
220 READ e$(count),f$(count)
230 IF f$(count,1) <> "*" THEN GO

```



```

0000 "BADA", "MUSTARD"
0050 DATA "QUENTE", "HOT", "FRIO",
"COOLD", "TODA A GENTE", "EVERYBODY"
"
0060 DATA "OLA", "HELLO", "BOM", "
GOOD"
0090 DATA "*", "*"

```

Nesta subrotina, E\$ é usada para guardar o texto original, e F\$ contém a tradução correspondente. O equivalente a E\$(4) é F\$(4), etc., o que torna extremamente fácil seguir a lógica do programa.

A variável COUNT conta o número de palavras alimentadas ao sistema. Os quadros alfanuméricos foram dimensionados de modo a poderem guardar mais palavras do que as actualmente incluídas no vocabulário do programa, o que permitirá ao leitor acrescentar algumas por sua iniciativa.

Ao voltar da rotina de inicialização, o programa aceita a entrada do utilizador (linha 30), verificando em seguida se se trata de uma cadeia vazia (isto é, se o utilizador se limitou a carregar em ENTER sem escrever qualquer texto). Se assim acontecer, isto é, se a entrada, A\$, for uma cadeia vazia, o programa termina.

```

30 INPUT a$: REM aceitar entra
40 IF a$="" THEN STOP
45 PRINT "?": a$
50 INK 6: GO SUB 1000: INK 7:
REM traduzir
60 GO TO 60

```

A linha 50, como se pode verificar, envia o texto apresentado para uma subrotina que se inicia na linha 1000. Esta subrotina é a encarregue de fazer todo o trabalho, isto é, de executar a tradução.

Podemos observar agora o «mecanismo de tradução». O Spectrum começa por imprimir uma pequena seta no início da linha, indicando o início das linhas traduzidas.

```

1000 REM ** traduzir **
1002 PRINT TAB 6; "→ ";
1005 LET l=LEN a$: LET k=0
1010 LET k=k+1

```

```

1010 IF K=1 THEN PRINT : PRINT :
      RETURN
1020 LET B$=" "+B$+" ": IF B$(K)
=" " THEN GO TO 1200
1030 GO TO 1010
1040 LET X=K+1
1050 LET Y=0
1060 LET Y=Y+1
1070 IF B$(X+Y)=" " THEN LET A$=
B$(X+Y): GO TO 1240
1080 GO TO 1020
1240 LET M=0
1090 LET M=M+1
1100 IF A$(1 TO LEN A$) <> B$(M), TO
(LEN A$) THEN GO TO 1260
1260 FOR Q=15 TO 1 STEP -1: IF Y
#(M,Q)=" " THEN NEXT Q
1270 PRINT Y#(M,1 TO Q): " ": GO
TO 1200
1280 IF #COUNT THEN GO TO 1250
1270 LET A$=A$(1 TO LEN A$-1): PR
INT FLASH 1/A$: PRINT " "
1280 GO TO 1010

```

O programa estuda o texto, procurando um espaço que indique o início de uma palavra nova (a palavra, como é óbvio, inicia-se a seguir ao espaço, sendo por essa razão que na linha 1020 acrescentamos um espaço a cada final da entrada, a fim de o programa não ignorar a primeira e a última palavra). Depois de descobrir um (linha 1020), o programa passa para a rotina que se inicia em 1200, e continua a procurar um novo espaço, que permita isolar a palavra usada. Depois consulta muito simplesmente o seu vocabulário procurando uma palavra correspondente.

```

? A MAE FOI VER O MAR COM O CAO
AZUL E COMEU UMA SOPA DE GALINHA
-> THE MOTHER FOI VER THE SEA
WITH THE DOG BLUE AND COMEU A SO
PA OF CHICKEN

```

Se encontra tal palavra, imprime-a em vez da original, voltando depois a 1010 para continuar a procura. Note que depois de ter sido encontrado um equivalente, o programa volta imediatamente a este ponto. Não perde tempo estudando o resto do vocabulário. Isto significa que as palavras que se encontram no início da lista serão traduzidas mais rapidamente

do que as outras. É por isto que as palavras vulgarmente usadas (como artigos, preposições, certas formas verbais) se devem encontrar no início desta lista.

É chegado o momento de o leitor fazer algumas experiências com este nosso programa, tentando aperceber-se das possibilidades deste e dos métodos a que poderá eventualmente recorrer para o alterar de acordo com as suas conveniências:

```

110 DIM ** TADOUNIA **
120 DIM ** 0000,000
130 DIM ** 0000: REM initialize
140
150 BRIGHT 1: PAPER 1:
160 BRIGHT 0: BORDER 1: CLR
170 INPUT #0: REM accept enter
180
190 DIM P " "
200 INPUT #0: THEN STOP
210 PRINT " "
220 INK 0: GO SUB 1000: INK 7:
230
240 GO TO 170
250 INPUT #0 = 1
260 PRINT " "
270 PRINT "*****"
280 PRINT "*****"
290 PRINT "*****"
300 PRINT "*****"
310 PRINT "*****"
320 PRINT "*****"
330 PRINT "*****"
340 PRINT "*****"
350 PRINT "*****"
360 PRINT "*****"
370 PRINT "*****"
380 PRINT "*****"
390 PRINT "*****"
400 PRINT "*****"
410 PRINT "*****"
420 PRINT "*****"
430 PRINT "*****"
440 PRINT "*****"
450 PRINT "*****"
460 PRINT "*****"
470 PRINT "*****"
480 PRINT "*****"
490 PRINT "*****"
500 PRINT "*****"
510 PRINT "*****"
520 PRINT "*****"
530 PRINT "*****"
540 PRINT "*****"
550 PRINT "*****"
560 PRINT "*****"
570 PRINT "*****"
580 PRINT "*****"
590 PRINT "*****"
600 PRINT "*****"
610 PRINT "*****"
620 PRINT "*****"
630 PRINT "*****"
640 PRINT "*****"
650 PRINT "*****"
660 PRINT "*****"
670 PRINT "*****"
680 PRINT "*****"
690 PRINT "*****"
700 PRINT "*****"
710 PRINT "*****"
720 PRINT "*****"
730 PRINT "*****"
740 PRINT "*****"
750 PRINT "*****"
760 PRINT "*****"
770 PRINT "*****"
780 PRINT "*****"
790 PRINT "*****"
800 PRINT "*****"
810 PRINT "*****"
820 PRINT "*****"
830 PRINT "*****"
840 PRINT "*****"
850 PRINT "*****"
860 PRINT "*****"
870 PRINT "*****"
880 PRINT "*****"
890 PRINT "*****"
900 PRINT "*****"
910 PRINT "*****"
920 PRINT "*****"
930 PRINT "*****"
940 PRINT "*****"
950 PRINT "*****"
960 PRINT "*****"
970 PRINT "*****"
980 PRINT "*****"
990 PRINT "*****"

```



```

000000 DATA "CASA", "HOUSE", "CADEIRA
000001", "CHAIR", "OLHO", "EYE", "SOL", "S
000002UN"
000003 DATA "CANCAO", "SONG", "AMIGO
000004", "FRIEND", "ATRAS", "BEHIND", "MAR
000005", "SEA", "MARE", "HOTHEAR"
000006 DATA "GATO", "CAT", "CAO", "DO
000007", "ANUL", "BLUE", "PAQUENO", "LITT
000008LE"
000009 DATA "MUSICA", "MUSIC", "QUER
000010", "WANT", "BAPAZ", "BOY", "RAPARIG
000011", "GIRL"
000012 DATA "PEIXE", "FISH", "GALINH
000013", "CHICKEN", "PATO", "DUCK", "MOST
000014", "MUSTARD"
000015 DATA "QUENTE", "HOT", "FRIO",
000016", "TOO HOT", "EVERYBODY"
000017 DATA "OLA", "HELLO", "BOM", "
000018"
000019 DATA "*", "*"

```

## HANSHAN

O último programa desta parte sobre linguagem permite criar aleatoriamente poemas. Trata-se de um programa bastante trivial, e que talvez lhe pareça não ser muito sugestivo em termos de inteligência artificial.

No entanto, imagine que estava a ler um livro como este há trinta anos. Suponha que o autor se referia a um aparelho de baixo custo capaz de escrever poesia automaticamente. Há trinta anos este facto pareceria extraordinário. E de facto, quando se pensa no assunto, ainda hoje o é. Estamos de tal forma habituados a milagres que já não lhes damos importância.

Tendo portanto em conta estas observações, voltamos a HANSHAN a fim de criarmos alguns poemas. O nome do programa recorda o poeta chinês HAN-SHAN, que viveu nos séculos VIII e IX da nossa era. Depois de ter vagabundeado durante muitos anos, instalou-se na Montanha Fria (Han-Shan), por cujo nome é conhecido.

Todas as frases usadas no programa foram obtidas no livro *Chinese Poems* (Arthur Waley, Unwin Paperbacks, Londres, 1982):

```

360 DIM ***** dados *****
370 DIM * "pelelavivas", "soledades", *
380 DATA "PARROCIPITANOD", "ARRAST
ANDO", "LENTAMENTE", "ATROFIADO", "
CINZELADO"
390 DATA "ESCONSO", "RETORCIDO",
"FLANQUEADO", "DOBRADO", "CONTORCI
DO"
400 DATA "BATIDO", "PENDURADO", "
SINUOSO", "SINUOSO", "TRANSPARENTE

```



