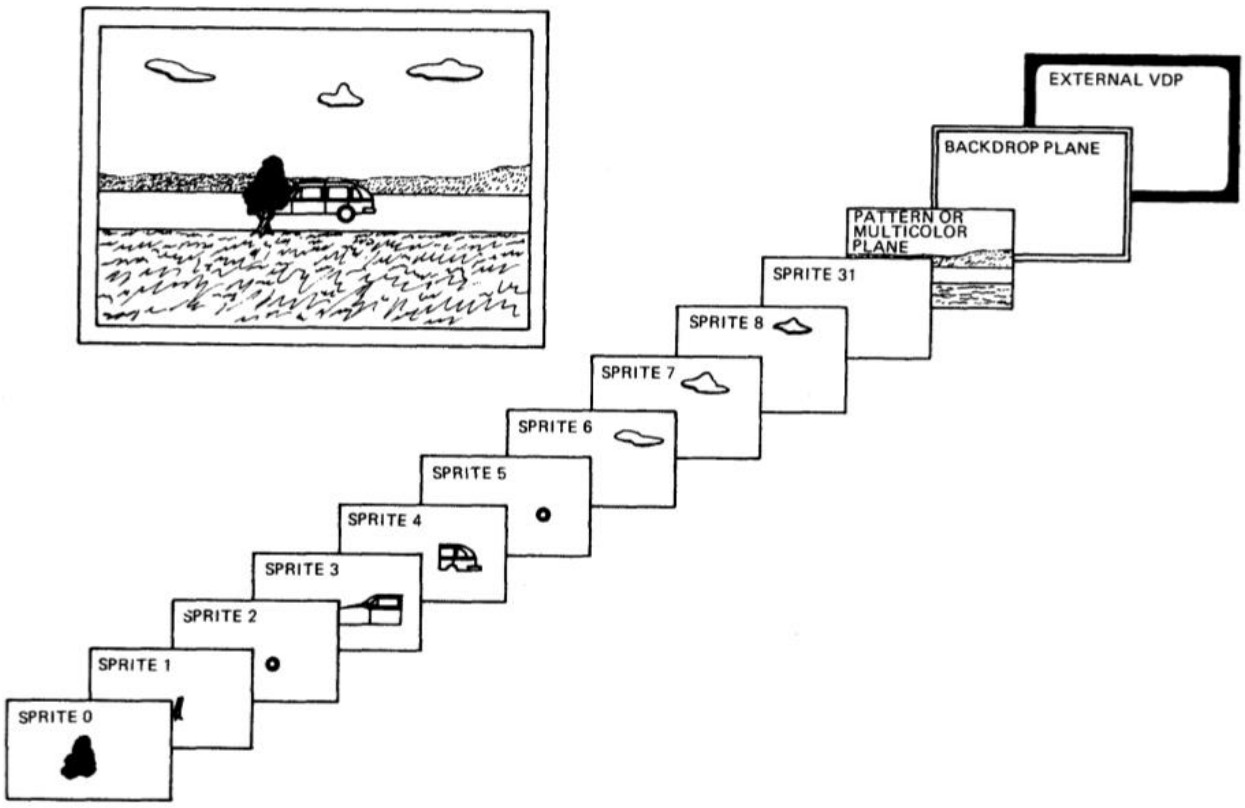
Let’s Make a Retro Game

Episode 15 – TMS9918A Graphics - Sprites

In this episode we are going to continue on from the last episode where we were looking at how the TMS9918A graphics chip displays tile graphics, making a bit of a graphical demo in the process.

As well as tiles the TMS processor allows 32 sprites to be displayed on top of the current back ground.



