

MSX Article

MARMSX

Sprites and Gravity

Summary

This article shows how to apply the free fall laws on a MSX 1 sprite ball.

1- Creating the ball - sprites

The MSX resource called sprites aims at drawing masks (images) on screen to be moved freely without interfering on the contents of the background image.

The sprite shape is defined by pixels that will turn on and overlap the background or stay turned off, preserving the background content. Each sprite can only be represented by a single color on MSX 1, defined by the instruction “PUT SPRITE”.

There are two possible sprites sizes: 8x8 e 16x16, defined by the instruction “SCREEN, mode, sprite”, where the option “sprite” can be:

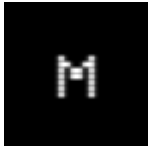
- 0 – sprites 8x8 normal
- 1 – sprites 8x8 big
- 2 – sprites 16x16 normal
- 3 – sprites 16x16 big

The Basic instruction “SPRITE\$(n) = string” stores the sprite image that will be displayed on screen. The variable “n” is the sprite identification, ranging from 0 to 255 on 8x8 mode or 0 to 63 on 16x16 mode. The “string” variable holds the sprite pattern.

Each “string” character describes one sprite pattern line (1x8 pixels) and each bit defines a pixel. Value 1 means pixel turned on, while value 0 means pixel turned off.

See the next example for a 8x8 sprite that draws the letter “M”.

```
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$+H$
```



Notice that the bits 0 and 1 define the image shape. The values “1” were detached as red to be more comfortable to see the “M” shape.

Yet, the 16x16 mode is composed by 4 quadrants, where each quadrant is composed by an 8x8 block arranged as follows:

Q1 Q3
Q2 Q4

Where the quadrants will be stacked: Q1, Q2, Q3 and Q4.

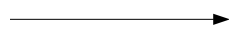
Screen results	The READ-DATA pattern stack
0101010100000000	01010101
0101010100000000	01010101
0101010100000000	01010101
0101010100000000	01010101
0101010100000000	01010101
0101010100000000	01010101
0101010100000000	01010101
0101010100000000	01010101
0101010100000000	01010101
1010101011111111	10101010
1010101011111111	10101010
1010101011111111	10101010
1010101011111111	10101010
1010101011111111	10101010
1010101011111111	10101010
1010101011111111	10101010
1010101011111111	10101010
1010101011111111	10101010
1010101011111111	00000000
1010101011111111	00000000
1010101011111111	00000000
1010101011111111	00000000
1010101011111111	00000000
1010101011111111	00000000
1010101011111111	00000000
1010101011111111	00000000
1010101011111111	11111111
1010101011111111	11111111
1010101011111111	11111111
1010101011111111	11111111
1010101011111111	11111111
1010101011111111	11111111
1010101011111111	11111111
1010101011111111	11111111
1010101011111111	11111111

An example:

```

0000000000000000
1000100110001100
1101101001010010
1010101111011100
1000101001011100
1000101001010110
0000000000000000
0000000000000000
0000000000000000
0000000000000000
10001001111010010
1101101000010010
1010100110001100
1000100001010010
1000101110010010
0000000000000000
0000000000000000

```



We can find a basic sprite algorithm on the Expert manual [1], located at the “PUT SPRITE” instruction. The next program replaces the original sprite pattern by the ball.

sprite.bas
<pre> 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 001111100 110 DATA 011111110 120 DATA 111111111 130 DATA 111111111 140 DATA 111111111 150 DATA 111111111 160 DATA 011111110 170 DATA 001111100 </pre>

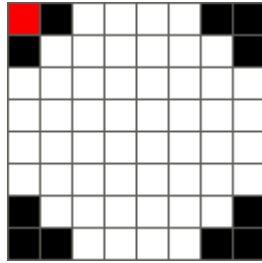
This program considers the value after the DATA as a string with 8 characters length. This value will be concatenated with the prefix “&B” to be converted to a binary number. At last, the converted value will represent an ASCII code on the string “S\$”.

The instruction “PUT SPRITE layer, (x,y), color, id_sprite” draws the sprite on screen. Also, it is possible to use other value formats for the DATA instruction. Let's see.

