**COLÉGIO EVANGÉLICO ALBERTO TORRES - CEAT**
**ÁREA DE INFORMÁTICA**

# MANIPULAÇÃO DE *SPRITES* EM LOGO

**PAULO ROBERTO BAGATINI**
**Segundo Grau - Terceiro Ano**
**Informática**

**Lajeado, outubro de 1991**

"Se você já realizou tudo o que planejou,
você não planejou o suficiente."

Martial

**SUMÁRIO**



## INTRODUÇÃO

Este trabalho é a síntese de oito meses de pesquisa, experiências e muita programação em torno de um tema ordinariamente desconhecido pela maioria dos que trabalham com LOGO: a manipulação de sprites.
Em poucas palavras, pode-se dizer que, para o LOGO, sprites são pequenas telas de 16x16 pontos nas quais o programador pode criar uma figura. Entre os comandos primitivos integrados na linguagem estão os que permitem utilizar o recurso riquíssimo que os sprites representam. Contudo, se analizarmos com cuidado, notaremos que essas primitivas deixam a desejar no momento em que nos dispomos a explorar com vontade as figuras dos sprites. A programação é lenta e cansativa. É freqüente perdemos o controle sobre o conteúdo de cada sprite. A idéia então foi produzir um programa que nos desse total controle sobre os sprites. O procedimento adotado foi o de criar e juntar em menus as primitivas que faltassem para a programação.
Foi um trabalho duro no qual teve imenso valor a ajuda prestada pelo colega Marcelo Bihre, a quem sou muito grato. Meus agradecimentos também ao professor Hugo Landmeier, pelas manifestações positivas em relação ao trabalho, e ao CEAT, como entidade, por ter permitido desenvolver em seu Laboratório de Informática o projeto e o presente relatório, alvos desta Mostra de Ciências.

## 1 - DEFINIÇÃO DO PROBLEMA

É possível ampliar e desenvolver os métodos de edição e programação de sprites em LOGO? Se for, o quê deve ser desenvolvido e como fazê-lo?

## 2 - JUSTIFICATIVA

Considerando:
- que o LOGO é relativamente pobre no que diz respeito à programação de sprites;
- que o assunto "sprite" é mais abrangente do que simplesmente "desenho";
- que até agora esse campo foi pouco explorado em LOGO;
- que é interessante abordar em LOGO um assunto que outras Linguagens já abordaram;
- que desenvolvendo programas nessa área, indiretamente desenvolve-se programas úteis em outras áreas;
- que nessa área pode-se aplicar, com vantagens, programas desenvolvidos em outras áreas;
- que o processo de elaboração de algo completamente novo, beneficia visivelmente nossa capacidade de raciocínio lógico e abstrato;
- que as idéias desenvolvidas com o projeto serão bastante úteis aos que realmente se interessarem pelo assunto;
- justifica-se a realização do projeto.

## 3 - OBJETIVOS

### 3.1 - PRIMÁRIOS

- desenvolver rotinas que possam rolar, rotar, deletar, inverter, editar, ampliar e reduzir sprites;
- desenvolver um procedimento pelo qual se programe um sprite através do comando primitivo "listafig", com entrada de dados em binário ou decimal;
- entender o processo pelo qual passa o sprite durante sua programação no editor ou no comando "listafig".

### 3.2 - SECUNDÁRIOS

- entender o que significam e para que servem as primitivas: "edfig", "criafigl", "listafig", e "copiafig";
- aprender a utilizar o sistema de notação binária e a fazer transformações do tipo bin-dec e dec-bin;
- desenvolver funções recursivas e subprocedimentos que sejam úteis em outras áreas do LOGO;
- utilizar procedimentos desenvolvidos noutros projetos;
- produzir um programa bem apresentável esteticamente, que reúna o produto dos objetivos primários e que permita manter um rígido controle sobre as figuras dos sprites;
- generalizar ao máximo o programa final.

## 4 - HIPÓTESES

- a máquina com a qual o projeto está sendo desenvolvido talvez não tenha memória e/ou processamento rápido o suficiente para a execução de certos procedimentos, como por exemplo "rolar", devido às transformações matemáticas necessárias, que consomem demasiado tempo e memória;
- conforme o número total final de procedimentos do projeto, o programa pode se tornar inviável, uma vez que a maior parte da memória será ocupada pelos procedimentos. Caso isso aconteça, é provável ter que separar os programas em dois ou mais arquivos de disco;
- falta de lógica em procedimentos poderá impedir o bom andamento do trabalho, fazendo o programa "empacar" no meio do caminho;
- é possível que o tempo disponível para a programação não seja suficiente para que se possa concluir de modo satisfatório uma parte significativa do projeto até o momento de edição da VII MOSTRA DE TRABALHOS PRÁTICOS DO CEAT.

## 5 - MATERIAL USADO

### 5.1 - MATERIAL PARA O DESENVOLVIMENTO DO PROJETO

- computador HB8000;
- aparelho televisor colorido PANASONIC;
- drive 5¼;
- disco 5¼ face dupla;
- fonte transformadora para o drive;
- cartucho HOT-LOGO versão 1.1;
- editor de textos MSXWORD versão 3.0;
- impressora EPSON FX 100+;
- folhas tamanho ofício;
- papel próprio para impressora.

## 5.2 - MATERIAL PARA A MOSTRA

- computador HB8000;
- aparelho televisor colorido PANASONIC;
- drive 5¼;
- disco 5¼ face dupla com o projeto desenvolvido;
- fonte transformadora para o drive;
- cartucho HOT-LOGO versão 1.1;
- relatório do projeto.

# 6 - DEFINIÇÕES

Devido à proposta do projeto, este relatório apresentará expressões próprias da linguagem LOGO. Com o intuito de facilitar a compreensão da leitura e evitar futuras confusões, serão apresentadas abaixo três definições básicas. Foram formuladas por mim e por isso é possível que nem todos concordem com elas. Não devemos nos esquecer no entanto, que o certo ou errado em trabalhos inovadores é muitas vezes relativo já que neles definições pré-estabelecidas freqüentemente não são capazes de abranger de modo satisfatório o novo contexto em que se inserem. No LOGO isso acontece principalmente porque é uma linguagem com fins imediatos educacionais. Por tratar com crianças, tomou-se cuidado ao escolher as expressões que a linguagem usaria. Elas deveriam dar essencialmente uma idéia de concretude, uma vez que a capacidade de abstração das crianças é relativamente reduzida.
Para dar essa aparência palpável aos programas gráficos, o LOGO reconhece três entidades básicas: a TARTARUGA, o SPRITE (pronuncia-se "spraite") e a FIGURA. Apesar de para muitos serem a mesma coisa, são distintas. Distintas, mas de tal modo entrelaçadas que uma depende da outra para existir. Pode-se comparar a relação TARTARUGA - SPRITE - FIGURA à relação PESSOA (invisível) - ROUPA - ESTAMPA (da roupa). Ambas as relações podem ser esquematizadas segundo a fig 6.1, onde a entidade mais externa abrange a mais interna e onde as setas indicam a área de ação de cada primitiva.

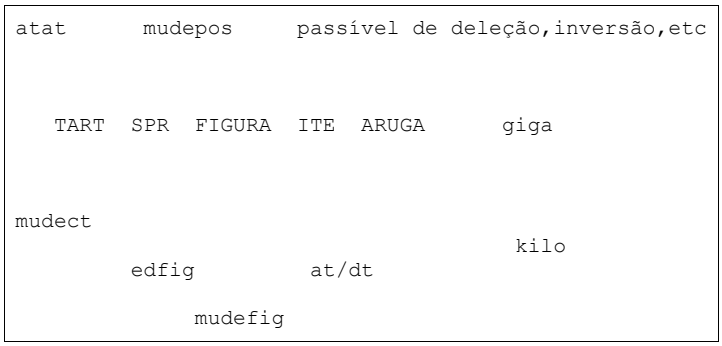



fig 6.1

### TARTARUGA

Se o LOGO é tão conhecido, devemos isso principalmente à sua famosa tartaruguinha. Apesar de para nós essa tartaruga ser o desenho que aparece na tela sempre que ligamos o computador, para o LOGO ela é uma entidade, quase um ser vivo, que dirige os desenhos na tela. É chamada pelo interpretador LOGO de TARTARUGA, não porque o desenho que aparece é o de uma tartaruga, mas porque a entidade que recebe nossas ordens para fazer os desenhos tem por nome TARTARUGA. Essa diferença sutil justifica-se pelo fato da tartaruga (letra minúscula) poder ser trocada por um cachorro ou helicóptero, continuando no entanto a ser tratada pelo LOGO como TARTARUGA (letra maiúscula). Podemos acessar em LOGO, pelo comando primitivo "atat" até 30 TARTARUGAs, 30 entidades independentes entre si, numeradas de 0 a 29, capazes de realizar suas próprias funções. A que normalmente usamos é a de número 0.

### SPRITE

Além disso, se usarmos a comparação, a TARTARUGA pode trocar de roupa. Pode tirar sua roupa de tartaruga e colocar uma de cachorro, por exemplo. É possível escolher entre 60 roupas, rotuladas de 0 a 59. Essas roupas são chamadas em LOGO de SPRITEs. Com dimensões quadrangulares de 16x16 pixels (pontos na tela), quando deletados (apagados), têm a forma de um quadrado cheio. O comando "mudefig" permite trocarmos de SPRITE, ou, de acordo com a comparação, mudarmos a roupa da TARTARUGA. O parâmetro do comando é um número inteiro de 0 a 59, correspondente ao SPRITE desejado.

FIGURA

Sabemos que cada SPRITE tem uma FIGURA. O SPRITE 36, por exemplo, contém a FIGURA de uma tartaruga voltada para cima na tela. Acontece que podemos modificar a FIGURA desse SPRITE. Podemos editá-lo através da primitiva "edfig" e modificar seus 256 pontos (bits). O fato do desenho do SPRITE poder ser mudado reforça a idéia da nossa comparação: podemos dizer que a FIGURA do SPRITE é a estampa da roupa da TARTARUGA, pois pode ser mudada segundo a nossa vontade.
Agora que foram definidas as 3 entidades do LOGO, podemos fazer uma síntese da relação entre elas.

