MSX Article

Sprites e Gravidade

Resumo

O objetivo deste artigo é mostrar como aplicar o efeito de gravidade em uma bolinha feita a partir do recurso de sprites do MSX 1.

1- Criando a bolinha - sprites

O recurso de sprites no MSX serve para criar máscaras (figuras) para serem desenhadas sobre a tela, podendo ser movimentadas livremente, e sem causar prejuízo a tela de fundo, como por exemplo, apagar a área sob a máscara ao movimentar-se.

A figura do sprite é definida por áreas que irão acender e sobrepor o fundo da tela e outras que ficarão apagadas e manter o fundo da tela. As áreas acesas serão representadas por uma única cor no MSX 1, definida no comando “PUT SPRITE”.

Há dois tamanhos de sprite: 8x8 e 16x16, que serão definidos no comando “SCREEN modo, sprite”, onde a opção “sprite” irá valer:

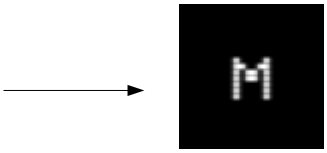
- 0 – sprites 8x8 normal
- 1 – sprites 8x8 aumentado
- 2 – sprites 16x16 normal
- 3 – sprites 16x16 aumentado

A instrução em Basic “SPRITE$(n) = string” define o desenho que irá aparecer na tela. A variável “n” é uma identificação para o sprite, que varia de 0 a 255 no modo 8x8, e de 0 a 63 no modo 16x16. Já “string” é uma cadeia de caracteres que irá conter a configuração de desenho do sprite.

Cada caractere da string de definição do sprite é um código ASCII que irá conter a informação de pixel aceso e pixel apagado do sprite. Ao converter-se esse valor do código ASCII para binário, os valores de bit iguais a um irão acender o pixel do sprite, enquanto que os valores de bit igual a zero irão apagar.

No modo 8x8, cada caractere define uma linha, onde cada bit do código ASCII define uma coluna da tela. O exemplo a seguir contém a configuração para desenhar a letra “M” no modo 8x8:

```
A$ = CHR$(&b01000010)
B$ = CHR$(&b01100110)
C$ = CHR$(&b01011010)
D$ = CHR$(&b01011010)
E$ = CHR$(&b01000010)
F$ = CHR$(&b01000010)
G$ = CHR$(&b01000010)
H$ = CHR$(&b00000000)
```



```
SPRITE$(1) = A$+B$+C$+D$+E$+F$+G$+H$
```

Note que os bits 0 e 1 definem a forma do desenho. Os valores igual a “1” foram destacados em vermelho para melhor visualizar o formato da letra “M”.

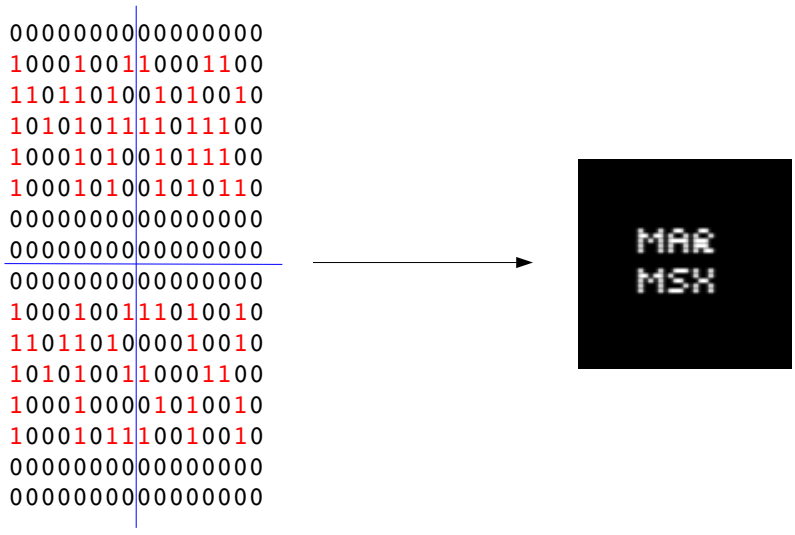
Já o modo 16x16 é composto de 4 quadrantes, onde cada quadrante é formado por um bloco de 8x8 de 0 e 1's, configurados da seguinte maneira:

Q1 Q3
Q2 Q4

Onde os quadrantes serão empilhados na ordem: Q1, Q2, Q3 e Q4.