Let's Make a Retro Game

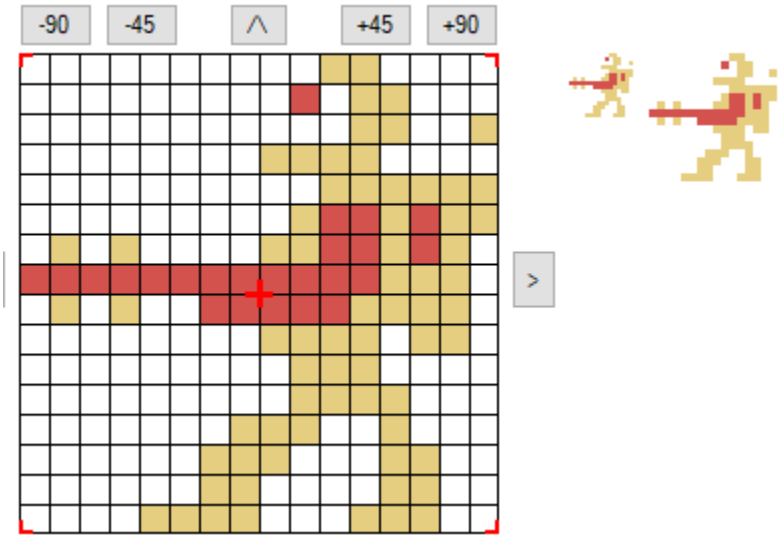
Episode 15 – TMS9918A Graphics - Sprites

The sprites can either be in one of four modes:

- 8x8 pixels
- 16x16 pixels
- 8x8 pixels magnified
- 16x16 pixels magnified

This mode applies to all sprites in use i.e. you cannot mix and match.

Each sprite can be set to one of the 15 available colours, if you want multi-coloured sprites then you need to place multiple sprites on top of each other e.g.



All 32 sprites can be on screen at the same time, but only the 1st four displayed on any row will show on screen i.e. the fifth sprite displayed on a row will not be drawn.

This will cause some of your sprites to disappear or flicker as they move past other sprites.

The example libraries that have been supplied already include code to allow up to 8 sprites per line, but changing the order the sprites are drawn each alternate frame.

The human eye will not even notice the minor flickering this introduces so this works quite well. But still means there is an upper limit of eight sprites in a row, so you need to take this into account when designing your games.

Let’s Make a Retro Game

Episode 15 – TMS9918A Graphics - Sprites

Using Sprites

TMS Sprites are quite easy to use, each of the 32 sprites are controlled using four bytes of video ram (and called the sprite control table) as follows:

Byte 1 – sprite Y position (0-255)

Byte 2 – sprite X position (0-255)

Byte 3 – sprite pattern number (0-255)

Byte 4 – sprite colour (0-15)

Rather than having to deal with reading and writing the sprite control table in video ram, I like to have a copy of the table in normal Z80 Ram, and copy it to video ram each vertical blank.

This also simplifies your code greatly as you don’t have to worry about when you can change a sprite value.

And with the supplied routines it also automatically allows 8 sprites per line by reversing the draw order of the sprites every 2nd frame as part of the routine to update video ram.

To make a sprite disappear from the screen i.e. not be drawn then it’s Y co-ordinate can be set to 209.

Let's Make a Retro Game

Episode 15 – TMS9918A Graphics - Sprites

On with Our Demo

Now let's extend the graphics demo from the last episode and have all 32 sprites on screen moving around so that we can see this in effect.

Sprite shapes

We are going to use 16x16 sprites for our demo and I have included four shape designs as follows:

We need to add these to our file ending with "-Patterns.asm", you can either cut and paste the code below or use the Sprite Editor to output the file from the supplied "TMSDemo.spr" file.

```
SPRITE_1:
; Sprite sprite_1 pattern 1
db 003,012,016,042,074,064,186,147
db 146,146,082,064,042,026,012,003
db 192,048,008,164,170,002,077,209
db 073,069,090,002,172,168,048,192
SPRITE_1:
; Sprite sprite_1 pattern 1
db 003,012,016,037,085,064,164,189
db 164,164,101,064,053,021,012,003
db 192,048,008,084,082,002,209,017
db 145,065,146,002,084,088,048,192
SPRITE_1:
; Sprite sprite_1 pattern 1
db 003,012,016,042,074,064,154,162
db 146,136,114,064,042,026,012,003
db 192,048,008,164,170,002,117,039
db 037,037,038,002,172,168,048,192
SPRITE_1:
; Sprite sprite_1 pattern 1
db 003,012,016,037,085,064,167,162
db 162,130,098,064,053,021,012,003
db 192,048,008,084,082,002,073,121
db 073,073,074,002,084,072,048,192
```