TARTARUGA - SPRITE - FIGURA

A TARTARUGA é invisível, não tem forma. Ela é o espírito do SPRITE assim como o SPRITE é o espírito da FIGURA. Esta última é a única das três entidades que tem forma. Ainda assim, só porque está contida no SPRITE. Por causa disso, ela dá forma ao SPRITE, que por sua vez dá forma à TARTARUGA. É por isso que comandando "dt" não provocamos o desaparecimento da TARTARUGA (embora nos dê essa impressão), mas sim o desligamento do SPRITE. Acontece o mesmo que acontece à uma TV ligada, quando subitamente falta luz. Segundo a comparação, seria como se tirássemos a roupa da TARTARUGA.
Da mesma forma, não devíamos dizer que editar um SPRITE é editar uma figura (apesar do formato do comando "edfig"), já que a FIGURA é resultado da edição e modificação do conteúdo do SPRITE.Veja bem, 1 SPRITE (existem 60) é uma seqüência de 32 bytes, onde cada byte é uma seqüência de 8 bits, ou seja, num SPRITE existem 32x8=256 bits. Cada bit pode estar aceso ou apagado. Essa combinação aceso/apagado, é que forma a FIGURA. Com um pouco de análise combinatória, podemos deduzir que em 1 SPRITE de 256 bits podem ser formadas até $2^{256}$ FIGURAs diferentes. Ou, se tomarmos os bytes teremos $256^{32}$ FIGURAs distintas, uma vez que 1 byte apresenta 256 combinações diferentes. Note que:

$$2^{256} = 256^{32} = 115792089237316195423570985008687907853269984665640564039457584007913129639936$$

Sendo assim, se FIGURA fosse o mesmo que SPRITE, SPRITE seria o mesmo que FIGURA e existiriam em LOGO não 60, mas $2^{256}$ SPRITEs independentes, o que é um absurdo. Logo, está mais do que claro que FIGURA não é o mesmo que SPRITE.

7 - NOÇÕES DE NUMERAÇÃO BINÁRIA

O trabalho com sprites exige, além de certo ponto, conhecer e saber operar com número binários. Normalmente, para executar nossos cálculos, utilizamos o sistema de numeração decimal, que é formado através da combinação de dez símbolos diferentes (0 a 9). Por que dez e não oito ou vinte símbolos diferentes? Provavelmente devido ao número de dedos de nossas mãos, pois nada impede que se adote um número maior ou menor de símbolos para um determinado sistema de numeração.
O sistema utilizado em computadores é o sistema de numeração binária, já conhecido na época de Newton. Os números binários foram desenvolvidos por Boole, e podem ser encontrados em "Álgebra Booleana". Utilizam apenas dois símbolos, ou seja, trabalham na base binária ao invés da decimal. Os dois símbolos são 1 e 0, podendo ser representados respectivamente por SIM e NÃO, uma lâmpada ACESA e uma APAGADA, uma chave LIGADA e uma DESLIGADA, tensão de 5 e de 0 volts.
O motivo da utilização de números binários em computadores é que, para eles, é muito mais fácil trabalhar com chaves abertas ou fechadas, presença ou não de tensão, lâmpadas acesas ou apagadas, do que com dez diferentes símbolos. Abaixo, a tabela da fig 7.1 compara a numeração binária com a decimal:

| base bin | base dec | | base bin | base dec |
|---|---|---|---|---|
| 0000 | 00 | | 0110 | 06 |
| 0001 | 01 | | 0111 | 07 |
| 0010 | 02 | | 1000 | 08 |
| 0011 | 03 | | 1001 | 09 |
| 0100 | 04 | | 1010 | 10 |
| 0101 | 05 | | 1011 | 11 |

fig 7.1

Note que a tabela pode continuar indefinidamente, tanto na numeração decimal, quanto na binária. É importante salientar também, que os 00 à esquerda do número, tanto decimal quanto binário, servem apenas para caracterizar o número em termos de quantidade de dígitos, podendo perfeitamente serem ignorados. Se usássemos lâmpadas para representar números binários, as apagadas seriam os 00, mas nem por isso as que estivessem à esquerda dos números seriam retiradas, já que poderiam ser reutilizadas em outras situações, exemplo:

```
O O O O => 1
O O O O => 9
O O O O => 9
O O O O => 1
```

fig 7.2

A numeração binária não se restringe aos números inteiros, mas se estende até os reais. Foge, porém, ao objetivo desse relatório a exploração de números irracionais ou fracionários binários, uma vez que nos sprites utiliza-se apenas números inteiros. Não deixa contudo, de ser uma sugestão, que um trabalho futuro explore também este lado dos números binários. Veja ainda, que a base binária permite os mesmos operadores que a base decimal, por exemplo: 010+100=110 <=> 2+4=6.
Existem dois termos muito usados em computação: bit e byte. O bit é definido como um elemento da memória que pode estar em duas situações: 1 ou 0, sim ou não, etc. Na tela, um bit em 1 é chamado de bit aceso, e é um ponto

ligado (em qualquer cor) cuja unidade de medida é o pixel. Por definição, um ponto (bit aceso) mede um pixel. Já o bit em zero (0) é um ponto apagado, que não aparece na tela. Nos números binários, os algarismos são denominados bits, sendo representados também por 1 ou 0. Byte é um número binário de 8 bits.
Do mesmo modo que bit é elemento de byte, byte é elemento de muitos sistemas em computação: a matriz das letras de um computador, por exemplo, é um quadrado de 8x8 bits, logo, uma pilha de 8 bytes. Um número decimal é considerado 1 byte, quando seu correspondente binário for um número de 8 bits. A fig 7.3 mostra a estruturação de 1 byte e seus 8 bits:

| B Y T E | | | | | | | |
|---|---|---|---|---|---|---|---|
| bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
| 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 |

Obs: 0/1 => 0 ou 1 fig 7.3

São comuns também, e cada vez mais, devido ao aumento da memória RAM dos computadores, o uso dos múltiplos de byte: kilo(K), mega(M), giga(G) e tera(T). Contudo, nos binários não se usa, como na base decimal,

$1x10^3$ para Kb, mas sim $1x2^{10}$;
$1x10^6$ para Mb, mas sim $1x2^{20}$;
$1x10^9$ para Gb, mas sim $1x2^{30}$;
$1x10^{12}$ para Tb, mas sim $1x2^{40}$.

Obs: Kb => kilo byte, ou simplesmente K.

Um computador de 64 Kb (ou simplesmente 64 K) de RAM, por exemplo, permite armazenar 64x210 = 64x1024 = 65536 bytes (caracteres), da seguinte forma.
Cada caracter da memória é armazenado no padrão ASCII, em base binária. O código ASCII é a sigla de "American Standard Code for Information Interchange", ou seja, Código Standard Americano para Intercâmbio de Informações. Sabemos que o computador opera somente em base binária, apesar da saída ser sempre em decimal. Podemos representar caracteres alfabéticos (a,b,c,...), numéricos (0,1,2,...) ou simbólicos (+,-,*,...) usados no computador por meio de códigos binários, numerando cada um com um valor diferente. Poderíamos associar de várias formas estes caracteres a números binários, porém a padronização dessa associação é muito conveniente, pois permite a compatibilidade entre vários produtos, tanto em termos de "software", quanto de "hardware". Por esse motivo, o código ASCII é um dos mais populares. A letra "A" em ASCII é representada pelo número binário 1000001, que é 65 em decimal. Assim subseqüentemente, "B" =66, etc.
Dizer que a memória de um computador pode armazenar esses 65536 caracteres, significa dizer que estes são guardados na forma de bytes binários, através do código ASCII. O caracter da letra "A" (maiúscula), por exemplo, é guardado sob a forma 65, 01000001 em binário (note que é um byte: 8 bits).

## 7.1 - PASSAGEM DA BASE BINÁRIA PARA A DECIMAL

Um byte é uma combinação de 8 bits, que podem estar acesos ou apagados. Em Análise Combinatória, isso é um caso de Arranjo com repetição de elementos (A'), cuja função é calcular o número de combinações que se pode formar com n elementos, cada um tendo p possibilidades de estar representado. Nos números binários, a lógica é a seguinte (fig 7.4):

| 1° | 2° | 3° | 4° |
|---|---|---|---|
| | | | ·0 |
| | | ·0· | ·1 |
| | ·0· | | ·0 |
| | | ·1· | ·1 |
| 0· | | | ·0 |
| | | ·0· | ·1 |
| | ·1· | | ·0 |
| | | ·1· | ·1 |
| | | | ·0 |
| | | ·0· | ·1 |
| | ·0· | | ·0 |
| | | ·1· | ·1 |
| 1· | | | ·0 |
| | | ·0· | ·1 |
| | ·1· | | ·0 |
| | | ·1· | ·1 |

fig 7.4

O primeiro dígito tem duas possibilidades de acontecer: 0 ou 1. O segundo, tem quatro: 0 ou 1 caso o primeiro tenha sido 0, e 0 ou 1 caso o primeiro tenha sido 1. A mesma idéia é usada para o terceiro, quarto, etc.
Pela árvore de possibilidades nota-se que para o primeiro bit, com duas possibilidades de combinação, pode-se formar dois números binários diferentes: 0 ou 1. Até o segundo bit, pode-se formar 00, 01, 10 e 11, quatro números diferentes. Até o terceiro bit, forma-se 8 números distintos.
O algoritmo, então, para se calcular o número de combinações possíveis que pode se fazer com n elementos, cada um com p possibilidades de ocorrência, é dada pela lei: A' = pn. Restringindo, para o caso dos números binários tem-se p=2 (0 ou 1) e n igual à quantidade de bits do número. Exemplificando, com 10 bits existem $2^{10}$ números binários diferentes.
No caso específico do byte, concluímos que com seus 8 bits podem se formar 28 = 256 números binários distintos. É esse o motivo do código ASCII estar restringido a 256 caracteres (0 a 255). Com os 8 bits do byte tem-se 256 combinações, que permitem que a relação homem-máquina seja no mínimo inteligível.
Ampliando a árvore de posibilidades, notaremos que um byte localiza-se entre 00000000 (=0) e 11111111 (=255). Mas como podemos calcular o valor decimal de um número binário sem ter que usar a árvore?
Nos números binários cada bit tem um número que o caracteriza segundo sua posição. A numeração cardinal dos bits varia de 0 a n, a contar da direita, como já foi visto na fig 7.3. Além disso, cada bit tem um valor, que é igual a 2 elevado ao número do bit em questão. Veja: o sétimo bit é o de número 6, logo seu valor é $2^6$=64. Note que o valor do bit é quantidade de números binários distintos que se pode formar até o bit anterior. Deve-se ainda multiplicar cada valor pelo seu bit correspondente. A transformação do número binário em decimal se dá somando os resultados de todas as multiplicações feitas. Note então, que na soma usa-se apenas os valores dos bits em 1, uma vez que o 0 dos bits apagados anula seu valor.
Confira todas as informações do parágrafo acima na tabela da fig 7.5, uma ampliação da fig 7.3. Depois tente transformar o byte binário 01001101 em decimal. O resultado deve ser 77.

| ... | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | valor decimal |
|---|---|---|---|---|---|---|---|---|---|---|
| ... | 28 | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | valor na base 2 |
| ... | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | numeração |
| ... | 9° | 8° | 7° | 6° | 5° | 4° | 3° | 2° | 1° | posição do bit |
| ... | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | possibilidades |
| | | B Y T E | | | | | | | | |

fig 7.5

## 7.2 - PASSAGEM DA BASE DECIMAL PARA A BINÁRIA

Assim como é possível transformar números binários em decimais, é possível fazer o caminho de volta ou seja, transformar números decimais em binários. Para entendermos esse caminho, devemos antes entender o processo bin-dec visto antes, uma vez que dec-bin é conseqüência desse processo.
No último parágrafo do capítulo anterior, dissemos que a numeração cardinal dos bits varia de 0 a n. Vemos que cada bit é um dígito do número binário (bit é uma contração da expressão BInari digIT, dígito binário), como cada dígito do número decimal. Vimos ainda que o valor do bit é dado por $0/1x2^n$. A base 2 indica que existem dois símbolos a combinar para formar números binários. Se isso é verdade, então o valor de um dígito de um número decimal é dado por $0/1/2/3/4/5/6/7/8/9x10^n$, pois existem dez símbolos a combinar. Exemplo:

$$143 = 1.10^2 + 4.10^1 + 3.10^0 = 100 + 40 + 3.$$

Generalizando, $D.b^n$ <=> valor do (n+1)ésimo dígito D do número na base b. Ainda, conforme a tabela da fig 7.5 temos:

n° na base b:

| $b^3$ | $b^2$ | $b^1$ | $b^0$ |
|---|---|---|---|
| $D_4$ | $D_3$ | $D_2$ | $D_1$ |

o mesmo em dec:

$$d = D_4.b^3 + D_3.b^2 + D_2.b^1 + D_1.b^0$$

Expandimos d colocando b em evidência. Temos:

$$D = Db^0$$
$$Db = Db^1$$

$$d = D_1 + D_2b + D_3b^2 + D_4b^3$$

$$d = D_1 + b[D_2 + D_3b + D_4b^2]$$

$$d = D_1 + b[D_2 + b[D_3 + D_4b]]$$

O desenvolvimento acima nos mostra outra maneira de se transformar um número numa base b em decimal: multiplica-se o último dígito pela base. Soma-se o resultado ao penúltimo dígito. O novo resultado é multiplicado pela base. Assim sucessivamente até que se some o primeiro dígito.
Note que os dígitos do número na base b estão escondidos em seu correspondente decimal. A transformação do decimal para o da base consiste em tirar os dígitos do decimal, separando-os. A formação do número se dá então, con-

catenando os dígitos tirados.
Analizemos a última parte da demonstração:

$$d = D_1 + b[D_2 + b[D_3 + D_4 b]]$$

$$d = D_1 + c$$

Note que c é divisível por b:

$$\frac{d}{b} = \frac{D_1}{b} + \frac{b[D_2 + b[D_3 + D_4 b]]}{b}$$

Logo, $D_1$ é o resto da razão d/b.
Recursivamente, d passa a ser o inteiro de d/b, veja:

$$d/b = D_1/b + [D_2 + b[D_3 + D_4 b]]$$

* resto de d/b -> $D_1$

* int de d/b -> $D_2 + b[D_3 + D_4 b] = d$

Repetindo o processo até d = 0, separamos os dígitos do número na base b, escondidos no número decimal d.
Falta agora juntar os dígitos. Observe que na transformação de uma base para decimal, o número é tratado apenas matematicamente. No caminho inverso, primeiro é tratado matematicamente e depois, na concatenação, como palavra. Veja que na expansão invertemos a ordem das parcelas. O dígito D1 passou da última para a primeira posição. O penúltimo assumiu a segunda e assim sucessivamente. Veja também que na primeira divisão do processo dec-base, o primeiro dígito separado é o primeiro dígito do número na base, o segundo o segundo, etc. Na concatenação eles devem ser posicionados na mesma ordem para que não haja uma mudança no seu valor. Em outras palavras, cada novo dígito obtido é colocado à esquerda do dígito anterior.
No caso específico dos binários, o seu valor decimal é obtido através de sucessivas divisões por 2.
Com isso sabemos transformar um número numa base qualquer em decimal, e vice-versa. Nos capítulos 2.1 e 2.2 do item ANEXOS estão os fluxogramas desses dois algoritmos.

## 8 - USO DO SISTEMA BINÁRIO NOS SPRITES

Como já foi visto, o sprite é uma tela (grade) de 256 bits, ou 32 bytes. Em termos de bit, estão dispostos de modo a formar um quadrado de 16x16 pixels. Em termos de byte, estão dispostos de modo a formar duas pilhas de 16 bytes cada.
Na tela da TV, cada ponto do sprite que aparece está, como já foi dito, aceso. Os pontos em 0 não aparecem. No modo de edição ocorre uma inversão total dos bytes, de modo que o que está em 0 na tela, está em 1 no editor. Isso porém, não é problema, uma vez que ao sairmos do editor ocorre uma reinversão dos bytes, voltando a ficar em 0 o que era 0 e em 1 o que era 1. Por isso, quando no editor, devemos formar as figuras que queremos invertidas, pois ficarão como desejamos na tela (fig 8.2).
A numeração dos bytes no sprite é vertical e de cima para baixo, a contar da esquerda.
Além do editor, podemos programar um sprite pelo comando "criafigl". O formato da primitiva é <criafigl (n) (l)>, onde (n) é o número do sprite que se deseja programar e (l) é uma lista de 32 números decimais, correspondentes aos bytes do sprite.
Essa lista pode ser entendida pela análise do comando "listafig", função que envia uma lista contendo 32 elementos (bytes), onde cada byte é um número entre 0 e 255 (lógico!). Exemplificando, se ao editarmos um sprite seus 8 primeiros pontos na horizontal forem "OOOOOOOO", teremos as seguintes transformações:

```
inversão   [][][][][][][][] modo edição [] apagado [] aceso
ao trocar                              (CODIFICAÇÃO LUMINOSA)
de modo    [][][][][][][][] modo tela   [] aceso   [] apagado

            0 0 0 1 1 0 0 1           (CODIFICAÇÃO BINÁRIA)

                 2423    20
                 18+8  +  1  =   25  (CODIFICAÇÃO DECIMAL)

  [25 ...           ] -> listafig - 32 elementos (bytes)
```

fig 8.1

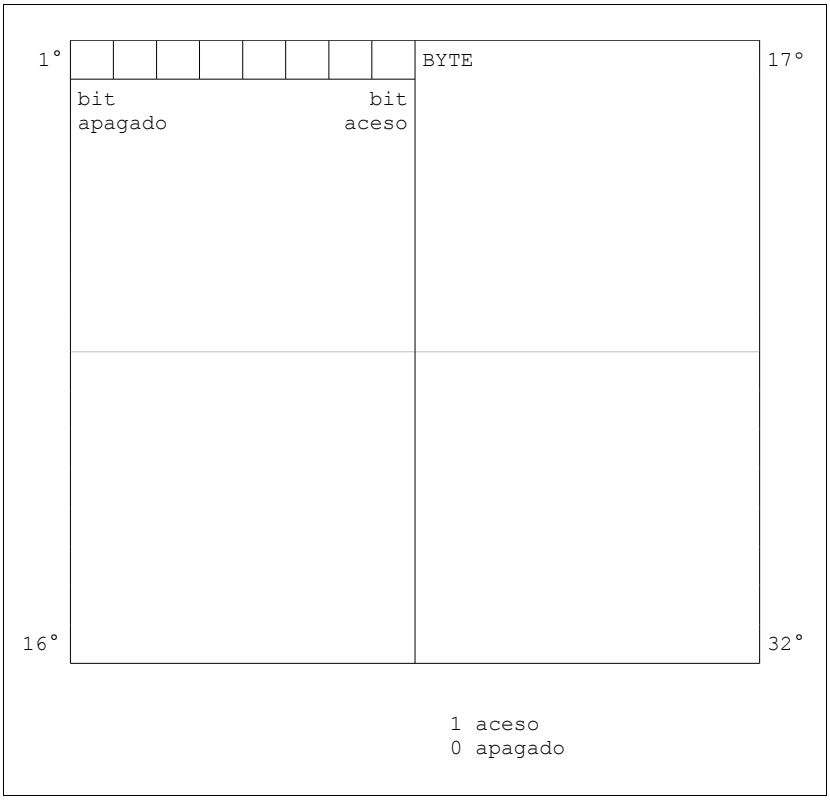



fig 8.2

9 - INTERPRETAÇÃO DE PRIMITIVAS

Não é raro usarmos comandos, quando trabalhamos com alguma linguagem de programação, sem ao menos termos uma idéia clara de como funcionam. Quando pensamos em tentar entender um comando, não é raro também nos apavorarmos, pensando que isso está acima do nosso alcance. Porém nem sempre é verdade. Se tivermos uma mínima noção do que trata certo comando, sua área de atuação, com um pouco de trabalho e boa vontade é fácil entender seu funcionamento, podendo inclusive criar procedimentos com função similar. Abaixo serão interpretadas quatro primitivas diretamente relacionadas com a edição de sprites: "edfig", "listafig", "criafigl" e "copiafig".