Configuração do desenho na tela	Empilhamento para as instruções READ-DATA
0101010100000000 0101010100000000 0101010100000000 0101010100000000 0101010100000000 0101010100000000 0101010100000000 0101010100000000 1010101011111111 1010101011111111 1010101011111111 1010101011111111 1010101011111111 1010101011111111 1010101011111111	01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111

Um exemplo de desenho fica:



No exemplo do manual do Expert [1] relativo ao comando “PUT SPRITE”, há uma “receita de bolo” de como desenhar um sprite na tela. O programa a seguir, substitui o desenho do programa do livro por uma bolinha.

sprite.bas

```
10 SCREEN 2,0
20 FOR T=1 TO 8
30 READ A$
40 S$=S$+CHR$(VAL("&B"+A$))
50 NEXT T
60 SPRITE$(1)=S$
70 PUT SPRITE 0,(128,96),15,1
80 GOTO 80
100 DATA 00111100
110 DATA 01111110
120 DATA 11111111
130 DATA 11111111
140 DATA 11111111
150 DATA 11111111
160 DATA 01111110
170 DATA 00111100
```

Esse programa trata os dados contidos em DATA como uma string de 8 caracteres, onde futuramente irão receber o prefixo “&B”, para transformá-los em um número binário. Então, cada valor de DATA é convertido de “binário-string” para um valor inteiro. Finalmente, obtém-se o caractere correspondente ao valor de inteiro obtido.

O comando “PUT SPRITE camada, (x,y), cor, id_sprite” irá desenhar o sprite na tela.

É possível fornecer também os valores de DATA em outros formatos, como, por exemplo binário, decimal e hexadecimal.

sprite2.bas

```
10 SCREEN 2,0
20 FOR T=1 TO 8
30 READ A
40 S$=S$+CHR$(A)
50 NEXT T
60 SPRITE$(1)=S$
70 PUT SPRITE 0,(128,96),15,1
80 GOTO 80
100 DATA &B00111100
110 DATA &B01111110
120 DATA &B11111111
130 DATA &B11111111
140 DATA &B11111111
150 DATA &B11111111
160 DATA &B01111110
170 DATA &B00111100
```

Dados no formato binário.

sprite3.bas

```
10 SCREEN 2,0
20 FOR T=1 TO 8
30 READ A$
40 S$=S$+CHR$(VAL("&H"+A$))
50 NEXT T
60 SPRITE$(1)=S$
70 PUT SPRITE 0,(128,96),15,1
80 GOTO 80
100 DATA 3C,7E,FF,FF,FF,FF,7E,3C
```

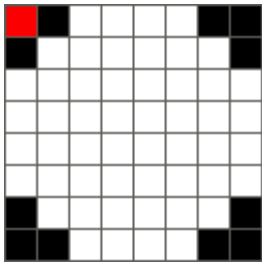
ou

```
10 SCREEN 2,0
20 FOR T=1 TO 8
30 READ A
40 S$=S$+CHR$(A)
50 NEXT T
60 SPRITE$(1)=S$
70 PUT SPRITE 0,(128,96),15,1
80 GOTO 80
100 DATA &H3C,&H7E,&HFF,&HFF,&HFF,&HFF,
&H7E,&H3C
```

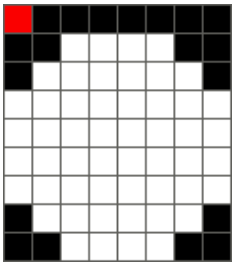
Dados no formato hexadecimal.

A movimentação da bolinha é feita alterando-se o valor das coordenadas x, y do “PUT SPRITE”.