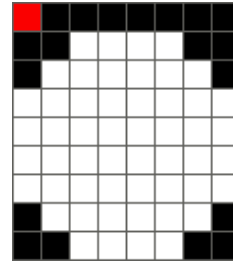
sprite2.bas	sprite3.bas
<pre> 10 SCREEN 2,0 20 FOR T=1 TO 8 30 READ A 40 S\$=S\$+CHR\$(A) 50 NEXT T 60 SPRITE\$(1)=S\$ 70 PUT SPRITE 0,(128,96),15,1 80 GOTO 80 100 DATA &B001111100 110 DATA &B011111110 120 DATA &B111111111 130 DATA &B111111111 140 DATA &B111111111 150 DATA &B111111111 160 DATA &B011111110 170 DATA &B001111100 </pre>	<pre> 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 </pre>
or	
<pre> 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 </pre>	<pre> 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 </pre>
Pattern as binary data.	Pattern as hexadecimal data.

To move the ball, just change the x,y coordinates on the “PUT SPRITE”.

The x,y coordinates define the sprite location on screen based on the top-left pixel, as seen on figure 1a. Nevertheless, there is a bug on MSX that shifts the image one pixel downwards, as described on figure 1b.



a) Theoretical x,y sprite coordinates.



b) Real x,y sprite coordinates.

Figure 1. Sprite x,y coordinates.

According to that, the sprite bounding-box is defined as: $(x, y+1) - (x+7, y+8)$.

When the sprite is 16x16 mode, line 20 is replaced by:

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

The “layer” from “PUT SPRITE” defines the image render priority level. The lower layer value sprites will draw over those with higher values. The layer ranges from 0 to 31.

2- Free fall motion on MSX

After drawing the ball, we may apply the physics Free Fall Motion laws and watch it fall and bounce on a screen “surface”.

These equations defines the Uniform Motion laws:

$$\text{I. } v = v_0 + a \cdot \Delta t$$

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

For a free fall vertical motion, the aceleration is the gravity aceleration, represented by the letter “g”. So, these equations are rewritten as follows:

$$\text{III. } v = v_0 + g \cdot \Delta t$$

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

The routine responsible for the ball motion must lie inside a loop, where each iteration calculates the new ball speed and position using these formulas.

And what about the Δt ? Δt is the time spent between each iteration. Let's assume that each iteration takes 0.2 seconds.

We will consider the system origin as the ball location, defined by x,y , when $t=0$. Figure 2 describes the screen and object (ball) coordinates system.

The equation IV is rearranged in order to calculate the spatial shift directly.

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

The known values are:

- $g = 9,81 \text{ m/s}^2$
- $\Delta t = 0.2 \text{ s}$

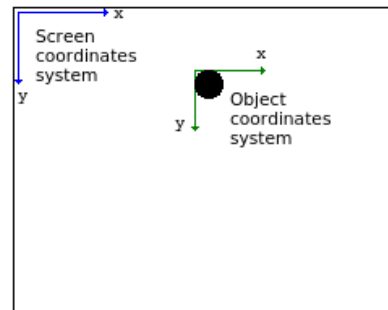


Figure 2. Coordinates systems.

The next program calculates the ball speed and position for a given iteration.

```

                                grav1.bas
10 SCREEN 0
20 G=9.81
30 DT=0.2
40 V0=0
50 S0=0
60 GOSUB 500
70 PRINT"NEW SPEED:";V0
80 PRINT"NEW POSITION:";S0+DS
90 END
497 '
498 ' GRAVITY
499 '
500 V = V0 + G*DT
510 DS = V0*DT + (G*DT^2)/2
520 RETURN

```

3- Applying the free fall laws onto the ball

Now our task is to put everything done here together.

The next program draws the ball and make it fall freely.

gravbol.bas

```
10 SCREEN 2,0
17 '
18 ' Initial parameters
19 '
20 G=9.81
30 DT=0.2
40 V0=0
50 S0=0
57 '
58 ' Create the ball
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 ' Ball motion loop
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 ' GRAVITY
499 '
500 V = V0 + G*DT
510 DS = V0*DT + (G*DT^2)/2
520 RETURN
```

This program creates a ball that falls on screen until passes completely the bottom of the screen. In order to be more realistic, let's add a "floor" on screen and do the ball bounce on it until stop. For that, we may consider that:

- When the ball exceeds the floor limits (remember the ball's bounding-box), fix its position placing it on the floor.
- Reverse the motion (Newton's third law).
- Consider that the floor absorbs a part of the ball's energy (reduce the speed value) .