**"edfig"**: ao pé da letra, significa "edite figura". Como foi definido anteriormente porém, será tratado como "edite sprite". O que esse comando faz é simples: ele pega os bytes do sprite em questão, inverte-os e então entra um programa fechado, em linguagem de máquina, que edita os bytes. Nessa situação eles são colocados à nossa disposição para contato via teclado. As teclas abaixo são as únicas com alguma função no editor:

* <CTRL>+<K> (ou <CLS/HOME>) deleta a figura em 0, ou seja, apaga todos os pontos na tela (fora do editor), No editor eles aparecem brancos. Além disso, posiciona o cursor na posição HOME (canto superior esquerdo);
* <CTRL>+<Y> retorna com a figura editada inicialmente e posiciona o cursor em HOME;
* [ ][ ][ ][ ] o teclado do cursor tem as funções indicadas;
* <SPACE> a barra de espaço acende ou apaga o ponto abaixo do cursor, conforme esteja apagado ou aceso;
* <ESC> (ou <CTRL>+<]/[>) tem a função de sair do editor, retornando ao modo direto. Nesse momento a figura antiga do sprite é trocada pela nova formada;
* <CTRL>+<STOP> também sai do editor, mas a nova figura não substitui a antiga. Apresenta na tela a mensagem "Parei!".

**"listafig"**: significa "liste figura". O comando envia uma lista de 32 números decimais inteiros entre 0 e 255, que pode ser armazenada numa variável para uso imediato ou futuro. Cada número equivale ao byte correspondente do sprite;

**"criafigl"**: por extenso seria "crie figura a partir da lista". O que esse comando faz é transformar de decimal para binário todos os elementos de uma listafig, transportando cada um para suas respectivas posições no sprites, recriando a figura armazenada;

**"copiafig"**: significa "copie figura". O que ele faz é copiar o conteúdo do sprite determinado (a figura) para outro, também determinado. A rigor, este comando não precisaria existir como primitiva, uma vez que pode ser facilmente definido num procedimento. Simplesmente manda-se criar uma figura com a listafig de outra.

Agora, uma breve análise das citadas primitivas:

**"edfig"**: é um comando deveras útil, mas também substituível. Pode ser criado um procedimento de função similar, que apesar de mais lento pode oferecer muito mais em termos de versatilidade. Mais adiante ser apresentada idéia do procedimento, que ainda não está pronto;

**"listafig"** e **"criafigl"** são insubstituíveis a primeira vista. Fora do editor são a única forma de se acessar e introduzir dados no sprite. Talvez haja uma forma de criar procedimentos análogos em máquina, mas isso foge ao tema do projeto. Fica como sugestão;

**"copiafig"**: como foi visto, pode ser substituído.

Vimos o que existe. Vejamos agora o que falta e o que pode ser criado. Abaixo temos a seqüência de alguns questionamentos que deram origem ao projeto:

1) É possível ampliar os métodos de edição e programação de sprites em LOGO?
2) Se é possível, porque o faríamos?
3) Qual o melhor método de trabalho a ser adotado?
4) Onde poderíamos encontrar apoio bibliográfico?
5) O que desenvolver, para facilitar essa programação?
6) Que dificuldades encontraríamos pelo caminho?
7) Então, o que incrementar na linguagem LOGO para permitir a criação dos manipuladores de sprites?
8) Finalmente, depois de desenvolvido o projeto, qual o seu destino?

Analizando o que falta e porquê faz falta, cheguei às seguintes conclusões:
- No editor é muito difícil passar uma figura de uma posição para outra dentro do sprite. Para movimentá-la é necessário copiá-la ponto a ponto no local exato. Seria interessante se houvesse um método para ROLAR (scroll) a figura no sprite;
- Caso quiséssemos inverter uma figura para que ela aparecesse na tela como no editor (ou vice-versa), seria necessário invertê-la ponto a ponto no editor. É preciso um comando para INVERTER a figura;
- As teclas <CTRL>+<K> no edito deletam apenas em negativo (0). Seria interessante um comando para deletar em positivo (1);
- Uma boa idéia seria um programa que fizesse a figura rotar no sprite.Assim seria possível trocar todas as tartarugas que vestem a TARTARUGA por setas, por exemplo, a partir de uma única. As outras seriam obtidas por rotações da seta matriz;
- E outras, como espelhamento, interferência, redução e ampliação. Todos esses procedimentos podem ser criados, com um pouco de pesquisa em cima da Linguagem. Já foram criadas as funções para rolar, deletar, inverter e programar o sprite via "listafig", em binário ou decimal. Imaginou-se os processos para espelhar, editar, interferir e rotar. Os maiores problemas são ampliar e reduzir. A estrutura desses procedimentos pode ser encontrada na listagem do programa, no item ANEXOS.

## 10 - ESTRUTURAS LÓGICAS UTILIZADAS

Apesar do tema inovador, a estrutura básica do programa firma-se quase unicamente em cima de conhecimentos pré-adquiridos com outros projetos. Ao dizer isso, estamos entrando num capítulo técnico do projeto e é aconselhável para a melhor compreensão dos procedimentos que a listagem dos programas acompanhe a leitura.
A estrutura que concatena os menus foi aproveitada de um projeto anterior, ou por que não dizer, paralelo: o SISTEMA.
Iniciado em 90, o SISTEMA é um projeto que reúne em seus menus todas as primitivas do LOGO que atuam em programas, variáveis, propriedades, observadores e arquivos. Devido ao tema permite uma fantástica possibilidade de generalização. Isso significa poder fazer o máximo com o mínimo. Nele foi desenvolvido o esquema de menu genérico, escolha genérica (depois substituída pela seta, de escolha também genérica) e execução genérica de primitivas. Com isso, de 60 blocos o programa passou a ocupar 14, realizando mais operações do que antes.
O menu genérico funciona baseado nas três entradas: a coluna da margem esquerda, uma lista na qual são colocadas as opções do menu, e o título do menu. Primeiramente o título é centralizado na tela, na linha 0. Depois o menu é apresentado. Centraliza-se verticalmente o menu na tela e finalmente são apresentadas algumas mensagens (Veja programa "m", em Anexo 3).
Pelo programa da seta a escolha é feita movendo-se o caracter 0 (->) com as teclas [ ] e [ ] do cursor. Pressionada a tecla <RETURN>, o programa usa a linha em que está a seta para calcular a posição do ítem do menu na lista de entrada do menu genérico. A cada elemento da lista corresponde um elemento noutra lista (préviamente definida), que é o nome do programa que deve ser executado caso escolha-se aquela opção.Com o comando "faça" o programa é executado.
Agora já há uma estrutura de vídeo reverso que desenvolvi para substituir a seta. A principal vantagem é que permite uma total demarcação da opção pela barra, facilitando a leitura da linha. Além disso, não deixa de ser mais interessante visualmente.
A execução genérica de primitivas é especialmente simples: uma das entradas do programa define se a primitiva precisa ou não de alguma entrada e outra define a primitiva. Caso a primitiva precise de entrada, é apresentada uma linha de entrada na qual será colocada a entrada da primitiva. Depois, ou caso o comando não precise de entrada, é executada a primitiva.
Além desses recursos básicos, pois sem eles a memória do computador certamente não poderia conter o projeto (que assim mesmo já ocupa 63 blocos), foram usadas várias rotinas em máquina, operadores (funções que utilizam a primitiva "envie") e subprocedimentos. Ainda durante o desenvolvimento do projeto, sentiu-se necessidade de algumas rotinas para ajudar a realizar funções como rolar e espelhar: var, bd, db e dlt. Foram criadas com o objetivo de agirem como ferramentas que facilitassem a formação do programa maior: o FIGSTAR.
Abaixo estão listados os suprocedimentos mais importantes, classificados segundo sua estruturação, juntamente com sua utilidade:

* rotinas em máquina:
    * aum: aumenta duas vezes as dimensões da figura da TAT se inicialmente ela estiver no estado normal;
    * dim: se a figura da TAT tiver sido préviamente aumentada, esta rotina lhe devolve o tamanho original;

* operadores:
    * éimpar: envia verd se é e falso se não;
    * éinteiro: envia verd se é e falso se não;
    * potência: realiza potenciação;
    * var: envia mensagem de cor e número da figura da TAT;
    * bd: transforma binário em decimal;
    * db: transforma decimal em binário;
    * dlt: divide uma lista ao meio;
    * invert: inverte a ordem dos elementos de uma lista;

* subprocedimentos:
    * ctl2: centraliza uma palavra;

* f: indica a figura da TARTARUGA;
* c: indica a cor da figura da TARTARUGA;
* .: muda a posição do cursor;
* ,: muda a posição da TARTARUGA.

Desses, apenas dois merecem uma atenção especial: db e bd. Significam respectivamente decimal-binário e binário-decimal e realizam as transformações indicadas. Os capítulos 7.1 e 7.2 do capítulo 7 explicam como funcionam essas rotinas. Acompanhe à sua leitura os fluxogramas 2.1 e 2.2 do item ANEXOS.