A coordenada x, y define a origem do desenho e ela se localiza no canto superior do sprite, conforme mostrado pelo ponto vermelho da figura 1a. Entretanto, há um bug no MSX quando o sprite é desenhado, no qual a figura é deslocada y+1 pixel (figura 1b).



a) Coordenada x,y teórica do sprite.



b) Coordenada x,y real do sprite.

Figura 1. Coordenada x,y do sprite.

Assim, os limites do sprite são definidos por: (x, y+1) – (x+7, y+8).

Quando o sprite é no formato 16x16, a linha 20 é substituída por:

```
20 FOR T=1 TO 32
```

A “camada” do comando “PUT SPRITE” irá definir o nível de prioridade de desenho. Os sprites de menor nível irão desenhar sobre os de maior nível. Varia de 0 a 31.

2- Movimento de Queda Livre no MSX

Depois de desenhar a bolinha, aplicamos nela os conceitos de aceleração da gravidade e queda livre, para vê-la caindo e quicando sobre uma “superfície” na tela.

Na física, as seguintes equações definem o movimento uniformemente variado (MUV) de um corpo:

I. $v = v_0 + a \,.\, \Delta t$

II. $s = s_0 + v_0 \,.\, \Delta t + \frac{a \,.\, \Delta t^2}{2}$

Para um movimento vertical, o valor da aceleração é a própria aceleração da gravidade, representada pela letra “g”. As equações são reescritas da seguinte maneira:

III. $v = v_0 + g \,.\, \Delta t$

IV. $s = s_0 + v_0 \,.\, \Delta t + \frac{g \,.\, \Delta t^2}{2}$

A rotina responsável pelo movimento da bolinha estará dentro de um loop, onde serão calculados a cada iteração os valores de duas incógnitas: a nova velocidade da bolinha e sua nova posição na tela.

Você deve ter se perguntado e o valor de Δt? Δt é o tempo decorrido entre uma iteração e outra. Esse tempo pode ser medido e adotado no programa. Entretanto, vamos assumir que o tempo entre cada iteração seja igual a 0.2 segundos.

Será adotada como a origem do sistema o local da bolinha no instante t=0. O sentido do eixo y, no qual o movimento da bolinha se dará, será o mesmo do eixo y da tela do MSX, conforme mostra a figura 2.

A equação IV será reescrita, de forma a se calcular o deslocamento da bolinha e facilitar os cálculos.

V. $$\Delta s = v_0 . \Delta t + \frac{g . \Delta t^2}{2}$$

Os valores conhecidos são:

- g = 9,81 m/s²
- Δt = 0.2 s

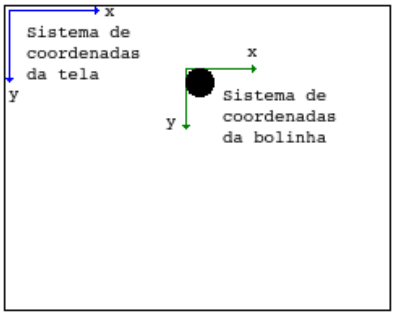


Figura 2. Sistemas de coordenadas.

O programa a seguir faz o cálculo da velocidade e posição da bolinha em uma dada iteração.

grav1.bas

```
10 SCREEN 0
20 G=9.81
30 DT=0.2
40 V0=0
50 S0=0
60 GOSUB 500
70 PRINT"NOVA VELOCIDADE:";V0
80 PRINT"NOVA POSICAO:";S0+DS
90 END
497 '
498 ' GRAVIDADE
499 '
500 V = V0 + G*DT
510 DS = V0*DT + (G*DT^2)/2
520 RETURN
```

3- Aplicando o Conceito de Queda Livre na bolinha

Agora é só juntar tudo o que foi feito até aqui. O código de desenho da bolinha e o código de cálculo da gravidade.