Let's Make a Retro Game

Episode 15 – TMS9918A Graphics - Sprites

Loading Sprite Patterns

Next, we need to load the sprite pattern data into Video Ram so that we can use them in our sprites.

```
; Send the sprite definitions to the VDP
LOAD_SPRITES:
    LD HL,VRAM_SPRGEN
    LD DE,SPRITE_1
    LD BC,32*SPRITECOUNT
    CALL LDIRVM
    RET
```

The TMS processor allows us to have 256 8x8 patterns that can be used for our sprites. When we use 16x16 sprites, each sprite needs four patterns that are one after each other, as you can only specify the pattern number of the 1st pattern. The TMS processor then assumes the other three patterns are the next three patterns after the one you specify.

In our main code file, we need to update the SPRITECOUNT value to equal the number of sprite patterns we have i.e. 4

```
SPRITECOUNT: EQU 4
```

And of course, we need to call the new routine above, which we might as well do just after we load the character tiles as follows:

```
    ; Load the character set, make all three sections the same
    CALL LOAD_CHR_SET
    ; Load our sprite patterns
    CALL LOAD_SPRITES
```

Draw Our Sprites

Next, we need to put our sprites on screen, so to start with let's display all 32 of them in a rough circle on the screen.

The easiest way to do this is to have a set of data and copy it to our sprite table, so add the following data section to our main code file:

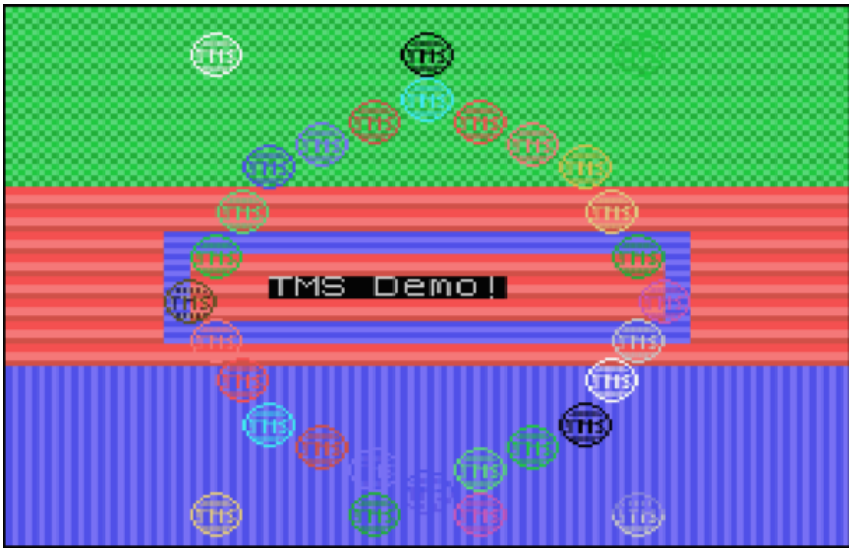
```
; Place our sprites on screen
PLACE_SPRITES:
    LD HL,SPRITE_PLACEMENT
    LD DE,SPRTBL
    LD BC,32*4
    LDIR
    RET

; Sprite placement data
SPRITE_PLACEMENT:
db 096,048,0,01
db 080,056,0,02
db 064,064,0,03
db 048,072,0,04
db 040,088,0,05
```

```
db 032,104,0,06
db 024,120,0,07
db 032,136,0,08
db 040,152,0,09
db 048,168,0,10
db 064,176,0,11
db 080,184,0,12
db 096,192,0,13
db 110,184,0,14
db 124,176,0,15
db 140,168,0,01
db 148,152,0,02
db 156,136,0,03
db 164,120,0,04
db 156,104,0,05
db 148,088,0,06
db 140,072,0,07
db 124,064,0,08
db 110,056,0,09
db 096,048,0,10
db 172,056,0,11
db 172,104,0,12
db 172,136,0,13
db 172,184,0,14
db 008,056,0,15
db 008,120,0,01
db 008,184,0,02
```

And after the CALL LOAD_SPRITES command above add a call to the placement routine as follows:

```
    ; place the sprites on the screen in their initial positions
    CALL PLACE_SPRITES
```

Build the code and run in the emulator and you should get something like this:



Let's Make a Retro Game

Episode 15 – TMS9918A Graphics - Sprites

Animate Our Sprites

Now, lets make things a bit more interesting and animate our sprites by making them cycle through a few different patterns a couple of times a second.