## 11 - FIGSTAR: MANIPULADOR DE SPRITES

### 11.1 - ORGANOGRAMA

O menu principal reúne quatro campos de controle: edição, figura, variável e arquivos. A opção escolhida pelo usuário irá agir sempre sobre a figura do sprite em questão, cuja imagem e número podem ser visualizados nos dois lados do menu.
Cada campo ramifica-se segundo o esquema abaixo:

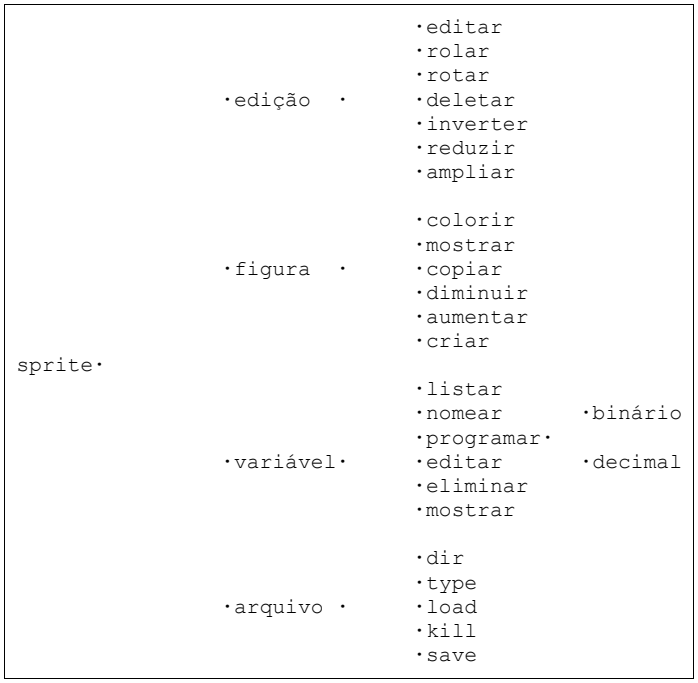



fig 11.1

### 11.2 - CONTROLE DE EDIÇÃO

Este menu reúne algumas operações com as quais pode-se atuar diretamente sobre a figura do sprite, mudando sua forma, tamanho ou posição no mesmo.

A opção EDITAR coloca o sprite no modo de edição, através da primitiva "edfig". Como já foi explicada e analizada no capítulo 9, agora será explicada a idéia de como criar um procedimento em LOGO com função similar, que não pôde se concluído.
Existem dois tipos de editor: o que utiliza pontos luminosos para representar os bit acesos do sprite e o que utiliza o sistema binário. Nesse, os pontos em 1 são representados pelo número 1 e os pontos em 0 pelo número 0.
A idéia é inicialmente definir a dimensão do sprite, que pode variar de 1x1 a 180x180. No caso da dimensão máxima do sprite, os pontos não seriam ampliados, e programar esse sprite seria praticamente programar uma tela. A medida que a dimensão fosse diminuindo, o tamanho do ponto que representaria o bit aumentaria gradualmente.
No momento de edição, os bytes do sprite seriam transformados em bytes binários que uma rotina leria bit a bit. Caso a edição fosse em pontos, os bits em 1 seriam representados por um quadrado na tela e os bits em 0, não teriam representação visível. Caso o editor fosse binário, os bits seriam simplesmente passados para a tela.
Para o editor luminoso há dois tipos de cursor: um vazio, que envolve o ponto e outro cheio, que pisca sobre ele. Para o editor binário pode-se escolher entre o cursor de vídeo reverso, que faz com que a imagem do número sob ele apareça invertida e o cursor que pisca sobre o bit.
Para manipular os bits, a idéia é relacionar o nome da variável que contém o bit à posição atual do cursor. Assim, caso apagássemos o ponto, a rotina trocaria o conteúdo da variável cujo nome fosse a posição do cursor.
Além disso, poderíamos incluir nesse editor mais teclas de função, para inverter, rolar e deletar. Há ainda a possibilidade de recriar a figura a cada mudança de bit. Assim, quando acendêssemos um ponto no editor, acender-se-ia o ponto correspondente na figura do sprite da TARTARUGA.
Certamente que a lentidão desse editor tornaria a rotina desagradável de ser usada, contudo está longe de ser uma idéia inviável. Caso houvesse um compilador para programas em LOGO, provavelmente se tornaria tão ágil quanto o editor da linguagem.

A opção ROLAR (scroll) permite que a figura mude de posição no sprite. Com o teclado do cursor, comandamos o rolamento. Quando a figura chega nas extremidades do sprite, automaticamente essa parte da figura passa para a extremidade oposta do sprite. Os procedimentos que realizam essa função são role (principal), scr1 (cima), scr2 (baixo), scr3 (esquerda), scr4 (direita) e scro (teste).

A opção ROTAR também não pôde ser concluída. Ela permitiria que a figura girasse no sprite, como se gira a mão no ar. A idéia que se tem é a seguinte:Transforma-se os 32 bytes do sprite em binário. Uma rotina lê bit a bit e quando ler 1, junta no fim do texto de um procedimento especial, a ordem [ul pf 1]. Quando ler 0, junta a ordem [un pf 1]. Após ter lido os oito bits que formam o primeiro byte, juntaria a ordem [un pe 90 pt 1 pd 90 pt 8], e começaria a leitura do seguinte byte, reiniciando o processo. Quando terminasse a leitura do 16° byte, juntaria a ordem [un pe 90 pf 16 pd 90], e reiniciaria a seqüência. Ao terminar de ler os 256 bits do sprite, teríamos um programa que reproduziria com "pf" e "pd" toda a figura do sprite. Com isso, outra rotina desenharia a figura com a rotação desejada. Por exemplo, com uma rotação de 90 graus, primeiro a TARTARUGA daria um [pd 90] e desenharia a figura. A TARTARUGA então se posicionaria sobre o desenho e com o comando "inverta", assumiria a figura.
Esse processo já foi criado por mim, porém não foi possível incorporá-lo ao projeto, pois ainda está imperfeito. É lento, mas permite rotações de grau ínfimo, coisa que nenhum outro algoritmo permite fazer. Na melhor das hipóteses, os algoritmos dos editores comuns, em BASIC ou Máquina, permitem rotações de 90 graus.

A opção DELETAR deleta a figura do sprite em 1. Na tela a figura aparece como um quadrado cheio e no editor não aparecem os pontos, devido à inversão que se dá automaticamente.

A opção INVERTER inverte o estado dos bits. O que está em 1 passa a estar em 0 e vice-versa. Ao contrário da inversão do editor, essa se mantém.

As rotinas das opções REDUZIR e AMPLIAR ainda não foram criadas, nem se possui uma idéia clara de como fazê-las. Elas serviriam para que a figura fizesse exatamente o que o nome diz. Um "P" pequeno por exemplo, poderia ser ampliado até que sumisse, saindo do sprite. Com o inverso, poderia se reduzir até virar um ponto.

## 11.3 - CONTROLE DE FIGURA

Este menu atua sobre a aparência da TARTARUGA, SPRITE e FIGURA.
A opção COLORIR permite mudar a cor da figura. A relação abaixo indica as teclas com alguma função:

- [ ] incrementa em 1 o n° da cor da figura;
- [ ] decrementa em 1 o n° da cor da figura;
- <ESC> volta ao menu.

A opção MOSTRAR permite mostrar a figura de cada sprite.
Abaixo estão as teclas de função:

- Barra de espaço: edita a figura em questão;
- [ ] realiza a função "aum";
- [ ] realiza a função "dim";
- [ ] incrementa em 1 o n° da figura em questão;- [ ] decrementa em 1 o n° da figura em questão;
- <ESC> volta ao menu.

A opção COPIAR ainda não foi concluída. Ela servirá para que se possa copiar o conteúdo de um sprite para outro.

As opções DIMINUIR e AUMENTAR realizam respectivamente as funções "dim" e "aum".

A opção CRIAR servirá para que se possa passar o conteúdo de um sprite guardado em uma variável para a figura em questão, criando uma nova. Apesar de não apresentar dificuldade, também não está concluída.

## 11.4 - CONTROLE DE VARIÁVEL

Este menu reúne todas as funções que manipulam as variáveis de figuras.

LISTAR é a opção que mostra a listafig da figura em questão, a já citada lista de 32 bytes decimais.

NOMEAR é a opção que atribui a um nome que o usuário define, a listagem da figura em questão (ainda não concluída).

PROGRAMAR é uma rotina interessante. Ela permite que se programe uma figura através do comando "listafig". Os bytes são programados em base decimal ou binária. Caso se deseje programar em binário, cada um dos 32 bytes da figura em questão será programado por um byte binário, que é digitado na linha de entrada que se apresenta. Com isso, pode-se fazer a figura que se quer em papel quadriculado e copiar direto para a figura da tela. Os pontos acesos representados no papel são 1 em binário, e os apagados são 0. Assim que se tecla <RETURN>, a linha é interpretada e os bytes binários transformados em decimal. Caso se tenha escolhido programar em decimal, essa transformação não é feita.
Cada byte que o programa recebe é passado para a figura, que se muda instantaneamente. O cursor que acompanha serve para que o programador tenha noção de qual é o byte que está programando.
Há ainda alguns comandos, que estão listados abaixo:

- s  : permite sair do modo de programação e voltar ao menu;
- v  : volta um byte atrás, para corrigí-lo;
- p  : pula o próximo byte, para não mudá-lo;
- e  : espelha o byte simétrico;
- ca : copia o byte anterior;

- cs : copia o byte seguinte.
Esses comandos devem ser digitados no lugar do byte.

A opção EDITAR coloca todas as variáveis que existem em edição, para que se possa manipulá-las diretamente. ELIMINAR acaba com todas as variáveis.

MOSTRAR exibe todas as variáveis.