gravbol.bas

```
10 SCREEN 2,0
17 '
18 ' Parametros iniciais
19 '
20 G=9.81
30 DT=0.2
40 V0=0
50 S0=0
57 '
58 ' Cria a bolinha
59 '
60 FOR T=1 TO 8
70 READ A$
80 S$=S$+CHR$(VAL("&B"+A$))
90 NEXT T
100 SPRITE$(1)=S$
110 DATA 00111100
120 DATA 01111110
130 DATA 11111111
140 DATA 11111111
150 DATA 11111111
160 DATA 11111111
170 DATA 01111110
180 DATA 00111100
190 '
191 ' Loop de movimento da bola
192 '
200 PUT SPRITE 0,(128,S0),15,1
210 GOSUB 500
220 S0 = S0 + DS
230 V0 = V
240 IF S0 > 220 THEN END ELSE 200
497 '
498 ' GRAVIDADE
499 '
500 V = V0 + G*DT
510 DS = V0*DT + (G*DT^2)/2
520 RETURN
```

O programa acima faz com que a bolinha “caia” até ultrapassar a borda inferior da tela. Entretanto, para ficar mais realista, vamos acrescentar um chão e fazer a bolinha quicar até parar. Para isso, as seguintes considerações terão que ser feitas:

- Quando a bolinha chegar ou ultrapassar o chão (lembre-se dos limites da bolinha apresentados na seção 1), travar a bolinha acima da superfície.
- De acordo com a lei de ação e reação, inverter o sentido de movimento (o chão empurra a bolinha de volta para cima com a mesma força).
- Absorver parte da energia da bolinha, retirando um pouco de sua velocidade (agora considera-se que o impacto no chão absorve energia).

Obs: levar em conta o bug da posição y+1 do sprite.

gravbol2.bas

```
10 SCREEN 2,0
17 '
18 ' Parametros iniciais
19 '
20 G=9.81
30 DT=0.2
40 V0=0
50 S0=0
57 '
58 ' Cria a bolinha
59 '
60 FOR T=1 TO 8
70 READ A$
80 S$=S$+CHR$(VAL("&B"+A$))
90 NEXT T
100 SPRITE$(1)=S$
110 DATA 00111100
120 DATA 01111110
130 DATA 11111111
140 DATA 11111111
150 DATA 11111111
160 DATA 11111111
170 DATA 01111110
180 DATA 00111100
181 '
182 ' Desenha o chão
183 '
184 LINE(0,180)-(255,211),3,BF
190 '
191 ' Loop de movimento da bola
192 '
200 PUT SPRITE 0,(128,S0),15,1
210 GOSUB 500
220 S0 = S0 + DS
230 V0 = V
240 IF S0+8 >= 179 THEN V0 = -V0*0.7 : S0=179-8
250 GOTO 200
497 '
498 ' GRAVIDADE
499 '
500 V = V0 + G*DT
510 DS = V0*DT + (G*DT^2)/2
520 RETURN
```

Nesse programa, a bolinha atinge e ultrapassa o chão antes de iniciar o movimento de volta para cima. Dessa forma, a velocidade utilizada para calcular o movimento de retorno corresponde a uma posição além do chão, em vez de ser a velocidade correspondente a posição localizada exatamente no chão.

Ao desprezarmos esse fato, geramos um movimento oscilatório quando a bolinha tenta parar no chão, pois uma energia “extra” é introduzida no sistema.

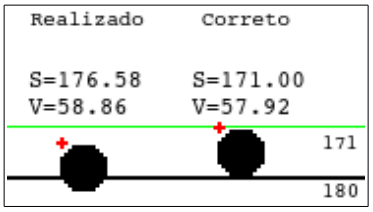


Figura 3. Correção da posição da bolinha.

A figura 3 ilustra o erro mencionado, quando a bola atinge o chão pela primeira vez. A velocidade calculada é 58,86 *m/s*, quando deveria ser 57,92 *m/s*. Dessa forma, uma "energia" correspondente a uma velocidade de 0,94 *m/s* é adicionada ao sistema.