```

1000 DEFM ** P@A@T@O U@ **
1100 DEFM INT @#(INT (RND*20)+1);".
1110 ##(INT (RND*20)+1)
1120 DEFM INT TAB 0;"....";@#(INT (
RND*20)+1)
1130 DEFM INT TAB 0;@#(INT (RND*20)
+1)
1140 RETURN
1150 DEFM ** P@A@T@O A@I@S **
1160 DEFM INT @#(INT (RND*20)+1)
1170 DEFM INT TAB 0;@#(INT (RND*20)
+1);".
1180 DEFM INT TAB 0;@#(INT (RND*20)
+1)
1190 RETURN
1200 DEFM ** P@A@T@O T@S **
1210 DEFM INT TAB 0;@#(INT (RND*20)
+1)
1220 PRINT @#(INT (RND*20)+1)
1230 PRINT TAB 0;@#(INT (RND*20)
+1);". " @#(INT (RND*20)+1)
1240 RETURN

```

Alguns dos poemas produzidos por HANSHAN conseguem uma qualidade surpreendente:

AFASTO A LANTERNA  
 NA CORRENTE FRIA...  
 APRESENTO O PASSO

QUANDO NOS ENCONTRAREMOS  
 POR NESTE POUCOS TEMPOS  
 NA ESCURIDÃO TEMPOS

SOMBRIO, SOMBRIO  
 ...CONTORNO DO  
 OS QUE SÃO DEIXADOS

PORQUE OS DEUS ESCOLHER  
 ONDE OS DEUS ESCOLHER...  
 ESCOLHER DE MADRUGADA

HOMENS DE SABER  
 HOMENS DE SABER  
 E MUITO SOFRIMENTO

Apresentamos em seguida a listagem de HANSHAN, que permitirá ao leitor criar uma sequência praticamente infinita





## IV PARTE AUXILIARES

### 11

#### SISTEMAS DEDICADOS

Existe no mundo um número limitado de peritos em qualquer tema. Não interessa qual é esse tema – mineração de urânio, concepção de veículos automóveis, diagnóstico de doenças humanas, separação de cogumelos saudáveis dos venenosos – é sempre limitado o número de peritos existente.

Se bem que o mundo talvez não necessite de mais especialistas na escolha de cogumelos, existem regiões do mundo (a maior parte delas aliás) onde não existem médicos suficientes. Um dos conceitos básicos dos sistemas dedicados consiste precisamente em «recolher» a perícia desses especialistas num computador, de tal modo que qualquer pessoa possa recorrer a ela.

Os sistemas dedicados constituem uma área da investigação em inteligência artificial em que já foram dados bastantes passos em frente. É uma área em que já se dispõe de sistemas capazes de darem contributos genuínos, economicamente viáveis. E é uma área da inteligência artificial em que pouco interessam as discussões sobre o «pensamento» ou ausência deste nas máquinas.

Na sua forma mais simples, um sistema dedicado é formado por uma série de declarações IF/THEN. Um sistema de diagnóstico pode ser tão simples como o que se segue:

SE o paciente tosse

E se esteve recentemente sujeito a uma boa chuvada

E depois ficou sujeito a um vento gelado durante uma hora

ENTÃO o doente sofre de constipação ou pneumonia.

Como é óbvio, dificilmente seria necessário um sistema dedicado para fazer um diagnóstico como este. Estes sistemas

são úteis quando ocorre alguma das seguintes condições:

- não se encontra presente o especialista, mas é emulada a sua perícia no campo considerado;
- mesmo o especialista não conhece com 100% de certeza as ligações casuais entre as observações e os resultados. Isto pode acontecer se um investigador no campo médico tem consciência de que os doentes que contraíram a doença x tenderam a ter contacto com os alimentos A e B e têm o grupo sanguíneo C... se bem que não tenha sido descoberta qualquer relação entre A, B e C para além de se manifestarem em conjunto no caso dessa doença. Neste caso, um sistema dedicado convenientemente programado pode realizar previsões sobre a probabilidade de o indivíduo D contrair a doença, mesmo quando a percentagem da contribuição dos factores A, B e C é desconhecida. Estudando um número suficiente de casos, o sistema dedicado pode não só definir regras próprias de previsão da ocorrência da doença, como ainda explicar o seu raciocínio a um médico humano.

Na parte do livro sobre aprendizagem e raciocínio, falámos de circuitos lógicos e discutimos o modo como estes tomam decisões, de acordo com as regras da álgebra booleana.

As «matemáticas do raciocínio» são muito importantes para a construção de sistemas dedicados. Muitas vezes, a tentativa de definir os conhecimentos de um ser humano a fim de os codificar numa base de dados de um sistema dedicado (e mais adiante observaremos alguns dos sistemas que actualmente estão a ser utilizados) conduz apenas à descoberta de que o especialista não sabe como atinge as decisões.

Esta revelação é tão inesperada para o especialista como para a pessoa que cria a base de conhecimentos necessária ao programa. Em *The Fifth Generation – Artificial Intelligence and Japan's Computer Challenge to the World* (Reading, Massachusetts: Feigenbaum, Edward A. e McCorduck, Pamela, 1983: págs. 85 e 86) podemos ler a história bastante triste de um especialista que explicou os seus métodos a um «engenheiro de conhecimento». O especialista era bastante considerado e bem pago, e começou por não acreditar quando o técnico de

informática descobriu que todos os seus conhecimentos podiam ser reduzidos a algumas centenas de regras simples. O especialista acabou por ficar deprimido e abandonar a sua profissão.

As máquinas tomam decisões baseando-se em regras internas. Estas – como já vimos na discussão sobre os programas que aprendem e raciocinam – são relativamente simples. O raciocínio lógico elementar reduz-se a algumas regras fáceis de exprimir e em pequena quantidade.

Vimos que os silogismos podem ser expressos, e resolvidos, pela máquina, porque assumem a seguinte forma:

A é um C  
C é um B

Portanto, A é um B

A esperança de reduzir o raciocínio a um processo mecânico acompanha-nos desde há muito. Já em 1677, no prefácio à obra *The General Sciences*, Gottfried Leibniz escreveu:

«Se pudéssemos descobrir características ou sinais apropriados para exprimir todos os nossos pensamentos de uma forma tão clara e exacta como a aritmética exprime os números ou a geometria analítica exprime as linhas, poderíamos em todos os temas, na medida em que possam ser reduzidos ao raciocínio, conseguir aquilo que é feito em matemática e geometria...

Mais ainda, poderíamos convencer o mundo do que descobríssemos ou calculássemos, dado que seria fácil verificar o cálculo... se alguém duvidasse dos resultados, dir-lhe-ia: “Calculemos, senhor”, e pegando numa caneta e em tinta poderíamos resolver a questão.»

Em vez de pegar numa caneta e em tinta, podemos pegar em silício, e encontrar pelo menos respostas para algumas questões que ainda nos ultrapassam (como a capacidade para prever a estrutura química de um composto ainda não desenvolvido, como faz um sistema dedicado) e indicar as soluções para problemas que nenhum ser vivo pode resolver.

## **Limitações**

A menos que estejam especificamente programados para alertarem um operador, os sistemas dedicados podem tornar-se bastante estúpidos quando encontram qualquer coisa que não caiba exactamente no reportório que contêm. É como alguém brilhante em xadrez, mas incapaz de fazer um nó de gravata. Muitos sistemas dedicados baseados apenas na interpretação de regras do tipo IF/THEN têm caracteristicamente um estatuto de sábios idiotas.

Estes sistemas não têm capacidade para aumentar a sua base de conhecimento ao funcionarem, e apenas podem pensar em linha recta entre A e B, deste para C, etc. Estes sistemas podem não ter possibilidades de saberem quando o seu conhecimento laboriosamente programado é insuficiente, nenhum modo de reconhecerem uma excepção à regra.

O sistema que desenvolveremos entra nesta categoria do sábio idiota. Mas apesar desta limitação, que se aplica à maioria dos sistemas dedicados actualmente em uso, o leitor verificará que os sistemas por si desenvolvidos são de facto fascinantes. O nosso sistema final, como verá, tem de facto a capacidade de aprender. Com efeito, basta-nos dizer-lhe – quando tenta distinguir entre qualquer número de coisas que nele tenham sido programadas – se a sua resposta está correcta ou não, para acabar por aprender a distinguir os objectos, sem que lhe seja dito explicitamente como deve fazer uma distinção entre eles.

## **A estrutura química e dendral**

Antes de entrarmos nos nossos sistemas dedicados, observaremos alguns dos actualmente usados, e veremos o que será possível aprender com eles.

O primeiro programa que observaremos, e talvez o primeiro sistema dedicado que de facto funcionou no mundo, é chamado DENDRAL. O desenvolvimento deste sistema – que é capaz de descobrir factos acerca de estruturas moleculares a partir de dados químicos grosseiros – iniciou-se na Stanford University

em 1965. Juntando conhecimentos de várias disciplinas, os criadores de DENDRAL produziram um sistema que actualmente dá melhores resultados do que qualquer pessoa neste campo (incluindo os homens que o construíram). DENDRAL é hoje usado em todo o mundo.

Foi também em Stanford que se desenvolveu MYCIN, um sistema que diagnostica infecções sanguíneas e meningite, sugerindo depois tratamentos possíveis. MYCIN baseia as suas conclusões em dados indicados por um médico, e pode – se lhe for pedido – explicar como constrói o diagnóstico. O sistema contém cerca de 450 regras.

A base de conhecimento em MYCIN é tão valiosa que foi desenvolvido um programa acompanhante – GUIDON – para permitir ao computador actuar como professor, construindo assim uma ponte entre o especialista humano (ou vários, neste caso) e um outro ser humano que procura obter esse conhecimento.

Não termina ainda aqui o valor de MYCIN. Grande parte do programa consiste em modos de manipular as regras que recebeu, e em tirar conclusões delas. Os mecanismos de manipulação e inferência são – em grande medida – independentes da base de conhecimento. Isto sugere que a informação relacionada com as infecções sanguíneas poderia ser eliminada, acrescentando-se em seu lugar novas informações. Isto já foi feito, tendo sido desenvolvido o sistema dedicado PUFF que fornece actualmente uma assistência semelhante à dada por MYCIN, mas relativamente a desordens pulmonares.

Este processo tornou-se tão eficaz (e numa experiência feita sobre 150 casos, PUFF produziu os mesmos diagnósticos que os especialistas humanos) que foi ainda desenvolvida uma outra versão do MYCIN, chamada simplesmente EMYCIN, que admite outras bases de conhecimento.

O sistema dedicado MOLGEN (de GENética MOlecular) auxilia os biólogos que trabalham com o DNA e em engenharia genética. É também muito usado.

O aspecto mais interessante – em termos de examinar as direcções que a investigação em inteligência artificial actualmente segue – é que os sistemas dedicados trabalham extremamente bem, sendo lógico, em termos económicos, usá-los. Isto garante

o seu uso, e o desenvolvimento de mais sistemas deste tipo. A linha da investigação «pura» produz obviamente resultados, mas estes tendem a vir mais rapidamente quando se desenvolvem em função de necessidades práticas imediatas.

Pensemos num sistema que aconselhasse sobre o local onde conviria fazer uma perfuração para encontrar petróleo. Uma única descoberta permitiria reembolsar todo o custo de desenvolvimento do sistema, mesmo que custasse milhões: bastariam talvez alguns dias para recuperar esse custo.

Feigenbaum e McCorduck (em *The Fifth Generation*, já mencionado, págs. 72, 73) apresenta-nos um exemplo gráfico do poder de recuperação do capital dos principais sistemas dedicados. Citam o caso de uma grande empresa americana que recentemente adquiriu um sistema dedicado concebido para diagnosticar falhas em certos tipos de centrais eléctricas. Ao ensaiar uma versão inicial, bastante incompleta, do programa usando dados reais sobre a paragem de uma das centrais em 1981, verificou-se que o sistema descobriu a causa do problema em alguns segundos. Os especialistas que trabalharam na central, na época, tinham necessitado de dias para chegarem à mesma conclusão. Entretanto a central estivera encerrada durante quatro dias, o que custara à empresa cerca de 1,2 milhões de dólares.

Neste momento estão a ser usados ou desenvolvidos muitos outros sistemas no mundo. Entre eles citamos:

– PROGRAMMER'S APPRENTICE: Um sistema que auxilia, como o próprio nome sugere, ao desenvolvimento de software.

– EURISKO: Um sistema dedicado que é capaz de aprender em funcionamento, e concebido para concepção de circuitos microeléctricos tridimensionais.

– TAXMAN: Actualmente a ser desenvolvido na Rutgers University, este sistema é concebido para examinar a alteração das regras fiscais, deduzindo daí conselhos a dar a empresas quanto à melhor forma de trabalharem dentro dessas regras.

**- GENESIS: Um programa com um nome sonante... Este sistema, que já foi comercializado, permite aos cientistas planearem e simularem experiências com genes.**

**Receio que o nosso programa não seja tão desenvolvido como os que acabámos de citar, se bem que permita já ao leitor descobrir algumas aplicações bastante interessantes – por exemplo, diferenciar entre um homem, um cavalo e um pardal... Passemos então ao estudo do primeiro dos nossos sistemas.**

## DIFERENÇA

O nosso primeiro sistema dedicado é DIFERENÇA. Este programa é capaz de descobrir, sem qualquer erro, a diferença entre três criaturas vivas – um homem, um cavalo e um parda. Se bem que se trate de uma situação pouco «inteligente», e que provavelmente não ocorre muitas vezes no nosso quotidiano, pode ensinar-nos bastante sobre a forma como alguns tipos de sistemas dedicados podem ser desenvolvidos.

Imaginemos um sistema de diagnóstico médico. Chamaremos ao nosso sistema imaginário MEDICI. MEDICI e DIFERENÇA são programas muito próximos, como o leitor compreenderá, e o estudo de DIFERENÇA dar-lhe-á uma base a partir da qual poderá desenvolver um certo grau de conhecimento de MEDICI ou de qualquer outro sistema dedicado de mais vastas proporções.

O leitor vai ter uma sessão com MEDICI. O sistema faz-lhe bastantes perguntas, às quais o leitor deve responder SIM ou NÃO do seguinte modo:

```

E' DO SEXO MASCULINO?
TEM MAIS DE 40 ANOS DE IDADE?
FUMA?
FEZ UM EXAME GERAL NOS ULTIMOS
DOZE MESES?
PREOCUPA-SE FREQUENTEMENTE?
CONSIDERA-SE UMA PESSOA NERVOSA?
  
```

Etc. Depois de apresentar estas perguntas, MEDICI faz uma pequena pausa e imprime a seguinte mensagem:

OBRIGADO, A SUA EXPECTATIVA DE VIDA É 79 ANOS, EXCEDENDO ASSIM 11% DA POPULAÇÃO. PARA AUMENTAR AS SUAS POSSIBILIDADES DE ATINGIR ESTA IDADE OU EXCEDI-LA, SUGIRO:

- TENHA DEIXAR DE FUMAR
- FAÇA EXAMES MÉDICOS REGULARES
- AUMENTE A QUANTIDADE DE EXERCÍCIO SEMANAL

OBRIGADO POR TER CONSULTADO  
MÉDICO

Que fez MEDICI? Como transformou as suas respostas numa expectativa de vida? O leitor terá já compreendido que este programa não é muito sofisticado, e que exige um nível elevado de conhecimento. No entanto, mostra como poderia começar um programa de diagnóstico médico, no caso de o sistema dedicado interagir directamente com o doente, em vez de o fazer com o médico como acontece geralmente.

Satisfeito por viver mais do que 11% da população, o leitor prepara-se para conhecer um outro sistema, o jovem DIFERENÇA. Vejamos então o que pode observar no visor:

PENSA EM UM HOMEM, NUM CAVALO OU  
NUM PARDAL

TEM DUAS PERNAS?

S OU N? S

PODE ANDAR?

S OU N? S

PODE VOAR?

S OU N? N

ESTAVA A PENSAR NUM HOMEM

CARREGUE 'ENTER' PARA CONTINUAR  
OU QUALQUER TECLA E 'ENTER' PARA  
DESISTIR.

**DIFERENÇA** está, evidentemente, certo. Não é difícil determinar a partir das suas respostas que o leitor estava a pensar num homem. Bastante impressionado, o leitor carrega na tecla ENTER, e executa novamente o programa:

PENSE NUM HOMEM, NUM CAVALO OU  
NUM PARDAL

TEM DUAS PERNAS?

S OU N? S

PODE ANDAR?

S OU N? S

PODE VOAR?

S OU N? S

ESTAVA A PENSAR NUM PARDAL

O leitor pensa que o programa não pode acertar de novo, e passa a um terceiro caso:

PENSE NUM HOMEM, NUM CAVALO OU  
NUM PARDAL

TEM DUAS PERNAS?

S OU N? N

PODE ANDAR?

S OU N? S

PODE VOAR?

S OU N? N

ESTAVA A PENSAR NUM CAVALO

Desta vez o leitor decide desistir. Como consegue DIFERENÇA registrar as respostas às suas perguntas de modo a descobrir que a criatura com duas pernas, capaz de andar, mas incapaz de voar, é um homem? Aliás, como pode MEDICI descobrir que você viverá até aos 79 anos?

É muito simples, pelo menos no caso de DIFERENÇA (e MEDICI funciona da mesma forma geral, apenas sendo bastante mais sofisticado). DIFERENÇA contou cada vez que o leitor deu a resposta SIM a uma pergunta. Se apenas disse SIM uma vez, estará a pensar num cavalo (dado que a pergunta «Pode andar» é a única a que pode responder afirmativamente no caso de um cavalo). Duas respostas afirmativas, trata-se necessariamente de um homem. Se forem três, DIFERENÇA sabe que se trata de um pardal.

MEDICI conta não só as respostas, mas também a pergunta a que se referem. Um SIM a FUMA? Tirar três anos à sua expectativa de vida, enquanto um SIM a FAZ EXERCÍCIO REGULARMENTE aumenta cinco anos a essa mesma perspectiva de vida.

A listagem de DIFERENÇA inicia-se do seguinte modo:

```
10 REM Diferencas
15 PAPER 1: INK 7: BRIGHT 0: F
LASH 0: BORDER 1
20 POKE 23856,200: OLS
30 PRINT "PENSE NUM HOMEM, NUM
CAVALO OU NUM PARDAL"
50 PAUSE 400
60 PRINT "
70 GO SUB 170: REM FAZER PERGU
NTAS
```

Depois de ter definido o ambiente, inicia-se a determinação da criatura considerada:

```
170 REM ** fazer perguntas **
180 LET count=0
190 PRINT INK 5: "TEM DUAS PERNA
30"
200 GO SUB 310
210 PRINT INK 5: "PODE ANDAR?"
220 GO SUB 310
```

```

0030 PRINT INK 6;"PODE VOAR?"
0040 GOTO SUB 010
0050 REM ***#**#**#**#**#**#**#**#**#**#
010 REM ** DOCE#SAD# DE#90810
**
000 INPUT " S OU N? ";Z$
030 IF Z#<>"N" AND Z#<>"S" THEN
00 TO 000
040 IF Z#="S" THEN LET count=co
unt+1
050 PRINT
060 RETURN

```

O leitor pode verificar que a variável COUNT é colocada a zero no início da execução e incrementada de cada vez que é recebida uma resposta SIM. Usando esta informação, DIFERENÇA não tem dificuldade em determinar qual a criatura em que o utilizador está a pensar:

```

250 PRINT "ESTAVA A PENSAR NUM
?"
050 IF COUNT=1 THEN PRINT FLASH
10 INK 4;"CAVALO"
070 IF COUNT=2 THEN PRINT FLASH
10 INK 4;"HOMEM"
090 IF COUNT=3 THEN PRINT FLASH
10 INK 4;"PARDAL"
000 RETURN

```

Como se pode verificar, trata-se de um programa bastante simples, mas que apresenta os fundamentos dos sistemas dedicados. Vejamos a listagem completa:

```

10 REM Diferenças
15 PAPER 1: INK 7: BRIGHT 0: F
LASH 0: BORDER 1
20 POKE 30690,200: CLS
30 PRINT "PENSE NUM HOMEM, NUM
CAVALO OU NUM PARDAL"
50 PUSE 400
60 PRINT "
70 GOTO SUB 170: REM FAZER PERGU
NTAS
80 PRINT " ) INK 2) "-----
-----"
100 PRINT "CARREGUE 'ENTER' PA
RA CONTINUAR OU DOUBLOUVER TECLA E
'ENTER' PARA QUESISTIR."
110 INPUT A#

```

```

120 IF A#<>" THEN STOP
140 GOTO 5
150 GOTO 30
160 REM *****
170 REM ** FAZER PERGUNTAS **
180 LET COUNT=0
190 PRINT INK 5;"TEM DUAS PERNA
200 GO SUB 310
210 PRINT INK 5;"PODE ANDAR?"
220 GO SUB 310
230 PRINT INK 5;"PODE VOAR?"
240 GO SUB 310
250 PRINT "ESTAVA A PENSAR NUM
260 IF COUNT=1 THEN PRINT FLASH
1) INK 4;"CAVALO"
270 IF COUNT=2 THEN PRINT FLASH
1) INK 4;"HOMEM"
280 IF COUNT=3 THEN PRINT FLASH
1) INK 4;"PARDAL"
290 RETURN
300 REM *****
310 REM ** PROCCESSAR RESPOSTAS
**
320 INPUT " S OU N? ";Z#
330 IF Z#<>"N" AND Z#<>"S" THEN
GO TO 320
340 IF Z#="S" THEN LET COUNT=CO
UNT+1
350 PRINT
360 RETURN

```

## Diferença X

Este programa é o irmão mais velho do anterior. Se bem que esteja directamente relacionado com o programa que acabamos de estudar, este é bastante mais sofisticado.

Esta maior sofisticação pode ser observada num simples exemplo de execução. Primeiramente, tentaremos obter resultados semelhantes aos dados por DIFERENÇA. No entanto, podemos já, desde início, constatar que se trata de um programa bastante diferente. Em particular, verifica-se que os seus conhecimentos são comunicados separadamente para cada execução.

```
Nome do sistema? CRIATURAS
Numero de resultados? 3
Numero de factores a considerar?
3
```

O leitor indicará ao programa o seu tema (neste caso, CRIATURAS), e depois o número de «resultados» e de «factores» a considerar. São estas as variáveis (por exemplo PODE VOAR) a ter em conta. Tendo definido este ambiente, DIFERENÇA-X faz em seguida as perguntas apropriadas:

```
CRIATURAS
Qual é o resultado 1? HOMEM
Qual é o resultado 2? CAVALO
Qual é o resultado 3? PARDAL
```

Tendo conhecido os resultados, pede ao utilizador que indique as perguntas relacionadas com os factores a considerar para determinação do resultado:

```
Por favor indique a pergunta 1?
VOA PELOS SEUS MEIOS
Por favor indique a pergunta 2?
POSSUI DUAS PERNAS
Por favor indique a pergunta 3?
ANDA
```

Isto pode parecer um pouco trabalhoso, mas – como o leitor compreenderá dentro em pouco – valerá a pena. Este simples exercício de cópia do programa anterior mostra claramente como é possível treinar DIFERENÇA-X de modo a transformar-se em especialista de praticamente qualquer campo.

DIFERENÇA-X percorre agora cada um dos resultados já indicados, e pergunta: «Se eu apresentasse a pergunta seguinte, relativamente a este resultado, que responderia?» A partir das informações assim recolhidas, DIFERENÇA-X pode formar uma base de conhecimentos equivalente à definida em DIFERENÇA. Claro que DIFERENÇA-X pode construir uma base de conhecimento sobre praticamente qualquer assunto.

Por favor responda ao seguinte:  
Para um resultado de > HOMEM <  
Por favor escreva "S" ou "N"

> VOA PELOS SEUS MEIOS? N

> POSSUI DUAS PERNAS? S

> ANDA? S

Por favor responda ao seguinte:  
Para um resultado de > CAVALO <  
Por favor escreva "S" ou "N"

> VOA PELOS SEUS MEIOS? N

> POSSUI DUAS PERNAS? N

> ANDA? S

Por favor responda ao seguinte:  
Para um resultado de > PARDAL <  
Por favor escreva "S" ou "N"

> VOA PELOS SEUS MEIOS? S

> POSSUI DUAS PERNAS? S

> ANDA? S

Depois de terem sido considerados todos os resultados possíveis, e de o computador aprender as respostas apropriadas em cada caso, DIFERENÇA-X constrói uma «base de conhecimento» que neste caso corresponde a pouco mais do que a simples soma do total de respostas SIM. O programa comunica então as suas descobertas ao utilizador:

E' esta a minha base de conheci-  
mento:

HOMEM --- 6

CAVALO --- 4

PARDAL --- 7

Por favor escreva "S" ou "N"

VDA PELOS SEUS MEIOS?

N

POSSUI DUAS PERNAS?

S

ANDA?

S