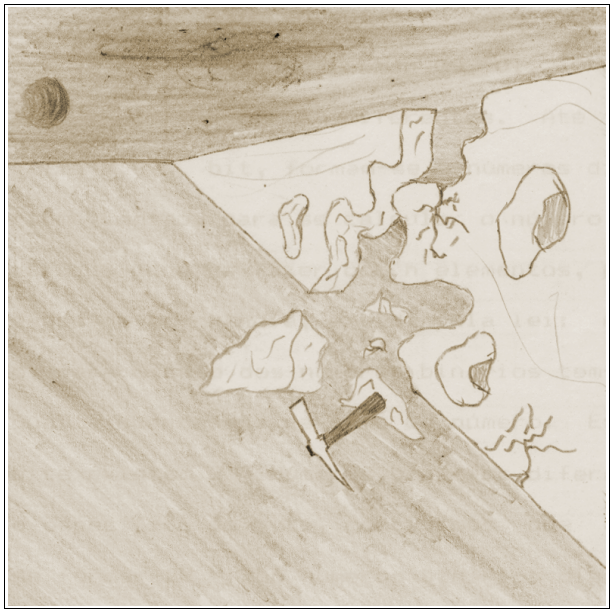
## 11.5 - CONTROLE DE ARQUIVOS

Abaixo estão relacionadas as opções e suas funções:

- DIR : mostra os arquivos de disco;
- TYPE : mostra o conteúdo de um arquivo;
- LOAD : carrega do disco um arquivo;
- KILL : elimina do disco um arquivo;
- SAVE : grava em disco as variáveis da memória.

# 12 - FILOSOFIA DE TRABALHO: O SISTEMA LOGO

Podemos comparar o LOGO com uma ferramenta com a qual abrimos uma parede, o conhecimento de programação. No início, antes de conhecermos a Linguagem, temos a nossa frente uma parede que impede que a luz do sol nos ilumine. Tateando no escuro descobrimos um orifício que nos permite ver ao longe o sol, brilhando com toda a sua imponência. Por esse orifício passa um filete de luz, que mostra a um canto uma ferramenta: o LOGO, a picareta com a qual comandaremos o trabalho de ampliar o buraco da parede. Assim, é pelo buraco, a parte gráfica do LOGO, que começamos o trabalho. Seria bem mais difícil começar um novo buraco, sem saber o que haveria atrás da parede.
Os comandos "pf", "pd", "pt" e "pe" são nossa primeira relação com a Língua e com ela começamos a raspar a parede com a ponta da picareta. Então, num dado momento, após refletir no que estamos fazendo, encontramos uma lógica na seqüência [pf 10 pd 90] [pf 10 pd 90] [pf 10 pd 90] [pf 10 pd 90] que faz um quadrado. Com isso descobrimos que usando o comando "repita" nosso trabalho fica menor e mais rápido. E começamos a dar "porradas" na parede com a picareta, ao invés de apenas raspá-la. O nosso buraco se torna cada vez maior e cada vez mais se abrem nossos horizontes. A medida que vamos avançando na programação vamos encontrando partes mais sólidas da parede, e abandonamos o buraco, começando outro. Repetimos isso sempre que necessário. No fim, a parede já está tão esburacada que seu próprio peso é suficiente para que uma parte de si desmorone. O buraco se alarga e cada vez mais fácil se torna passar por ele, penetrar nos mistérios do outro lado.
Quando o buraco já está largo o suficiente para que passemos por ele, já não nos contentamos com apenas isso. Queremos agora destruir a barreira que nos impede de pular livremente de uma lado para o outro. A programação se torna cada vez mais rápida e fácil. Usamos "ap", "ed", "envie". A parede cai gradualmente. No outro lado encontramos novos materiais. O LOGO não admite viseiras nem prende ninguém. A picareta é UM dos meios, não O meio. Aí mesclamos LOGO, BASIC e Linguagem de Máquina. Entra o WRITE, que como uma britadeira arrasa a parede. As dificuldades tornam-se menores, mas o trabalho aumenta. Surge o GRAPHOS, HELLO, SISTEMA OPERACIONAL, verdadeiras dinamites que arrombam a parede.
Porém ela é imensa, ela se perde num ponto ao infinito fundo e toma características dessa distância. Por isso, por mais que trabalhemos, sempre encontramos mais. Imaginem um lenhador numa floresta. Inicialmente ele precisa derrubar apenas uma árvore. Depois, caso deseje aumentar todo o perímetro da clareira, deverá derrubar duas ou três. No dia seguinte seriam necessárias dez ou mais. Cada vez ele teria mais trabalho para completar o mesmo ato do dia anterior. Porém, cada dia apresentaria mais prática e tiraria mais lenha.
Assim é o conhecimento: infinito. Por mais recursos que usemos, por mais que saibamos programar, nunca se esgotam as possibilidades, combinações, idéias. E o trabalho de avançar na programação é nosso. Cada um de nós deve quebrar a sua própria parede. Não adianta deixar o trabalho para os outros. A força exata para esse fim é propriedade de cada um. Os outros a tem a mais, ou a menos.



13 - O PROCESSO: DIFICULDADES, APERFEIÇOAMENTO, SOLUÇÕES

O processo de desenvolvimento de um projeto inovador, mais do que o de qualquer outro, é lento, trabalhoso e não raro cansativo. Aparecem inúmeras dificuldades. Algumas parecem tão impossíveis de serem solucionadas que realmente, quase nos fazem desistir. Nesse momento é hora de parar e de repensar o que estamos fazendo.
O melhor método sem dúvida (e não me refiro apenas à informática), é começar analizando o que se quer, os objetivos. Parte-se então para o que se tem, os fatos, e finalmente, define-se clara e suscintamente o que falta. Traça-se daí, caminhos a serem seguidos (agora sim é informática), que nem sempre precisam seguir a linha do projeto. Basta que enquadrem o assunto em que se tem alguma dificuldade. O trabalho nem precisa ser feito na mesma linguagem. Um exemplo é o procedimento usado para programar sprites pelo comando "listafig". Foram necessários três meses de reflexão sobre o assunto para que a idéia se esclarecesse na minha cabeça. Por isso prefiro acreditar que por mais que demore para a solução aparecer, ela existe, apenas minha cabeça ainda não est pronta para vê-la. Afinal, há uma hora e um momento certo para tudo na vida.
Portanto, geralmente não é do meu interesse imediato concluir projetos. Interessa-me muito mais as idéias que um projeto desenvolve. É certo que quanto mais um assunto for ampliado, mais possibilidades haverá de ação. Cada projeto avança em um determinado campo. Acontece que esses campos têm muitos pontos em comum, que podem não ser o tema central do assunto, mas podem ser usados como ferramenta nesse assunto. Por isso não vejo como grande problema ter vários projetos inacabados. Cada vez que exploro um, descubro algo novo. Com o tempo todos vão se concluindo, paralelamente.
Nesse projeto, a principal fonte de aperfeiçoamento dos procedimentos que apresentavam alguma falha foi sua reestruturação em fluxogramas. Além disso grande parte das dificuldades se deu por falta de comandos no LOGO que realizassem funções como as dos operadores apresentados no capítulo 10. Nessas situações, através de pesquizas e debates com colegas, chegou-se à solução desejada em 100% dos casos.

## CONCLUSÃO

É possível concluir um trabalho como esse? Penso que não. Fazer isso seria afirmar que já se entrou, quando na verdade ainda se está procurando a chave.
Mas dessa fase inicial, podemos tirar pelo menos duas conclusões importantes:
Primeiro: é realmente possível ampliar os métodos de programação de sprites em LOGO. Se os processos obtidos são mais lentos do que os de outras linguagens, isso não diminui o valor do LOGO. O mérito de uma linguagem não precisa estar relacionado com a quantidade de informações que dá pronta para o usuário ou programador. Em muitos casos, é mais valiosa a linguagem que desafia a pessoa a raciocinar, pensar, imaginar. Isso torna a pessoa criativa e versátil, torna-a capaz de enfrentar as mais diferentes situações, encontrando com seus próprios recursos as soluções dos seus problemas. Talvez os comandos do LOGO por si só, não façam tudo o que os de outras linguagens fazem, mas ofereçem condições para que VOCÊ faça o resto.
Segundo: a pesquisa é o método por excelência da boa aprendizagem. O assunto desse trabalho não está esgotado e nem será facilmente esquecido. Idéias evoluiram nesse período e muito se pode fazer, tendo esse projeto como base.
Gostaria também de deixar registrado que, a meu ver, o LOGO é uma linguagem que promete. Com tempo e afinco é possível alcançar tudo o que se aspira em relação a ela. No início, concretizar esse projeto parecia inconcebível e no entanto, aqui está ele. O fato de ser lento e em pequena escala não é culpa do LOGO, mas da máquina e da estrutura técnica da versão que foi utilizada. Tudo o que falta para o LOGO se tornar uma das linguagens mais poderosas disponíveis é bastante pesquisa técnica em cima dele, assim como se fez com o BASIC, COBOL e tantas outras linguagens.
Como sugestão para próximas fases deixo as operações lógicas com números binários (and, or, not, xor), que podem ter interessantes aplicações sobre o bytes dos sprites. Junto a isso a notação fracionária dos binários, o desenvolvimento de comandos semelhantes a "edfig", "listafig" e "criafigl" e a conclusão do que não foi alcançado neste projeto, como rotação, ampliação, redução, espelhamento e interferência de sprites.
E ainda uma sensação final: foi gratificante desenvolver o projeto.