A velocidade para S=171 *m* pode ser calculada utilizando-se a Equação de Torricelli:

VI. $v^2 = v_0^2 + 2a\Delta s$, onde a=g.

Assim:

$(58{,}86)^2 = v_0^2 + 2 \times 9{,}81 \times (176{,}58 - 171)$ | $v_0^2 = 3464{,}4996 - 109{,}4796$

$v_0^2 = 3464{,}4996 - 2 \times 9{,}81 \times 5{,}58$ | $v_0 = \sqrt{3355{,}02} = 57{,}92$

A linha 240 pode ser modificada para:

```
240 IF S0+8 >= 179 THEN DS=171-S0:V0=SQR(ABS(V0^2+2*G*DS))*-.7:S0=179-8
```

O uso desse cálculo gera um retardo na animação da bolinha. No MSX Turbo-R, o retardo dos cálculos não é notado.

Uma solução alternativa é detectar quando a bolinha atinge o chão e não tem mais forças para subir, interrompendo a animação. Utilizando-se a Equação de Torricelli, calculamos a velocidade mínima para a bolinha quicar e mover um pixel para cima:

$v^2 = v_0^2 + 2a\Delta s$ | $v_0^2 = -(-19{,}62)$

$0 = v_0^2 + 2 \times 9{,}81 \times (-1)$ | $v_0 = \sqrt{19{,}62} = 4{,}43$

Alterando-se as seguintes linhas programa anterior, resolvemos o problema:

```
210 IF EN=1 THEN 200 ELSE GOSUB 500
...
240 IF S0+8>=179 THEN V0 = -V0*0.7 : S0=179-8 : IF ABS(V0)<4.4 THEN EN=1
```

Pode-se também introduzir um movimento horizontal na bolinha. O programa a seguir reproduz o efeito da bolinha como em um jogo de squash, onde há movimento horizontal e vertical e o acréscimo de duas paredes para a bolinha ricochetear.

squash.bas

```
10 SCREEN 2,0
17 '
18 ' Parametros iniciais
19 '
20 DT=0.2
30 VV=-50
40 SV=172
50 VH=40
51 SH=5
57 '
58 ' Cria a bolinha
59 '
60 FOR T=1 TO 8
70 READ A$
80 S$=S$+CHR$(VAL("&B"+A$))
90 NEXT T
100 SPRITE$(1)=S$
110 DATA 00111100
120 DATA 01111110
130 DATA 11111111
140 DATA 11111111
150 DATA 11111111
160 DATA 11111111
170 DATA 01111110
180 DATA 00111100
181 '
182 ' Desenha o chão e as paredes
183 '
184 LINE(0,180)-(255,211),3,BF
185 LINE(250,0)-(255,179),14,BF
187 LINE(0,0)-(5,179),14,BF
190 '
191 ' Loop de movimento da bolinha
192 '
200 PUT SPRITE 0,(SH,SV),15,1
210 IF EN<>1 THEN V0=VV : A=9.81 : GOSUB 500 : ' Mov. Vertical
220 IF EN<>1 THEN VV=V0 : SV = SV + DS
230 V0=VH : A=0 : GOSUB 500 : ' Mov. Horizontal
240 VH=V0 : SH = SH + DS
250 IF SV+8 >= 179 THEN VV = -VV*0.8 : VH = VH*0.8 : SV=179-8 : IF
ABS(VV) < 4.4 THEN EN=1
260 IF SH+8 >= 250 THEN VV = VV*0.8 : VH = -VH*0.8 : SH=250-8
270 IF SH <= 5 THEN VV = VV*0.8 : VH = -VH*0.8 : SH=5
280 GOTO 200
497 '
498 ' Movimento
499 '
500 V = V0 + A*DT
510 DS = V0*DT + (A*DT^2)/2
520 V0 = V
530 RETURN
```

Obs: Os programas-fonte acompanham o artigo na página MarMSX Development.

4- Créditos

Este artigo foi escrito por Marcelo Silveira, em Outubro de 2016 e revisado em Setembro de 2017 e Junho de 2018.

E-mail: flamar98@hotmail.com
Homepage: http://marmsx.msxall.com

Referências:

[1] - Livro: Linguagem Basic MSX, editora Aleph, 5a. Edição, 1987.
[2] - Livro: Dominando o Expert, editora Aleph, 5a. Edição, 1987.