```

>O meu resultado era 6
> Estava a pensar em
  HOMEM
```

Mas onde obtive estes números? O leitor não poderia ter dado quatro respostas no caso do cavalo, ou sete no caso do pardal, dado que apenas foram feitas três perguntas. Acontece no entanto que o programa não soma um único SIM por cada resposta, usando em vez disso em cada caso um número que varia para cada resposta. Se fosse atribuído apenas um factor unitário a cada SIM e tivéssemos respondido deste modo, por exemplo, às perguntas um e três num dos casos, e às perguntas dois e três no outro, obteríamos o mesmo total para ambos os objectos.

Para resolver este problema, concebemos um que permite definir a ordem pela qual foram dadas as respostas:

Uma resposta afirmativa à pergunta 1 vale 1

Uma resposta afirmativa à pergunta 2 vale 2

Uma resposta afirmativa à pergunta 3 vale 4

Uma resposta afirmativa à pergunta 4 vale 8  
Uma resposta afirmativa à pergunta 5 vale 16  
Uma resposta afirmativa à pergunta 6 vale 32  
Uma resposta afirmativa à pergunta 7 vale 64  
Etc.

Isto garante que, mesmo no caso de ser dado o mesmo número de respostas afirmativas, será obtido um número identificador diferente para cada caso.

Será que este método funciona? Claro que sim, e vejamos um exemplo:

VOA PELOS SEUS MEIOS?

N

POSSUI DUAS PERNAS?

S

ANDA?

S

> O meu resultado era 6

> Estava a pensar em  
HOMEM

Já disse anteriormente que DIFERENÇA-X é capaz de fazer muito mais do que DIFERENÇA, e agora vou demonstrar a verdade desta afirmação. Vamos treinar o nosso sistema num campo completamente diferente, no qual não dispõe de quaisquer conhecimentos.

A base de conhecimentos alimentada ao programa veio de um livro, escrito por um especialista chamado Oliver Chambers (*The Observer's Book of Rocks and Minerals*, Nova Iorque, Frederick Warne, 1979). Com o auxílio dos conhecimentos de Mr. Chambers, DIFERENÇA-X vai poder adquirir dados suficientes para distinguir cinco tipos de substâncias minerais, usando quatro factores.

Ser-me-ia impossível obter este resultado sem recorrer ao auxílio deste programa. É isto aliás que acontece à maior parte das pessoas que actualmente utilizam sistemas dedicados. Um destes sistemas codifica de facto os conhecimentos de um especialista, de tal modo que aqueles que não o são, possam mesmo assim utilizar esses conhecimentos.

Começemos por introduzir parte dos conhecimentos de Mr. Chambers no nosso sistema (começando de novo, a fim de os minerais não se verem confundidos com cavalos e pardais...).

Definimos primeiramente o tema:

Nome do sistema? IDENTIFICACAO M  
INERAL

Numero de resultados? 5

Numero de factores a considerar?  
4

Em seguida indicamos as quatro substâncias que desejamos identificar:

Qual é o resultado 1? PLEONASTO

Qual é o resultado 2? LIMONITE

Qual é o resultado 3? IODARGIRI  
TA

Qual é o resultado 4? IRIDOSMIN  
A

Qual é o resultado 5? SILVANITA

Depois, indicamos ao programa quais as perguntas a fazer para discriminar os minerais:

Por favor indique a pergunta 1?  
E' DURO

Por favor indique a pergunta 2?  
CONTEM FAIXAS DE DIFERENTE COR

Por favor indique a pergunta 3?  
TEM UM PESO ESPECIFICO SUPERIOR  
A 5

Por favor indique a pergunta 4?  
E' FUSIVEL

**Em seguida iniciamos o longo trabalho de codificação destes conhecimentos:**

Por favor responda ao seguinte:  
Para UM resultado de > PLEONASTO

Por favor escreva "S" ou "N"

> E' DURO? S

> CONTEM FAIXAS DE COR DIFERENTE  
? N

> TEM UM PESO ESPECIFICO SUPERIOR  
A A 5? N

> E' FUSIVEL? N

**O programa actua do mesmo modo para o resto dos minerais. Finalmente, relata-nos as suas descobertas:**

PLEONASTO --- 1

LIMONITE --- 6

IODARGIRITA --- 12

IRIDOSMINA --- 7

SILVANITA --- 13

## Vejamos os resultados de tudo isto:

```
Por favor pense num dos temas:
PLEONASTO
LIMONITA
IODARGIRITA
IRIDOSMINA
OU SILVANITA
```

Por favor escreva "S" ou "N"

E' DURO?  
N

CONTEM FAIXAS DE COR DIFERENTE?  
N

TEM UM PESO ESPECIFICO SUPERIOR  
A 50?  
N

EM FUSIVEL?  
S

> O meu resultado era S

TE > Estava a pensar em LIMONI

Em apenas alguns minutos, DIFERENÇA-X adquiriu conhecimentos que me permitem, nada percebendo neste campo, usar conhecimentos de especialista numa situação prática.

Vamos observar em seguida a construção do programa a fim de conhecermos o modo de construir um programa deste tipo.

Primeiramente observamos um ciclo principal, que chama diversas subrotinas:

```
10 REM Diferencas-X
15 POKE 23553,200: INVERSE 0:
FLASH 0: BRIGHT 1: PAPER 1: INK
```

```

7: BRIGHT 0: BORDER 1: OLS
200 GO SUB 940: REM inicializar
300 GO SUB 450: REM encher peric
ci:
400 GO SUB 120: REM demonstrar
peric:
500 GO SUB 1050
600 PRINT "CARREGUE 'ENTER' PA
RA CONTINUAR OU QUALQUER TECLA E
'ENTER' PARA RESISTIR."
800 INPUT "A#="
900 IF A#="" THEN GO TO 40
1000 STOP

```

Na rotina de inicialização, o programa adquire um nome que identifica o sistema (o que é útil a fim de poder gravar em fita toda a base de conhecimento do programa, sob a forma de um ficheiro). São dimensionados alguns quadros ou matrizes que guardarão os nomes e os totais correspondentes aos diversos resultados, além das perguntas relacionadas com os factores usados:

```

940 REM ** inicializacao **
950 OLS
960 INPUT "Nome do sistema? " ; n
#
970 GO SUB 1050
980 INPUT "Numero de resultados
? " ; outcomes
990 GO SUB 1050
1000 INPUT "Numero de factores a
considerar? " ; factors
1010 DIM a#(outcomes,15): DIM b#
(factors,30)
1020 DIM d(outcomes)
1030 OLS
1040 RETURN

```

A secção de código que se segue obtém os nomes dos resultados:

```

450 REM *** encher quadros ***
460 PRINT TAB (20-LEN n#/2) ; n#
470 GO SUB 1050
480 REM * obter nomes de result
#
490 *
490 FOR J=1 TO outcomes
500 GO SUB 1050

```

```

510 PRINT "Qual e' o resultado
(70)"
520 INPUT a$(J): PRINT INK 4; a$
(8)
530 NEXT J

```

Depois o programa pede os factores, as perguntas a fazer:

```

550 REM ** obter pergunta a faz
em **
560 FOR J=1 TO factors
570 GO SUB 1050
580 PRINT "Por favor indique pe
rgunta "J
590 INPUT b$(J)
600 NEXT J

```

Tudo isto são apenas preparativos. DIFERENÇA-X deseja agora uma informação mais substancial, e portanto verifica todos os resultados (usando o ciclo J, entre 630 e 810), factor a factor (com o ciclo K, 720 a 800):

```

630 REM ** adquirir pericia **
640 FOR J=1 TO outcomes
650 CLS
660 GO SUB 1050
670 PRINT "Por favor responda a
o seguinte: " "Para um resultado
de ">"a$(J); "
680 GO SUB 1050
690 PRINT "Rescreva ""S"" ou ""N""
"
700 LET x=.5
710 FOR k=1 TO factors
720 LET x=x+x
730 GO SUB 1050
740 PRINT TAB 4; "> "; b$(K);
750 LET multi=0
760 INPUT y$
770 IF y$="S" THEN LET multi=1
780 LET d(J)=d(J)+x*multi: REM
computar base de pericia
790 NEXT k
810 NEXT J

```

Tendo feito isto, o programa mostra-nos o que aprendeu:

```

830 PRINT "E esta a minha base
de conhecimento:"

```

```

0040 FOR J=1 TO outcomes
0050 GO SUB 1000
0060 PRINT a#(J); " ---"; a(J)
0070 NEXT J
0080 GO SUB 1000
0090 PRINT TAB 0; "Carregue em ""
ENTER ""
0100 INPUT a#
0110 GOSUB 1000
0120 RETURN

```

Tendo esta informação guardada em segurança, o programa está pronto a funcionar de uma forma semelhante à do programa anterior, fazendo perguntas, somando números, e tomando uma decisão a partir dos totais obtidos:

```

100 REM demonstrar pericia
110 GOSUB 1000
120 PRINT INK 0; INVERSE 1; " Por
130 favor pense num dos temas - "
140 FOR J=1 TO outcomes
150 PRINT TAB J+2
160 IF J=outcomes THEN PRINT "O
170 PRINT a#(J)
180 NEXT J
190 GO SUB 1000
200 LET result=0
210 LET x=.5
220 PRINT "Por favor escreve ""
230 INPUT a#
240 FOR J=1 TO factors
250 LET x=x+x
260 GO SUB 1000
270 PRINT a#(J)
280 INPUT e#
290 IF e#"N" THEN LET result=r
300 result+x
310 NEXT J
320 PRINT TAB 0; INK 0; "> O meu
330 resultado era " result
340 GO SUB 1000
350 LET m=0
360 LET m=m+1
370 IF a(m)=result THEN GO TO 4
380
390 IF m<outcomes THEN GO TO 35
400 PRINT TAB 2; FLASH 1; INK 0

```