## ANEXOS

1 - IMAGEM DE ALGUMAS TELAS DO PROGRAMA

2 - FLUXOGRAMAS

3 - LISTAGEM DO PROGRAMA

3.1 - Primeira versão do projeto

TÍTULOS

```
aprenda , :X :Y
aprenda . :C :L
aprenda a
aprenda apague
aprenda atrifig :n° :nome
aprenda ctlz :L :list
aprenda dados
aprenda dados1.0
aprenda dados1.1
aprenda dados1.2
aprenda dados1.3
aprenda dados1.4
aprenda del :type :nfig
aprenda esco1.0
aprenda esco1.1
aprenda esco1.2
aprenda esco1.3
aprenda esco1.4
aprenda giga
aprenda grave :arquivo
aprenda help
aprenda input :var :string
aprenda inv :nfig
aprenda inv2 :nfig
aprenda kilo


PROCEDIMENTOS


ap , :X :Y
   mudepos ( lista :X :Y )
fim

ap . :C :L
   mudecursor ( lista :C :L )
fim

ap a
   tat
   rg
   dt
   kilo
   mudecf 1
fim

ap apague
   tat
   eltudo
   liberememfim

ap atrifig :n° :nome
   att
   at
   mudefig :n°
   edfig   :n°
   atr :nome listafig :n°
fim

ap ctlz :L :list
   atr "elementos 0
   atr "n°        1
   repita nel :list [atr "sublist   nel elemento :n° :list
                     atr "elementos :elementos + :sublist
                     se :n° = ult :list [pare]
                                        [atr "n° :n° + 1]]
   atr "elementos :elementos + nel :list - 1
   se  :elementos > 27 [pare]
   atr "C int ( 29 - :elementos ) / 2
   mudecursor ( sn :C :L ) esc :list
fim

ap dados
   atr "c 7
   atr "l 9
   atr "esco [[] [] [] [] [] []]
   atr "menu [[] [] [] [] [] []]
fim

ap dados1.0
   atr "c 7
   atr "l 9
   atr "esco    [? [] 1 2 3 4]
   atr "menu    [HELP [] EDITOR FIGURAS VARIÁVEIS ARQUIVOS]
   atr "titulo1 [CONTROLADOR]
   atr "titulo2 [D E]
```

```
   atr "titulo3 [SPRITES]
fim

ap dados1.1
   atr "c 7
   atr "l 9
   atr "esco    [1 2 3 4 5 6]
   atr "menu    [EDITAR ROLAR DELETAR AMPLIAR REDUZIR INVERTER]
   atr "titulo3 [EDIÇÃO]
fim

ap dados1.2
   atr "c 7
   atr "l 9
   atr "esco    [1 2 3 4 5 6]
   atr "menu    [COLORIR MOSTRAR REPRODUZIR AUMENTAR DIMINUIR CRIAR]
   atr "titulo3 [FIGURAS]
fim

ap dados1.3
   atr "c 7
   atr "l 9   atr "esco    [1 2 3 4 5 6]
   atr "menu    [LISTAR NOMEAR PROGRAMAR EDITAR ELIMINAR MOSTRAR
   atr "titulo3 [VARIÁVEIS]
fim

ap dados1.4
   atr "c 7
   atr "l 9
   atr "esco    [1 2 3 4 5]
   atr "menu    [MOARQ ARQS ELARQ LOAD SAVE]
   atr "titulo3 [ARQUIVOS]
fim

ap del :type :nfig
   atr "limpo :type
   repita 31 [atr "limpo ( sn :type :limpo )]
   criafigl :nfig :limpo
fim

ap esco1.0
   atr "x ascii care
   se  :x = 27 [att nívelinicial]
   se  :x = 49 [p1.1]
   se  :x = 50 [p1.2]
   se  :x = 51 [p1.3]
   se  :x = 52 [p1.4]
   se  :x = 63 [help]
   esco1.0
fim

ap esco1.1
   atr "x ascii care
   se  :x = 27 [p1.0]
   se  :x = 49 [editar]
   se  :x = 50 [rolar]
   se  :x = 51 [deletar]
   se  :x = 52 [ampliar]
   se  :x = 53 [reduzir]
   se  :x = 54 [inverter]
   esco1.0
fim

ap esco1.2
   atr "x ascii care
   se  :x = 27 [p1.0]
   se  :x = 49 [colorir]
   se  :x = 50 [mostrar]
   se  :x = 51 [reproduzir]
   se  :x = 52 [aumentar]
   se  :x = 53 [diminuir]
   se  :x = 54 [criar]
   esco1.0
fim

ap esco1.3
   atr "x ascii care
   se  :x = 27 [p1.0]
   se  :x = 49 [listar]
   se  :x = 50 [nomear]   se  :x = 51 [programar]
   se  :x = 52 [editar]
   se  :x = 53 [eliminar]
   se  :x = 54 [mostrar]
   esco1.0
fim
```

```
ap esco1.4
   atr "x ascii care
   se  :x = 27 [p1.0]
   se  :x = 49 [moarq]
   se  :x = 50 [arqs]
   se  :x = 51 [elarq]
   se  :x = 52 [load]
   se  :x = 53 [save]
   esco1.0
fim

ap giga
   .deposite 62432 227
   .chame 68
fim

ap grave :arq
   eliminearq :arq
   gravetudo  :arq
fim

ap help
fim

ap input :var :string
   ( ponha :string )
   atr :var ( pri line )
fim

ap inv :nfig
   criafigl :nfig tp listafig :nfig
fim

ap inv2 :nfig
   atr "var listafig :nfig
   repita 32 [atr "byte 255 - ( pri :var )
              atr "var  ( jf :byte ( sp :var ) )]
   criafigl :nfig :var
fim

ap kilo
   .deposite 62432 226
   .chame 68
fim
```