Obs: take in account the sprite position y+1.

gravbol2.bas

```
10 SCREEN 2,0
17 '
18 ' Initial parameters
19 '
20 G=9.81
30 DT=0.2
40 V0=0
50 S0=0
57 '
58 ' Create the ball
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 ' Draw the floor
183 '
184 LINE(0,180)-(255,211),3,BF
190 '
191 ' Ball motion loop
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 ' GRAVITY
499 '
500 V = V0 + G*DT
510 DS = V0*DT + (G*DT^2)/2
520 RETURN
```

In this program, the ball reaches and exceeds the floor limits before reverse the motion. By doing that, the considered speed to start the upwards motion is greater than the ball speed when exactly on the floor. Notice that despite the ball position fix, the speed is not correct for that position. See figure 3.

If we do not consider the ball floor exceeding position, we are introducing an extra energy to the system and the ball never stops.

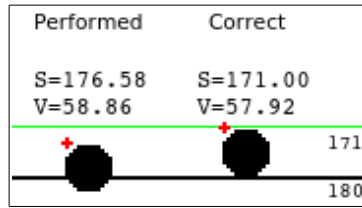


Figure 3. Ball position fix.

Figure 3 illustrates the problem, when the ball reaches the floor at the first time. The calculated speed is 58,86 m/s when should be 57,92 m/s. According to that, an extra “energy” corresponding to the speed of 0,94 m/s is added to the system.

By using the Torricelli Equation, we can calculate the speed at $S=171\text{ m}$.

$$\text{VI. } v^2 = v_0^2 + 2a \Delta s, \text{ where } a=g.$$

So:

$$\begin{aligned} (58,86)^2 &= v_0^2 + 2 \times 9,81 \times (176,58 - 171) & v_0^2 &= 3464,4996 - 109,4796 \\ v_0^2 &= 3464,4996 - 2 \times 9,81 \times 5,58 & v_0 &= \sqrt{3355,02} = 57,92 \end{aligned}$$

The line 240 can be rewritten as:

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

This extra expression slows down the ball animation. If we use the MSX Turbo-R, this delay is not noticed.

A paliative solution for that is to detect when the ball reaches the floor and does not have enough energy to go upwards. By using the Torricelli Equation, we calculate the minimum speed necessary to move 1 pixel upwards:

$$\begin{aligned} v^2 &= v_0^2 + 2a \Delta s & v_0^2 &= -(-19,62) \\ 0 &= v_0^2 + 2 \times 9,81 \times (-1) & v_0 &= \sqrt{19,62} = 4,43 \end{aligned}$$

Then change the following lines on the last program:

```
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
```

It is also possible to introduce a horizontal motion on the ball. The next program simulates a squash game, with both vertical and horizontal motions.

squash.bas

```
10 SCREEN 2,0
17 '
18 ' Initial parameters
19 '
20 DT=0.2
30 VV=-50
40 SV=172
50 VH=40
51 SH=5
57 '
58 ' Create the ball
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 001111100
120 DATA 011111110
130 DATA 111111111
140 DATA 111111111
150 DATA 111111111
160 DATA 111111111
170 DATA 011111110
180 DATA 001111100
181 '
182 ' Draw the floor and the walls
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 ' Ball motion loop
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 ' Motion
499 '
500 V = V0 + A*DT
510 DS = V0*DT + (A*DT^2)/2
520 V0 = V
530 RETURN
```

Obs: this article is followed by the sources at MarMSX Development page.

4- Credits and References

This article was written original in Portuguese and translated into English by Marcelo Silveira.

Written: October 2016

Review 1: September 2017

Review 2: June 2018

E-mail: flamar98@hotmail.com

Homepage: <http://marmsx.msxall.com>

References:

[1] - Book: Linguagem Basic MSX, editora Aleph, 5a. Edição, 1987.

[2] - Book: Dominando o Expert, editora Aleph, 5a. Edição, 1987.