```

1470 PRINT TAB J+2
1500 PRINT J="outcomes THEN PRINT "O
C
1600 PRINT "R#(J)
1700 GOTO 1800
1800 PRINT "R#(J)
1900 PRINT "R#(J)
2000 PRINT "R#(J)
2100 PRINT "R#(J)
2200 PRINT "R#(J)
2300 PRINT "R#(J)
2400 PRINT "R#(J)
2500 PRINT "R#(J)
2600 PRINT "R#(J)
2700 PRINT "R#(J)
2800 PRINT "R#(J)
2900 PRINT "R#(J)
3000 PRINT "R#(J)
3100 PRINT "R#(J)
3200 PRINT "R#(J)
3300 PRINT "R#(J)
3400 PRINT "R#(J)
3500 PRINT "R#(J)
3600 PRINT "R#(J)
3700 PRINT "R#(J)
3800 PRINT "R#(J)
3900 PRINT "R#(J)
4000 PRINT "R#(J)
4100 PRINT "R#(J)
4200 PRINT "R#(J)
4300 PRINT "R#(J)
4400 PRINT "R#(J)
4500 PRINT "R#(J)
4600 PRINT "R#(J)
4700 PRINT "R#(J)
4800 PRINT "R#(J)
4900 PRINT "R#(J)
5000 PRINT "R#(J)
5100 PRINT "R#(J)
5200 PRINT "R#(J)
5300 PRINT "R#(J)
5400 PRINT "R#(J)
5500 PRINT "R#(J)
5600 PRINT "R#(J)
5700 PRINT "R#(J)
5800 PRINT "R#(J)
5900 PRINT "R#(J)
6000 PRINT "R#(J)
6100 PRINT "R#(J)
6200 PRINT "R#(J)
6300 PRINT "R#(J)
6400 PRINT "R#(J)
6500 PRINT "R#(J)
6600 PRINT "R#(J)
6700 PRINT "R#(J)
6800 PRINT "R#(J)
6900 PRINT "R#(J)
7000 PRINT "R#(J)
7100 PRINT "R#(J)
7200 PRINT "R#(J)
7300 PRINT "R#(J)
7400 PRINT "R#(J)
7500 PRINT "R#(J)
7600 PRINT "R#(J)
7700 PRINT "R#(J)
7800 PRINT "R#(J)
7900 PRINT "R#(J)
8000 PRINT "R#(J)
8100 PRINT "R#(J)
8200 PRINT "R#(J)
8300 PRINT "R#(J)
8400 PRINT "R#(J)
8500 PRINT "R#(J)
8600 PRINT "R#(J)
8700 PRINT "R#(J)
8800 PRINT "R#(J)
8900 PRINT "R#(J)
9000 PRINT "R#(J)
9100 PRINT "R#(J)
9200 PRINT "R#(J)
9300 PRINT "R#(J)
9400 PRINT "R#(J)
9500 PRINT "R#(J)
9600 PRINT "R#(J)
9700 PRINT "R#(J)
9800 PRINT "R#(J)
9900 PRINT "R#(J)

```

```

0520 INPUT a#(J): PRINT INK 4);a#
0530 NEXT J
0540 CLOS
0550 REM ** obter pergunta a faz
0560 **
0570 FOR J=1 TO factors
0580 GO SUB 1000
0590 PRINT "Por favor indique pe
0600 rgunta";J
0610 INPUT b#(J)
0620 NEXT J
0630 CLOS
0640 REM ** adquirir pericia **
0650 FOR J=1 TO outcomes
0660 CLOS
0670 GO SUB 1000
0680 PRINT "Por favor responde a
0690 seguinte:";"Para um resultado
0700 a#>"a#(J);";"
0710 GO SUB 1000
0720 PRINT "Escreva ""S"" ou ""N""
0730 LET x=.5
0740 FOR k=1 TO factors
0750 LET x=x+x
0760 GO SUB 1000
0770 PRINT TAB 4;")> ");b#(k);
0780 LET multi=0
0790 INPUT g#
0800 IF g#="N" THEN LET multi=1
0810 LET d(J)=d(J)+x*multi: REM
0820 compiler base de pericia
0830 NEXT k
0840 NEXT J
0850 CLOS
0860 PRINT "E' esta a minha base
0870 de conexi-mento:"
0880 FOR J=1 TO outcomes
0890 GO SUB 1000
0900 PRINT a#(J);" ---");d(J)
0910 NEXT J
0920 GO SUB 1000
0930 PRINT TAB 0;"Carregue a# ""
0940 ENTER """"
0950 INPUT g#
0960 CLOS
0970 RETURN
0980 REM *****
0990 REM ** inicializacao **

```

```

0950 CLS
0960 INPUT "Nome do sistema? ";n
#
0970 GO SUB 1060
0980 INPUT "Numero de resultados
? ";outcomes
0990 GO SUB 1060
1000 INPUT "Numero de factores a
considerar?";factors
1010 DIM a$(outcomes,15): DIM b#
(factors,30)
1020 DIM d(outcomes)
1030 CLS
1040 RETURN
1050 REM *****
1060 PRINT : PRINT
1070 RETURN

```

### Escolha de um circuito integrado

Seria pouco prático se tivéssemos de educar o nosso «especialista», como acontece no caso do programa anterior, sempre que quiséssemos utilizar os seus conhecimentos. É improvável, numa situação real, que seja necessário usar um programa assim, completamente dependente de dados externos. O programa que se segue apresenta-nos um corpo de conhecimentos que lhe permitem distinguir entre diferentes processadores completamente codificado em declarações DATA.

Vejamos o programa em acção:

É esta a minha base de conhecimento:

```

TMS 9940 (NMOS)      --- 44
68000 (NMOS)        --- 12
9940 (I3L)          --- 55
MN1610 (NMOS)       --- 45
8086                --- 60

```

```

Possuo : identificar pelas caracte-
rísticas seguintes :
TMS 0040 (NMOS)
00000 (NMOS)
0040 (ICL)
MN1510 (NMOS)
0000
ou Z8001

```

Depois de nos dizer o que pode fazer, o programa pede-nos que respondamos a um certo número de perguntas relacionadas com o circuito que tentamos identificar. Em seguida dirá o nome do circuito:

```

Por favor escreva "S" ou "N".
Utiliza palavras de 32 bits      N
Endereça 84K      N
O impulso de relógio é de 5MHz
ou menos      S
O menor tempo de reacção é 3
microsegundos ou menos      S
Possui mais de 71 instruções      N
O invólucro tem 40 pines      N
O meu resultado foi 12

Possui um
00000 (NMOS)

```

A base de conhecimentos foi novamente obtida junto de um especialista, Ken Ozanne, cujos conhecimentos foram também concentrados num livro (*The Interface Computer Encyclo-*

pedia, Londres; Interface Publications, 1983). Depois de a informação estar incluída em declarações DATA no interior do programa, este pode ser usado em qualquer momento.

Vejamos a parte crucial do programa, onde se encontram armazenados estes conhecimentos:

```

0000 DIM ** inicializacao
0000 DIM S
0000 DIM STORE
0000 DIM OUTCOME#
0000 DIM FACTOR#
0000 DIM a##(outcomes,17): DIM b#
(factors,60): DIM d(outcomes)
0000 FOR j=1 TO outcomes
0000   DIM a##(j),d(j)
0000 NEXT j
0000 FOR j=1 TO factors
0000   DIM b#(j)
0000 NEXT j
0000 PRINT
0000 DIM *****
0000 PRINT : PRINT
0000 PRINT
0000 DIM *****
0000 DIM *****
0000 DATA "TMS 0040 (N400)",44,"
60000 (N400)",10
0000 DATA "0040 (TGL)",66,"MN1610
(N400)",40
0000 DATA "0006",66,"Z0001",28
0000 DIM ** perguntas **
0000 DATA "Utiliza palavras de 3
n bits"
0010 DATA "Endereça 5AK"
0020 DATA "O impulso de relógio
e de 5MHz ou menos"
0030 DATA "O menor tempo de reac
cao e' 3 microssegundos ou meno
s"
0040 DATA "Possui mais de 71 ins
trucoes"
0050 DATA "O involucro tem 40 pe
rnes"

```

Mesmo que o leitor não tenha qualquer interesse em identificar circuitos integrados, pode mesmo assim utilizar o sistema incorporado neste programa. Como pode verificar, as variáveis OUTCOMES e FACTORS são definidas nas linhas 610 e 620, sendo depois usadas para dimensionar quadros na linha 630.

Altere as variáveis tendo em conta os resultados e factores que lhe interessam, modifique as declarações DATA, e disporá de um sistema dedicado que lhe será muito útil. O número crucial que o sistema utiliza para identificar o circuito integrado (ou qualquer outro resultado que lhe interesse) é guardado em declarações DATA imediatamente a seguir ao nome do circuito.

Para determinar estes números, construi a tabela seguinte. Será fácil construir uma tabela semelhante para o tema que lhe interessar:

RESULTADO	FACTOR						TOTAL
	1	2	3	4	5	6	
TMS 9940	0	0	1	1	0	1	44
68000	0	0	1	1	0	0	12
994013 L	0	0	0	1	1	1	56
MN1610	0	1	1	1	0	1	46
8086	0	0	1	1	1	1	60
Z8001	0	0	1	1	1	0	28

Vejamos agora a listagem do programa completo. A partir dela, e de uma tabela de totais como a que acabámos de apresentar, o leitor poderá facilmente criar um sistema dedicado, um «especialista» no assunto que mais lhe interessar:

```

10 REM Circuitos integrados
15 POKE 20000,0: FLASH 0: BR
IGHT 1: INK 7: PAPER 1: BRIGHT 0
: BORDER 1: CLS
20 GO SUB 300: REM inicializar

```

```

30 GO SUB 400: REM mostrar con
teúdo da base
40 GO SUB 100: REM identificar
circuito
50 GO SUB 700
60 PRINT "Corregue ""ENTER"" p
ara continuar ou outra para para
r."
70 INPUT A#
80 IF A#="" THEN GO TO 40
90 STOP
100 REM *****
110 REM ** identificar circuito
**
120 GOS
130 GO SUB 700
140 PRINT @RIGHT 1;"posso ident
ificar pelas caracte-rísticas se
guintes:"
150 FOR J=1 TO outcomes
170 PRINT TAB J+2;
180 IF J=outcomes THEN PRINT "o
u "
190 PRINT INK 4);a#(J)
200 NEXT J
210 GO SUB 700
220 LET result=0
230 LET x=.0
240 PRINT "Por favor escreva ""
S"" ou ""N""."
250 FOR J=1 TO factors
260 LET x=x+x
270 PRINT "S"(J)
280 INPUT S#
290 IF S#<>"S" AND S#<>"N" THEN
GO TO 200
300 IF S#<>"N" THEN LET result=
result+x
310 NEXT J
320 PRINT TAB 3; INK 3;"O meu r
esultado foi ";result
330 GO SUB 700
340 LET #=0
350 LET #=#+1
360 IF d(#)=result THEN GO TO 4
00
370 IF #<outcomes THEN GO TO 35
0
380 FOR r=1 TO 5: BEEP .1,r: BE
EP .2,r+10: NEXT r: PRINT TAB 3;

```



```
780 DATA "0000",60,"Z0001",20
790 DIM ** perguntas **
800 DATA "Utiliza palavras de 3
a bit"
810 DATA "Endereco 64K"
820 DATA "O impulso de relógio
e de 5MHz
ou menos"
830 DATA "O menor tempo de reac
cao e 0
microsegundos ou meno
s"
840 DATA "Possui mais de 71 ins
trucoes"
850 DATA "O involucro tem 40 pe
ras"
```

## SISTEMAS CAPAZES DE APRENDEREM

O leitor recordará certamente que o segundo sistema observado nesta parte, DIFERENÇA-X, permitia ao utilizador comunicar à máquina os conhecimentos necessários sobre qualquer tema. Depois de estes dados lhe serem comunicados, o programa ficava pronto a aconselhar o utilizador sobre o assunto escolhido.

No entanto o programa apresentava uma grande desvantagem. Exigia que o utilizador respondesse a cada um dos factores correspondentes a cada resultado a fim de obter a base de conhecimentos que lhe permitia tomar decisões.

O programa que se segue, APRENDER, não obriga a ter todo esse trabalho. Vejamo-lo em funcionamento:

```
Quantos factores 3
Indique factor 1 ASA5
Indique factor 2 PAR DE OLHOS
Indique factor 3 COME VERMES
Indique resultado 1 PARDAL
Indique resultado 2 HOMEM
```

Depois de esta informação estar definida, pode executar-se

o programa, e este acabará por descobrir por si mesmo a diferença entre os vários resultados:

```
Agora demonstro os meus conheci-  
mentos...
```

```
Pense num dos resultados
```

```
Sera/  
IGRIS
```

```
Verdadeiro ("S" ou "N")?
```

```
N
```

```
0
```

```
Sera/  
PAR DE OLHOS
```

```
Verdadeiro ("S" ou "N")?
```

```
S
```

```
1
```

```
Sera/  
COMO VERMES
```

```
Verdadeiro ("S" ou "N")?
```

```
N
```

```
0
```

```
> BRAYN=0  
O resultado e' PARDAL
```

```
Esta' correcto ("S" ou "N")?
```

```
N
```

```
Agora demonstro os meus conheci-  
mentos...
```

```
Pense num dos resultados
```

```
Sera/  
IGRIS
```

```
Verdadeiro ("S" ou "N")?
```

```
S
```

```
1
```

```
Sera/  
PAR DE OLHOS
```

```
Verdadeiro ("S" ou "N")?
```

```
S
```

```
1
```

```
Sera/  
COMO VERMES
```

```
Verdadeiro ("S" ou "N")?
```

```
S
```

```
1
```

```
> BRAYN=-1
```

O resultado e' HQMEM

Esta' correcto ("S" ou "N")?

N

**Durante algum tempo o programa cometerá erros, como acabámos de ver, mas depois apresentará resultados cada vez mais correctos:**

Agora demonstro os meus conhecimentos...

Pense num dos resultados

Se eu

for

Verdadeiro ("S" ou "N")?

S

1

Se eu

for

PAR DE OLHOS

Verdadeiro ("S" ou "N")?

S

1

Se eu

for

COMO VERMES

Verdadeiro ("S" ou "N")?

S

1

O resultado e' PARDAL

BRAYN=2

Esta' correcto ("S" ou "N")?

S

Agora demonstro os meus conhecimentos...

Pense num dos resultados

Se eu

for

Verdadeiro ("S" ou "N")?

N

0

Se eu

for

PAR DE OLHOS

Verdadeiro ("S" ou "N")?

S

```

Será?
COME VERMES
Verdadeiro ("S" ou "N")?
N
                                > BRAYN=0
O resultado é PARDAL
Este é correcto ("S" ou "N")?
N

```

### Ao fim de algum tempo tornar-se-á infalível:

Agora demonstro os meus conhecimentos...

Pense num dos resultados

```

Será?
COMO UM
Verdadeiro ("S" ou "N")?
S
                                1

```

```

Será?
PAR DE OLHOS
Verdadeiro ("S" ou "N")?
S
                                1

```

```

Será?
COME VERMES
Verdadeiro ("S" ou "N")?
S
                                1

```

```

                                > BRAYN=1
O resultado é PARDAL
Este é correcto ("S" ou "N")?
S

```

### Como funciona

O aspecto importante (e a principal limitação) deste programa é que apenas consegue distinguir dois resultados (como PARDAL e HOMEM, no nosso exemplo). O programa começa por aceitar que o seu total (a variável BRAYN) será superior ou igual a zero, ou inferior a zero. O valor que de facto BRYAN assume não é relevante.

Quando começa a ser executado, o programa pede as informações grosseiras de que necessita:

```

400 REM ** inicializacao **
410 CLE
420 LET otco=2: REM numero de r
resultados
430 PRINT : PRINT
440 INPUT "Quantos factores ": f
act
450 DIM a$(otco,30): REM nomes
dos resultados
460 DIM b$(fact,30): REM nomes
dos factores
470 DIM c(fact): DIM d(fact)
480 FOR J=1 TO fact
490 PRINT "Indique factor ", J
500 INPUT b$(J): PRINT INK 4) b$(
(J)
510 NEXT J
520 CLE
530 FOR J=1 TO otco
540 PRINT "Indique resultado "
(J)
550 INPUT a$(J): PRINT INK 5) a$(
(J)
560 NEXT J
570 RETURN

```

De cada vez que o ciclo é executado, o programa «aprende» preenchendo elementos do quadro C (existe um elemento para cada FAC) com zeros:

```

60 REM * ciclo principal *
40 CLE
50 FOR J=1 TO fact
60 LET c(J)=0
70 NEXT J
80 PRINT
90 GO SUB 100
100 GO TO 40

```

Em seguida imprime os factores, um a um, pedindo ao utilizador que comente Sim ou Não conforme se referem ou não ao resultado em que este pensa:

```

120 REM * demonstracao *
130 PRINT INK 5) "Agora demonstr

```

```

0  os meus conhecimentos..."
140 PRINT "Pense num dos resul
tabios"
145 PRINT
145 FOR J=1 TO fact
170 PRINT INK 5;"Será: "b$(J)
INK 7; BRIGHT 1;" Verdadeiro ("
"5" ou "N")?"
180 INPUT z$
190 IF z#<>"S" AND z#<>"N" THEN
GO TO 180
200 IF z#="S" THEN LET c(J)=1
210 PRINT TAB 20; INK 4;c(J)
220 NEXT J

```

Se respondemos Sim, então o elemento do quadro C é passado para um. Depois de ter sido executado este ciclo, BRAYN determina um total para o resultado em causa, através das instruções contidas nas linhas 230 a 270:

```

230 LET brayn=0
240 FOR J=1 TO fact
250 LET brayn=brayn+c(J)*d(J)
260 NEXT J
270 PRINT TAB 20;"> BRAYN=";bra
yn

```

Se observarmos cuidadosamente esta linguagem, verificamos que da primeira vez que o ciclo é percorrido, BRAYN será igual a zero (porque todos os C(j) terão sido multiplicados por D(j), e todos os D(j) são inicialmente zero).

Isto significa que da primeira vez que o programa é executado, dar-nos-á a opção um (isto é, A\$(1), o primeiro resultado que indicamos), dado que BRAYN será igual a zero:

```

280 IF brayn>=0 THEN PRINT "O r
esultado é: "a$(1); LET ex=-1
290 IF brayn<0 THEN PRINT "O re
sultado é: "a$(2); LET ex=1

```

Em seguida o programa pergunta se o resultado é correcto. Se respondemos que sim, não modifica as informações de que dispõe, porque, no seu estado actual, dará a mesma informação da próxima vez que for apresentada essa informação. Se, no entanto, lhe dissermos que o resultado é errado, passará ao ciclo seguinte, modificando os valores de D(j) recorrendo ao uso dos valores C(j) que lhe fornecemos e à variável EX. Se

observarmos as linhas 280 e 290, verificaremos que EX é igualada a - 1 quando o resultado pensado foi A\$(1), e a 1 quando o resultado foi A\$(2).

D(j) é o componente vital do ciclo 240 a 260, ajudando a determinar o valor de BRAYN. É portanto necessário alterá-lo se o programa forneceu um resultado incorrecto:

```

300 PRINT "Esta é correcto (" "S"
" ou "N")?"
310 INPUT N#
320 IF N# <> "S" AND N# <> "N" THEN
330 TO 310
340 PRINT
350 IF N# = "S" THEN FOR I=1 TO 5
: BEEP .1, 1: NEXT I: GO TO 380
360 FOR J=-1 TO -5 STEP -1: BEE
P .1, 1: NEXT J
370 FOR I=1 TO fact
380 LET d(J) = d(J) + EX + c(J)
390 NEXT J
400 RETURN

```

Depois de ter feito as alterações necessárias a D(j), usando simultaneamente os valores dos elementos do quadro C (que só podem ser iguais a um ou a zero, como se pode verificar consultando as linhas 60 e 200), o programa tenta novamente. Dentro em pouco torna-se infalível.

Vejam agora a listagem completa:

```

10 REM no reader
20 REM LIGHT 1: FLASH 0: INK 7: B
30 REM R 1: LIGHT 0: PAPER 1: CLR
40 REM P 1: LIGHT 0: PAPER 1: CLR
50 REM 000 400: REM inicializar
60 REM * ciclo principal *
70 REM
80 REM FOR I=1 TO fact
90 LET c(J) = 0
100 NEXT J
110 PRINT I
120 REM TO 40
130 REM *****
140 REM * demonstracao *
150 PRINT INK 5: "demonstrac
0 06 REMUS CORRIGIDOS...
140 PRINT "entre num dos resul
tados"

```

```

1445 PRINT
1450 FOR J=1 TO fact
1460 PRINT INK 5; "Será: ";b#(J);
1470 IF BRIGHT 1;" Verdadeiro ("
"0"00 "00 "N"00?"
1480 INPUT Z#
1490 IF Z#<>"0" AND Z#<>"N" THEN
GO TO 1450
1500 IF Z#="0" THEN LET c(J)=1
1510 PRINT TAB 20; INK 4;c(J)
15200 NEXT J
15300 LET b#braço=0
15400 FOR J=1 TO fact
15500 LET b#braço=b#braço+c(J)*d(J)
15600 NEXT J
15700 PRINT TAB 20;"> BRAYN=";bra
u
15800 IF b#braço=0 THEN PRINT "0 r
resultado e ";b#(1); LET ex=-1
15900 IF b#braço=0 THEN PRINT "0 re
sultado e ";b#(2); LET ex=1
16000 PRINT "Resposta correcta (";"0"
"00 "N"00?"
16100 INPUT Z#
16200 IF Z#<>"0" AND Z#<>"N" THEN
GO TO 1450
16300 PRINT
16400 IF Z#="0" THEN FOR r=1 TO 5
: 0.1,r; NEXT r: GO TO 1650
16500 FOR r=1 TO -5 STEP -1: BEE
: 0.1,r; NEXT r
16600 FOR J=1 TO fact
16700 LET d(J)=d(J)+ex*c(J)
16800 NEXT J
16900 RETURN
17000 REM *****
17100 REM ** inicializacoes **
17200 DIM otco=0: REM numero de r
resultados
17300 PRINT PRINT
17400 INPUT "Quantos factores ";f
a nt
17500 DIM a#(otco,30): REM nomes
resultados
17600 DIM b#(fact,30): REM nomes
factores
17700 DIM c(fact): DIM d(fact)
17800 FOR J=1 TO fact

```

```

510 PRINT "Indique factor ";J;
520 INPUT B$(J): PRINT INK 4);B$(
(0)
530 NEXT J
540 GOTO 510
550 FOR J=1 TO otro
560 PRINT "Indique resultado "
(0)
570 INPUT a$(J): PRINT INK 5);a$(
(0)
580 NEXT J
590 RETURN

```

### Mais de duas alternativas

Se bem que seja fascinante dispor de um sistema dedicado que aprenda por si só, é muito restritivo dispor apenas de duas alternativas. O programa que se segue foi concebido para resolver este problema.

O programa começa da forma que o leitor certamente já espera:

```

Quantos resultados? 3
Quantos factores? 3
Por favor indique resultado 1
?HOMEM
Por favor indique resultado 2
?CAVALO
Por favor indique resultado 3
?PARDAL
Por favor indique factor 1
?UM UNICO PAA DE PERNAS
Por favor indique factor 2
?PODE VOAR PELOS SEUS MEIOS
Por favor indique factor 3
?RESPIRA OXIGENIO
Eis os resultados possíveis:
HOMEM
CAVALO
PARDAL

```

Por favor pense num deles

Carregue em ENTER quando estiver pronto

No entanto, durante a execução, vemos que o programa nos faz algumas perguntas e em seguida tenta adivinhar o resultado em que pensávamos. Se não acerta, pergunta-nos qual seria o resultado correcto:

Por favor responda "S" ou "N" em função do resultado em que pensa

UM UNICO PAR DE PERNAS

? S

PODE VOAR PELOS SEUS MEIOS

? N

RESPIRA OXIGENIO

? S

Estava a pensar em HOMEM

Escreva "S" se estou certo, ou "N" se estou errado

Qual deveria ser a resposta?

? HOMEM

Se o programa for executado durante tempo suficiente (e não será muito se apenas se utilizarem três resultados e três factores), acabará por acertar sempre:

Por favor responda "S" ou "N" em função do resultado em que pensa

UM UNICO PAR DE PERNAS

? N

PODE VOAR PELOS SEUS MEIOS

? N

RESPIRA OXIGENIO

? S

Estava a pensar em CAVALO

Escreva "S" se estou certo, ou  
"N" se estou errado  
? S

Por favor responda "S" ou "N" em  
funcao do resultado em que pensa

UM UNICO PAR DE PERNAS  
? S

PODE VOAR PELOS SEUS MEIOS  
? S

RESPIRA OXIGENIO  
? S

Estava a pensar em PARDAL

Escreva "S" se estou certo, ou  
"N" se estou errado  
S

A parte importante deste programa encontra-se entre as linhas 150 e 520. Nestas linhas começa-se por aceitar as respostas «Sim», o que incrementa uma variável chamada COUNT em função da resposta dada (somando 1 no caso do primeiro «Sim», 2 no caso do seguinte, 4 no terceiro, etc.):

```
150 PRINT "Por favor responda "  
"S" ou "N" em funcao do result  
ado em que pensa"  
160 PRINT  
170 LET count=0  
180 LET x="S"  
190 LET J=1 TO fact  
200 LET x=x+x  
210 PRINT "?(J)  
220 INPUT A#  
230 IF A#<>"S" AND A#<>"N" THEN  
GO TO 140  
240 IF A#="S" THEN LET count=count  
+x  
270 NEXT J
```

Depois de o programa funcionar durante algum tempo, terá atribuído valores a muitos elementos do quadro B. B(1)

será o total quando o resultado for A\$(1), B(6) será o total para um resultado A\$(6), etc. É executado um pequeno ciclo depois de ser dada a resposta Sim, que serve para verificar se o total obtido é igual a algum valor já guardado em B(j). Se assim acontecer, a variável X é igualada ao J relevante:

```
280 LET X=0
290 FOR J=1 TO otco
300 IF count=b(J) THEN LET X=J
310 NEXT J
```

Se esse valor tiver sido atribuído, X deixará de ser igual a zero, e o sistema terá tomado uma decisão:

```
320 IF X<>0 THEN GO TO 410
```

Se não tiver sido obtida aqui qualquer resposta definida, o computador produz um número aleatório entre 1 e o número de resultados, a fim de adivinhar o resultado. Mas não basta então dizer «ESTAVA A PENSAR EM»: A\$ (número aleatório). Se bem que o programa possa ainda não ter atribuído um A\$(n) ao total obtido, pode já ter definido alguns elementos de A\$. Pode portanto, e deve, rejeitar algumas «suposições» produzidas pelo gerador de números aleatórios:

```
330 LET X=INT (RND*otco)+1
340 REM ** rejeitar respostas
      erradas **
350 LET flag=0
360 FOR J=1 TO otco
370 IF b(J)=0 THEN GO TO 390
380 IF X=J AND count<>b(J) THEN
LET flag=1
390 NEXT J
400 IF flag=1 THEN GO TO 330
```

Se a variável FLAG é igual a qualquer valor diferente de zero, então a hipótese correspondente (o elemento de A\$ representado pelo valor aleatório de X) não pode ser usada, dado que o sistema já sabe que a resposta está errada. Nestas condições, usando a linha 400, escolhe um novo valor para X.

```
410 PRINT "Estava a pensar em "
;A$(X)
420 PRINT
430 PRINT "Escreva ""S"" se est
```

```

OU certo, ou "" "" "" "" "" "" "" "" se estou er
rado."
440 INPUT Z$
450 IF Z#(<>"S" AND Z#(<>"N" THEN
GO TO 440
460 IF Z#="S" THEN LET b(x)=cou
nt: FOR r=1 TO 5: BEEP .1,2: BEE
P .1,3: BEEP .1,r: GO TO 30
470 FOR r=1 TO 4: BEEP .1,7: BE
EP .2,8: BEEP .1,r*2: NEXT r: PR
INT "Qual deveria ser a resposta
?"
480 INPUT Z$
490 FOR J=1 TO otco
500 IF a#(J) TO LEN Z#)=Z# THEN
LET b(J)=count
510 NEXT J
520 GO TO 30

```

Se a hipótese estiver correcta, o sistema usa o ciclo entre 490 e 510 para descobrir qual o elemento de A\$ que corresponde ao total produzido nessa execução. Garante-se assim que, quando encontrar novamente o mesmo total, o programa seja capaz de identificar o elemento relevante de A\$.

Estudemos agora a listagem completa deste sistema capaz de aprender por si mesmo:

```

10 REM Multi-aprendizagem#
15 INVERSE 0: POKE 23050,200:
BRIGHT 1: FLASH 0: PAPER 1: INK
7: BRIGHT 0: BORDER 1: CLS
20 GO SUB 530: REM inicializar
30 CLS
40 PRINT "Eis os resultados po
ssiveis:"
50 PRINT
60 FOR J=1 TO otco
70 PRINT TAB J);a#(J)
80 NEXT J
90 PRINT
100 PRINT "Por favor pense num
deles:"
110 PRINT
120 PRINT "Carregue em "); FLASH
1: "ENTER"; FLASH 0;" quando est
iver pronto"
130 INPUT Z$

```

```

140 GOCS
150 PRINT "Por favor responde "
160 INPUT N#
170 PRINT "Resposta do resultado"
180 PRINT "Resposta"
190 PRINT count=0
200 PRINT X=0
210 FOR J=1 TO fact
220 PRINT X=X*X
230 PRINT X*(J)
240 INPUT Z#: PRINT "? "; FLASH
250 #
260 IF Z#<>"0" AND Z#<>"N" THEN
270 TO 240
280 IF Z#="0" THEN LET count=0
290 +X
300 NEXT J
310 PRINT X=0
320 FOR J=1 TO otco
330 IF count=b(J) THEN LET X=J
340 NEXT J
350 IF X<>0 THEN GO TO 410
360 LET X=INT (RND*otco)+1
370 REM ** rejeitar respostas
380 ** erradas **
390 LET flag=0
400 FOR J=1 TO otco
410 IF b(J)=0 THEN GO TO 390
420 IF X=0 AND count<>b(J) THEN
430 PRINT flag=1
440 NEXT J
450 IF flag=1 THEN GO TO 390
460 PRINT "Estava a pensar em "
470 #(X)
480 PRINT
490 PRINT "Escreva "0" se est
500 certo, ou ""N"" se estou er
510 rado"
520 INPUT Z#
530 IF Z#<>"0" AND Z#<>"N" THEN
540 TO 440
550 IF Z#="0" THEN LET b(X)=count
560 FOR r=1 TO 9: BEEP .1,r: BEE
570 .1,9: BEEP .1,r: GO TO 56
580 FOR r=1 TO 4: BEEP .1,r: BE
590 .1,0: BEEP .1,r*2: NEXT r: BE
600 PRINT "Ocult deveria ser a resposta"
610 #
620 INPUT Z#
630 FOR J=1 TO otco
640 IF #(J), TO LEN Z#)=Z# THEN

```

```

0010 LET B(J) = COUNT
0020 NEXT J
0030 GO TO 00
0040 PRINT *****
0050 PRINT "*****"
0060 PRINT "Quantos resultados?"
0070 INPUT otco
0080 PRINT INK 5; otco
0090 PRINT "Quantos factores?"
0100 INPUT fact
0110 PRINT INK 5; fact
0120 LET x = otco + fact
0130 DIM a#(otco, 00)
0140 DIM f#(fact, 00): DIM b(x)
0150 FOR J = 1 TO otco
0160 PRINT "Por favor indique re
sultado"; J
0170 INPUT a#(J): PRINT INK 4; "?"
0180 a#(J)
0190 NEXT J
0200 CLR
0210 FOR J = 1 TO fact
0220 PRINT "Por favor indique fa
ctor"; J
0230 INPUT f#(J): PRINT INK 4; "?"
0240 f#(J)
0250 NEXT J
0260 RETURN

```

### Correcções desnecessárias

O último programa desta parte constitui uma variante daquele que acabámos de estudar. A única diferença entre ambos é que não é necessário dizer ao novo programa qual deveria ter sido a resposta quando comete um erro. O programa acaba por descobri-la sozinho e com razoável rapidez.

Não renumerei este programa, o que permitirá ao leitor modificar mais facilmente a listagem anterior. Vejamos um exemplo de execução:

```

Quantos resultados? 5
Quantos factores? 5
Por favor indique resultado 1

```

```

?CORVO
?CORVO favor indique resultado 10
?CORVO
?CORVO
?CORVO favor indique resultado 3
?CORVO
?CORVO favor indique resultado 4
?CORVO
?CORVO favor indique resultado 5
?HOMEM

```

```

Por favor indique factor 1
?CORVIN DE FALDA

```

```

Por favor indique factor 2
?LADRA

```

```

Por favor indique factor 3
?CORVIN DE VOZ

```

```

Por favor indique factor 4
?POSSUI QUATRO PERNAS

```

```

Por favor indique factor 5
?PRODUZ LA

```

Depois de ser construída a base de conhecimento, o programa prossegue do seguinte modo:

Estes os resultados possíveis:

```

CORVO
  CORVO
    CARNEIRO
      CORVO
        HOMEM

```

Por favor pense num deles

Carregue em ENTER quando estiver pronto

Por favor responda  
 "S" ou "N" em  
 função do resultado em que pensa

CAPAZ DE FALAR  
? N

LADRA  
? S

CAPAZ DE VOAR  
? N

POSSUI QUATRO PERNAS  
? S

PRODUZ LA  
? N

Estava a pensar em CAVALO

Escreva "S" se estou certo, ou  
"N" se estou errado  
? N

**Se bem que a maior parte das respostas sejam inicialmente erradas, os resultados correctos tornar-se-ão progressivamente mais frequentes:**

Por favor responda "S" ou "N" em  
função do resultado em que pensa

CAPAZ DE FALAR  
? N

LADRA  
? N

CAPAZ DE VOAR  
? S

POSSUI QUATRO PERNAS  
? N

PRODUZ LA  
? N

Estava a pensar em CORVO

**Ao fim de algum tempo o programa deixará de cometer erros. Funciona de forma bastante semelhante à do programa anterior, atribuindo valores aos elementos do quadro B. No**

entanto, possui desta vez um quadro designado por C, que garante que nunca será cometido duas vezes um mesmo erro, partindo de um mesmo total. Isto limita progressivamente mais o número de resultados possíveis, sendo reduzido cada vez mais o número de hipóteses. Não é necessário muito tempo para construir uma «visão do mundo» que garanta a obtenção de resultados certos em todos os casos. O número de tentativas necessárias para atingir esta perfeição depende, obviamente, do número de resultados.

Depois de o programa conseguir aprender por si mesmo a diferença entre os vários resultados, passa a dispor de uma base de conhecimento de que se utilizará para cada nova execução subsequente. Vejamos a base que o programa constrói no exemplo considerado:

```

J 0001 00000000 0000 0000 00000000
C 0001 0001 0001 0001 00010000
C 0001 0001 0001 0001 00010001
C 0001 0001 0001 0001 00010004
0000 0000 0000 0000 00000000
0000 0000 0000 0000 00000000
0000 0000 0000 0000 00000000
0000 0000 0000 0000 00000000

```

E agora a listagem completa:

```

1100  DD 0000 MULT 4 - 0000
1110  DD 0000 MULT 4 - 0000
1120  DD 0000 MULT 4 - 0000
1130  DD 0000 MULT 4 - 0000
1140  DD 0000 MULT 4 - 0000
1150  DD 0000 MULT 4 - 0000
1160  DD 0000 MULT 4 - 0000
1170  DD 0000 MULT 4 - 0000
1180  DD 0000 MULT 4 - 0000
1190  DD 0000 MULT 4 - 0000
1200  DD 0000 MULT 4 - 0000
1210  DD 0000 MULT 4 - 0000
1220  DD 0000 MULT 4 - 0000
1230  DD 0000 MULT 4 - 0000
1240  DD 0000 MULT 4 - 0000
1250  DD 0000 MULT 4 - 0000
1260  DD 0000 MULT 4 - 0000
1270  DD 0000 MULT 4 - 0000
1280  DD 0000 MULT 4 - 0000
1290  DD 0000 MULT 4 - 0000
1300  DD 0000 MULT 4 - 0000
1310  DD 0000 MULT 4 - 0000
1320  DD 0000 MULT 4 - 0000
1330  DD 0000 MULT 4 - 0000
1340  DD 0000 MULT 4 - 0000
1350  DD 0000 MULT 4 - 0000
1360  DD 0000 MULT 4 - 0000
1370  DD 0000 MULT 4 - 0000
1380  DD 0000 MULT 4 - 0000
1390  DD 0000 MULT 4 - 0000
1400  DD 0000 MULT 4 - 0000
1410  DD 0000 MULT 4 - 0000
1420  DD 0000 MULT 4 - 0000
1430  DD 0000 MULT 4 - 0000
1440  DD 0000 MULT 4 - 0000
1450  DD 0000 MULT 4 - 0000
1460  DD 0000 MULT 4 - 0000
1470  DD 0000 MULT 4 - 0000
1480  DD 0000 MULT 4 - 0000
1490  DD 0000 MULT 4 - 0000
1500  DD 0000 MULT 4 - 0000
1510  DD 0000 MULT 4 - 0000
1520  DD 0000 MULT 4 - 0000
1530  DD 0000 MULT 4 - 0000
1540  DD 0000 MULT 4 - 0000
1550  DD 0000 MULT 4 - 0000
1560  DD 0000 MULT 4 - 0000
1570  DD 0000 MULT 4 - 0000
1580  DD 0000 MULT 4 - 0000
1590  DD 0000 MULT 4 - 0000
1600  DD 0000 MULT 4 - 0000
1610  DD 0000 MULT 4 - 0000
1620  DD 0000 MULT 4 - 0000
1630  DD 0000 MULT 4 - 0000
1640  DD 0000 MULT 4 - 0000
1650  DD 0000 MULT 4 - 0000
1660  DD 0000 MULT 4 - 0000
1670  DD 0000 MULT 4 - 0000
1680  DD 0000 MULT 4 - 0000
1690  DD 0000 MULT 4 - 0000
1700  DD 0000 MULT 4 - 0000
1710  DD 0000 MULT 4 - 0000
1720  DD 0000 MULT 4 - 0000
1730  DD 0000 MULT 4 - 0000
1740  DD 0000 MULT 4 - 0000
1750  DD 0000 MULT 4 - 0000
1760  DD 0000 MULT 4 - 0000
1770  DD 0000 MULT 4 - 0000
1780  DD 0000 MULT 4 - 0000
1790  DD 0000 MULT 4 - 0000
1800  DD 0000 MULT 4 - 0000
1810  DD 0000 MULT 4 - 0000
1820  DD 0000 MULT 4 - 0000
1830  DD 0000 MULT 4 - 0000
1840  DD 0000 MULT 4 - 0000
1850  DD 0000 MULT 4 - 0000
1860  DD 0000 MULT 4 - 0000
1870  DD 0000 MULT 4 - 0000
1880  DD 0000 MULT 4 - 0000
1890  DD 0000 MULT 4 - 0000
1900  DD 0000 MULT 4 - 0000
1910  DD 0000 MULT 4 - 0000
1920  DD 0000 MULT 4 - 0000
1930  DD 0000 MULT 4 - 0000
1940  DD 0000 MULT 4 - 0000
1950  DD 0000 MULT 4 - 0000
1960  DD 0000 MULT 4 - 0000
1970  DD 0000 MULT 4 - 0000
1980  DD 0000 MULT 4 - 0000
1990  DD 0000 MULT 4 - 0000
2000  DD 0000 MULT 4 - 0000

```

```

180 PRINT
190 LET count=0
200 LET x=.5
210 FOR J=1 TO fact
220 LET x=x*x
230 PRINT BRIGHT 1;";$(J)
240 INPUT Z$: PRINT "? "; FLASH
250 # ""
260 IF Z#(">"S" AND Z#(">"N" THEN
GO TO 240
270 IF Z#="S" THEN LET count=co
unt+x
280 NEXT J
290 LET x=0
300 FOR J=1 TO otco
310 IF count=b(J) THEN LET x=J
320 NEXT J
330 IF x<>0 THEN GO TO 410
340 LET x=INT (RND*otco)+1
350 REM ** rejeitar respostas
erradas **
360 LET flag=0
370 FOR J=1 TO otco
380 IF b(J)=0 THEN GO TO 380
390 IF x=J AND count<>b(J) THEN
LET flag=1
400 IF c(x)=count OR d(x)=count
THEN LET flag=1
410 NEXT J
420 IF flag=1 THEN GO TO 330
430 PRINT "Estava a pensar em "
)$(x)
440 PRINT
450 PRINT "Escreva ""S"" se est
ou certo, ou ""N"" se estou er
rado"
460 INPUT Z$
470 IF Z#(">"S" AND Z#(">"N" THEN
GO TO 440
480 IF Z#="S" THEN LET b(x)=cou
nt: FOR r=1 TO 5: BEEP .1,2: BEE
P .1,5: BEEP .1,7: GO TO 30
490 IF c(x)=0 THEN LET c(x)=cou
nt: GO TO 30
500 LET d(x)=count: GO TO 30
510 STOP
520 REM *****
530 REM inicializacao
540 CLS
550 PRINT "Quantos resultados?"

```

```

600 INPUT otro
610 PRINT INK 5; otro
620 PRINT "Cuantos factores? ";
: INPUT fact
630 PRINT INK 5; fact
640 LET x=otro+fact
650 DIM a$(otro,30)
660 DIM f$(fact,30) : DIM b(x) :
DIM c(x) : DIM d(x)
670 FOR J=1 TO otro
680 PRINT "Por favor indique re
sultado "; J
690 INPUT a$(J) : PRINT INK 4; "?"
700 a$(J)
710 NEXT J
720 CLS
730 FOR J=1 TO fact
740 PRINT "Por favor indique fa
ctor "; J
750 INPUT f$(J) : PRINT INK 4; "?"
760 f$(J)
770 NEXT J
780 RETURN

```

<b>PREFÁCIO</b> .....	9
<b>I PARTE – PENSAR</b> .....	11
<b>1 – APRENDER E RACIOCINAR</b> .....	11
Feedback .....	13
Como pensam as máquinas? .....	14
Interruptores e decisões .....	16
Portas lógicas «reais» .....	18
<b>2 – UM PROGRAMA QUE APRENDE</b> .....	21
Samuel e as damas .....	23
«Três em Linha» – o programa .....	25
<b>3 – UM PROGRAMA QUE RACIOCINA</b> .....	38
Voltando ao programa .....	43
<b>II PARTE – PROCURANDO</b> .....	54
<b>4 – ÁRVORES DE DADOS</b> .....	54
A que se chama árvore? .....	54
Processamento paralelo .....	56
Não há «damas» nem movimentos para trás .	57
Aprofundando .....	60
Mini-maxing .....	62
Peso dos elementos .....	67
O algoritmo alfa-beta .....	70
Como funciona o programa .....	81
<b>5 – VALOR DOS JOGOS</b> .....	98
A complexidade do mundo real .....	99
Outros jogos, outras lições .....	100

<b>III PARTE – FALAR</b> .....	103
<b>6 – COMPREENDER A LINGUAGEM NATURAL</b> .....	103
«Parsing» da linguagem .....	104
Problemas .....	107
Sintaxe e semântica .....	108
<b>7 – BLOCO DE PALAVRAS</b> .....	111
Como funciona? .....	115
Módulos do programa .....	121
Melhoramento do programa .....	138
<b>8 – O DOUTOR ESTÁ</b> .....	142
O «amigo russo» .....	145
Perspectivas muito chocantes para Weizenbaum .....	146
O programa .....	148
Como funciona .....	152
A base de dados de resposta .....	153
A estrutura do programa .....	157
Uso de myflag .....	160
<b>9 – TRADUÇÃO MECÂNICA</b> .....	172
Os seres humanos ainda são úteis .....	172
Portugalês .....	176
Estrutura do programa .....	177
<b>10 – HANSHAN</b> .....	184
<b>IV PARTE – AUXILIARES</b> .....	189
<b>11 – SISTEMAS DEDICADOS</b> .....	189
Limitações .....	192
A estrutura química e dendral .....	192
<b>12 – DIFERENÇA</b> .....	196
Diferença X .....	201
Escolha de um circuito integrado .....	215
<b>13 – SISTEMAS CAPAZES DE APRENDEREM</b> .....	222
Como funciona .....	225
Mais de duas alternativas .....	230
Correcções desnecessárias .....	236

Fotocomposto por  
Nova Força - Lisboa  
e impresso na  
Empresa Gráfica Feirense, Lda.  
Vila da Feira  
em 1985  
para a Editorial Presença, Lda.