3.2 - Última versão do projeto

TÍTULOS

```
aprenda , :x :y
aprenda . :c :l
aprenda a
aprenda aum
aprenda b
aprenda bd :bin
aprenda c
aprenda can
aprenda cap
aprenda cop :u
aprenda copy
aprenda cor
aprenda cri
aprenda cse
aprenda ctl2 :l :e
aprenda db :a :n
aprenda del :fig
aprenda dim
aprenda dlt :l1 :l2
aprenda eimpar :m
aprenda esp
aprenda f
aprenda fyg
aprenda grave
aprenda inv :l
aprenda invert :ls
aprenda k :t :c :m :f :v
aprenda l :v :c :l :i :z
aprenda liste :fig
aprenda m :c :l :t
aprenda most
aprenda n
```

```
aprenda o :p :m
aprenda o0
aprenda o1
aprenda o2
aprenda o3
aprenda o4
aprenda p1
aprenda p2
aprenda p3
aprenda pdb
aprenda pow :b :e
aprenda pro
aprenda pul
aprenda q
aprenda role :fig
aprenda sai
aprenda scr1
aprenda scr2aprenda scr3 :1 :2
aprenda scr4 :1 :2
aprenda scro
aprenda tudo
aprenda var
aprenda vol
aprenda z :z
aprenda éint :m


PROCEDIMENTOS


ap , :x :y
   mudepos sn :x :y
fim

ap . :c :l
   mudecursor sn int :c int :l
fim

ap a
fim

ap aum
   q
   dt
   .deposite 62432 227
   .chame 68
   sótat 0 [, -102 8]
   sótat 1 [,   83 8]
   at
fim

ap b
   . 5 19 ponha pal var "\  :n "°\ BYTE\
fim

ap bd :bin
   se évazia  :bin [envie 0]
   se 0 = pri :bin [envie bd sp :bin]
   se 1 = pri :bin [envie ( pow 2 -1 + nel :bin ) + bd sp :bin]
                   [envie -300]
fim

ap c
   . 12 17 ( ponha ".. var 0 cortat ".. )
fim

ap can
   atr "des sn :des ult :des
   atr "fon sp :fon
   atr "y   :y - 1,8
   atr "n   :n + 1
fim

ap cap
   .  8  0 esc   [VII \ MOSTRA DE
   .  6  1 esc   [TRABALHOS PRÁTICOS   .  6  3 esc   [Colégio Evangélico
   .  8  4 esc   [Alberto Torres
   ctl2  7       "ÁREA\ DE\ INFORMÁTICA
   ctl2 10       "\>\>\>\ FIGSTAR\ \<\<\<\<
   .  3 13 esc   [REALIZAÇÃO \ PROGRAMÁTICA
   ctl2 15       "\*\ Paulo\ Roberto\ Bagatini\ \*
   .  2 18 ponha [3° Ano A do 2° Grau Diurno
   ctl2 20       "\>\>\>\ Informática\ \<\<\<\<
   .  3 22 esc   [Lajeado, outubro de 1991
fim
```

```
ap cop :u
   mudedç 0
   mudecl 15
   un
   , -31 60 faça sn :u []
   ,  26 60
   ,  26 43
   , -31 43
   pf 17 pd 90
   pf 19 pd 90
   pf 17 pe 90
   pf 19 pe 90
   pf 17 pd 90
   pf 19 pd 90
   pf 17
   un
   mudecl :c
   mudefig :f
   , -40 48
   repita 3 [un
             , coorx + 19 52 faça sn :u []
             carimbe]
fim

ap copy
   atr "c cortat
   atr "f fig
   atat 2
   dt
   cop "ul
   atr "x care
   cop "ub
   att
   atat [0 1]
   o2
fim

ap cor
   c
   atr "c ascii care
   se e cortat =  0 :c = 29 [mudect 15 cor]
   se e cortat = 15 :c = 28 [mudect  0 cor]
   se               :c = 27 [o2]
   se               :c = 28 [mudect cortat + 1]
   se               :c = 29 [mudect cortat - 1]
   cor
fim
ap cri
   atr "des sn :des :bt
   atr "fon sp :fon
   atr "y   :y - 1,8
   atr "n   :n + 1
fim

ap cse
   atr "des sn :des pri sp :fon
   atr "fon sp :fon
   atr "y   :y - 1,8
   atr "n   :n + 1
fim

ap ctl2 :l :e
   . ( 30 - nel :e ) / 2 :l ponha :e
fim

ap db :a :n
   se :a < 1 [envie :b]
   se :n = 0 [atr "b pal :b 0 envie db :a / 2 :n]
   se e :a * 2 > :n :a - 1 < :n
    [atr "b pal :b 1 envie db :a / 2 :n - :a]
    [atr "b pal :b 0 envie db :a / 2 :n]
fim

ap del :fig
   criafigl :fig [255 255 255 255 255 255 255 255
                  255 255 255 255 255 255 255 255
                  255 255 255 255 255 255 255 255
                  255 255 255 255 255 255 255 255]
fim

 ap dim
   atat [0 1]
   dt
   .deposite 62432 226
   .chame 68
   sótat 0 [, -94 1]
```

```
   sótat 1 [,  91 1]
   at
fim

ap dlt :l1 :l2
   se ( nel :l1 ) = nel :l2 [envie lista :l1 :l2]
                            [envie dlt sn :l1 pri :l2 sp :l2]
fim

ap eimpar :m
   se 0 = resto :m 2 [envie falso]
                     [envie  verd]
fim

ap esp
   atr "b "
   se :n < 17
    [atr "des sn :des bd invert db 128 elemento 17 :fon]
    [atr "des sn :des bd invert db 128 elemento :n - 16 :des]
   atr "fon sp :fon   atr "y   :y - 1,8
   atr "n   :n + 1
fim

ap f
   .  2 15 ponha pal ". var 0 fig
   . 25 15 ponha pal ". var 0 fig
fim

ap fyg
   atat [0 1]
   criafigl fig sn :des :fon pdb
fim

ap grave
   elns
   eliminearq "figs
   gravetudo  "figs
fim

ap inv :l
   se :l = [] [envie []]
              [envie sn 255 - pri :l inv sp :l]
fim

ap invert :ls
   se évazia :ls [envie "]
                 [envie pal ult :ls invert su :ls]
fim

ap k :t :c :m :f :v
   atr "l ( 22 - nel :m ) / 2
   at
   m :c :l :t
   . 9 19 esc [\<ESC\> VOLTA]
   l :v :c + 1 :l :l :l - 1 + nel :m
fim

ap l :v :c :l :i :z
   . :c :l ponha car 0
   .chame 342
   atr :v ascii care
   se 27 = conteúdo :v [se :v = "a [tudo]
                                  [o0  ]]
   se 13 = conteúdo :v
    [atr "n :l - :i + 1 atr :v :n
     se :v = "g [faça elemento :n :f]
                [se :v = "o [faça sn "o elemento :n :f]
                            [faça sn pal "o :a []]]]
    . :c :l ponha "\
   se 30 = conteúdo :v [se :l = :i [l :v :c :z     :i :z]
                                   [l :v :c :l - 1 :i :z]]
   se 31 = conteúdo :v [se :l = :z [l :v :c :i     :i :z]
                                   [l :v :c :l + 1 :i :z]]
   l :v :c :l :i :z
fim

ap liste :fig
   . 0 19 mo listafig :fig   atr 1 care
   . 0 19
   att
   o3
fim

ap m :c :l :t
   atr "x 0
   .  0 19 esc []
```

```
   . 11  0 esc "CONTROLE
   . 14  1 esc "DE
   . 11  2 esc :t
   repita nel :m [atr "x :x + 1
                  . :c :l + :x - 1 esc sn "\ \ elemento :x :m]
   . 10 21 esc "Baga!Soft"
   . 10 22 esc "---------"
   .  2  8 ponha "FIG
   . 25  8 ponha "FIG
   . 12 16 esc [COR Nº]
   f
   c
fim

ap most
   f
   atr "c ascii care
   se e fig =  0 :c = 29 [mudefig 59 most]
   se e fig = 59 :c = 28 [mudefig  0 most]
   se            :c = 28 [mudefig fig + 1]
   se            :c = 29 [mudefig fig - 1]
   se            :c = 27 [o2]
   se            :c = 30 [aum]
   se            :c = 31 [dim]
   se            :c = 32 [edfig fig]
   most
fim

ap n
   se ( algum "m = pri :p "a = pri :p "p = pri :p ) [atr "x care]
fim

ap o :p :m
   ad
   att
   dt
   se :m = 9 [n faça sn :p []]
             [atr "e line
              se évazia :e
               [faça sn pal "o :a []]
               [se :m = 0 [faça lista :p :e]
                          [faça lista :p pal "" pri :e]]]
   n
   att
   faça sn pal "o :a []
fim

ap o0
   att
   k "\ SPRITE 8     [\ EDIÇÃO \ FIGURA VARIÁVEL ARQUIVOS]
     []
     "a
fim

ap o1
   k "\ EDIÇÃO 8
     [EDITAR ROLAR ROTAR DELETAR INVERTER REDUZIR AMPLIAR]
     [[edfig fig] [role fig] [a] [del fig]
      [criafigl fig inv listafig fig] [a] [a]]
     "g
fim

ap o2
   k "\ FIGURA 8
     [COLORIR MOSTRAR COPIAR DIMINUIR AUMENTAR CRIAR]
     [[cor] [most] [copy] [dim] [aum] [a]]
     "g
fim

ap o3
   q
   k "VARIÁVEL 8
     [LISTAR NOMEAR PROGRAMAR EDITAR ELIMINAR MOSTRAR]
     [[liste fig] [a] [pro] [o "edns 9]
      [o "eln 0] [o "mons 9]]
     "g
fim

ap o4
   att q
   k "ARQUIVOS 10
     [DIR TYPE LOAD KILL SAVE]
     [["arquivos   9] ["mostrearq 1] ["carregue 1]
      ["eliminearq 1] ["gravetudo 1]]
     "o
fim
```

```
ap p1
   atat [2 3]
   un
   criafigl 10 [112 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
   atr "x2 -112
   atr "x3   73
   atr "y     8
   sótat 2 [, :x2 :y]
   sótat 3 [, :x3 :y]
   mudefig 10
fim

ap p2
   se :n = 17 [atr "x2 -70
               atr "x3 115
               atr "y    8]
   sótat 2 [, :x2 :y]
   sótat 3 [, :x3 :y]
   at
fim
ap p3
   se e :n = 17 :bt = "v [atr "x2 -112
                          atr "x3   73
                          atr "y   -20,8]
fim

ap pdb
   b
   p2
   atr "bt line
   se évazia :bt            [pdb]
                            [atr "bt pri :bt p3]
   se e :z = "b 8 < nel :bt [pdb]
   se :bt  = "s             [sai]
   se :bt  = "v             [vol]
   se évazia :fon           [fyg]
   se :bt  = "p             [pul]
   se :bt  = "e             [esp]
   se :bt  = "ca            [can]
   se :bt  = "cs            [cse]
   se não énúmero :bt       [fyg]
   se não éint :bt          [fyg]
   se :z   = "b             [atr "bt bd :bt]
   se e :bt > -1 :bt < 256  [cri]
   fyg
fim

ap pow :b :e
   se :e = 0 [envie 1]
             [envie :b * pow :b :e - 1]
fim

ap pro
   atr "des []
   atr "fon listafig fig
   atr "bt  []
   atr "n   1
   . 7 19 esc [bINÁRIO##dECIMAL]
   p1
   atr "z care
   se algum :z = "b :z = "d [pdb]
                            [pro]
fim

ap pul
   atr "des sn :des pri :fon
   atr "fon sp :fon
   atr "y   :y - 1,8
   atr "n   :n + 1
fim

ap q
   atat 2
   mudecl 15
   un , -112 16 ul , -77 16 , -77 -15 , -112 -15 , -112 16
   un ,   73 16 ul , 108 16 , 108 -15 ,   73 -15 ,   73 16
   atat [0 1]
fim
ap role :fig
   atr "lfig dlt [] listafig :fig
   atr "p1   pri :lfig
   atr "p2   ult :lfig scro
fim
```

```
ap sai
   atat [2 3]
   dt
   atat [0 1]
   o3
fim

ap scr1
   atr "p1 sn sp :p1 pri :p1
   atr "p2 sn sp :p2 pri :p2
fim

ap scr2
   atr "p1 sn ult :p1 su :p1
   atr "p2 sn ult :p2 su :p2
fim

ap scr3 :1 :2
   atr "a1 2 * pri :1
   atr "a2 2 * pri :2
   se :a1 > 255 [atr "a1 :a1 - 256
                 atr "k1 1]
                [atr "k1 0]
   se :a2 > 255 [atr "a2 :a2 - 256
                 atr "k2 1]
                [atr "k2 0]
   atr "a1 :a1 + :k2
   atr "a2 :a2 + :k1
   atr "b1 sn :b1 :a1
   atr "b2 sn :b2 :a2
   criafigl :fig ( sn :b1 sp :1 :b2 sp :2 )
   se évazia sp :1 [atr "p1 :b1
                    atr "p2 :b2
                    pare]
                   [scr3 sp :1 sp :2]
fim

ap scr4 :1 :2
   atr "a1 pri :1
   atr "a2 pri :2
   se eimpar :a1 [atr "a1 :a1 - 1
                  atr "k1 1]
                 [atr "k1 0]
   se eimpar :a2 [atr "a2 :a2 - 1
                  atr "k2 1]
                 [atr "k2 0]
   atr "b1 sn :b1 :a1 / 2 + 128 * :k2
   atr "b2 sn :b2 :a2 / 2 + 128 * :k1
   criafigl :fig ( sn :b1 sp :1 :b2 sp :2 )
   se évazia sp :1 [atr "p1 :b1
                    atr "p2 :b2
                    pare]                    [scr4 sp :1 sp :2]
fim

ap scro
   atr "b1 []
   atr "b2 []
   atr "x ascii care
   se :x = 27 [o1]
   se :x = 30 [scr1]
   se :x = 31 [scr2]
   se :x = 29 [scr3 :p1 :p2]
   se :x = 28 [scr4 :p1 :p2]
   criafigl :fig sn :p1 :p2
   scro
fim

ap tudo
   atat todas
   dt
   tat
   elns
   mudecf 0
   cap
   liberemem
   un
   att
   aum
   o0
fim

ap var :o :prim
   se :prim < 10 [envie pal :o :prim]
                 [envie        :prim]
fim
```

```
ap vol
   se não évazia :des [atr "fon sn ult :des :fon
                       atr "des su :des
                       atr "y   :y + 1,8
                       atr "n   :n - 1]
fim

ap z :z
   att
   liberemem
   faça sn pal "o :z []
fim

ap éint :m
   se sãoiguais :m int :m [envie  verd]
                          [envie falso]
fim
```

**REFERÊNCIAS BIBLIOGRÁFICAS**


1 - AVALON Software. O livro vermelho do MSX.
São Paulo, McGraw-Hill, 1988. 323 p.

2 - BECKER, Fernando et alii. Apresentação de trabalhos escolares.
11 ed. Porto Alegre, Multilivro, 1990. 67 p.

3 - Manual de operação do TK2000 Color.
São Paulo, Microdigital Eletrônica Ltda. 210 p.

4 - VALENTE, José Armando & VALENTE, Ann Berger. Logo: conceitos, aplicações e projetos.
São Paulo, McGraw-Hill, 1988. 292 p.