This is also a good example of how to use the timing code included in the sample templates.

In our template we have a section of code that will be executed two times a second, add the following code there, but change from the HalfSecTimer to the QtySecTimer i.e. we want our change to happen four times a second instead of two:

```
SPLASH_TITLE2:
    LD A,(QtrSecTimer)
    CALL TEST_SIGNAL
    OR A
    JR Z, SPLASH_TITLE2

    ; animate our sprite shapes
    LD HL,ANIMATION_TABLE
    LD B,0
    LD A,(ANIMATION_STEP)
    LD C,A
    ADC HL,BC
    LD A,(HL)
    CP 255
    JR NZ,UL1
    ; we have reached the end of our animation table
    XOR A
    LD (ANIMATION_STEP),A
    LD A,(ANIMATION_TABLE)
UL1:
    LD HL,SPRTBL+2
    LD B,32
UL2:
    LD (HL),A
    INC HL
    INC HL
    INC HL
    INC HL
    DJNZ UL2

    ; increment our animation step
    LD HL,ANIMATION_STEP
    INC (HL)

    JR SPLASH_TITLE2

ANIMATION_TABLE:
    DB 0,4,8,12,255
```

We need to declare the bit of Ram we are using for our ANIMATION_STEP variable as follows:

```
ORG RAMSTART

WAIT: ds 1
LASTPATTERN1: ds 1
LASTPATTERN2: ds 1
LASTPATTERN3: ds 1
ANIMATION_STEP: ds 1
```

And we must always remember to initialise any variables to a known value in our INITRAM function as follows:

```
INITRAM:
    LD A,204
    LD (LASTPATTERN1),A
    XOR A
    LD (LASTPATTERN2),A
    LD (ANIMATION_STEP),A
```

Build the rom and give it a go in the emulator, you should now see our sprites animating.

Let's Make a Retro Game

Episode 15 – TMS9918A Graphics - Sprites

Move Our Sprites

To finish off let's make the sprites move around the screen. To make things simple, we will make each sprite move in a random direction.

Now we need to add a data table to hold each of our sprite X & Y velocities i.e how much they are going to move and which way.

So, add a new RAM space (after ORG RAMSTART) as follows:

```
SPRITE_VELOCITY: DS 64
```

Now let's add a function to set our random X & Y velocities as follows:

```
; Set our sprite velocities
SET_VELOCITY:
    LD HL,SPRITE_VELOCITY
    LD B,32
SV1:
    CALL RND
    LD C,A
    AND %00000111
    INC A
    BIT 7,C
    LD (HL),A
    JR Z,SV2
    LD A,255
    SUB (HL)
    LD (HL),A
SV2:
    INC HL
    CALL RND
    AND %00000111
    INC A
    BIT 7,C
    LD (HL),A
    JR Z,SV3
    LD A,255
    SUB (HL)
    LD (HL),A
SV3:
    INC HL
    DJNZ SV1
    RET
```

Call this new routine, just after the CALL PLACE_SPRITES we added earlier as follows:

```
    CALL PLACE_SPRITES
    CALL SET_VELOCITY
```

Now let's add a function to move the sprites based on their current velocity:

```
; Move our sprites based in their velocity
MOVE_SPRITES:
    LD HL,SPRTBL
    LD DE,SPRITE_VELOCITY
    LD B,32
MS1:
    LD A,(DE)
    LD C,A
    LD A,(HL)
    ADD A,C
    LD (HL),A
    INC HL
    INC DE
    LD A,(DE)
    LD C,A
    LD A,(HL)
    ADD A,C
    LD (HL),A
    INC HL
    INC DE
    INC HL
    INC HL
    DJNZ MS1
    RET
```

And we call it just after our animation code from above as follows:

```
    LD HL,ANIMATION_STEP
    INC (HL)

    CALL MOVE_SPRITES

    JR SPLASH_TITLE2
```

If you build and run that you should see all 32 sprites moving in different directions and speeds.

You can play around with the original placement data and try modifying the code that sets the velocities to get different movement patterns.

That's all for this episode, next time we start covering sound, so we can add sound effects to our game.