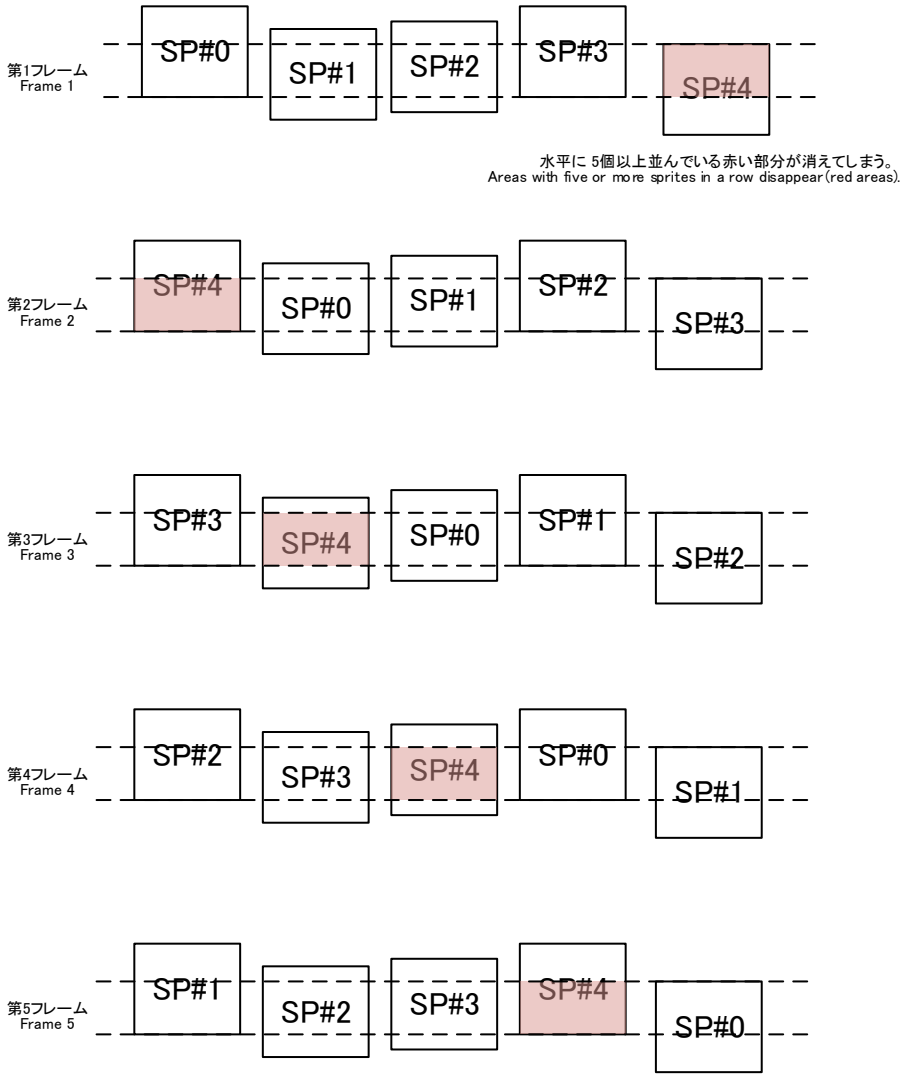
2023年2月1日 HRA!

【スプライトが 5枚以上並んだときに点滅表示で擬似的に 5枚以上見えるようにする方法】
【How to make 5 or more sprites look like 5 or more by blinking display when 5 or more sprites are lined up in a row.】

原理
theory



基本的な原理は下記の通りになります。

【原理】
左記の SP#n は、スプライトプレーン番号n のスプライトだと思ってください。
1/60秒単位の画面表示を1フレームと呼ぶことにすると、
この1フレーム毎に、各キャラクタに使うスプライトプレーン番号をコロコロ変えてやります。
すると、スプライトプレーン番号の若い番号順に表示の優先度が高いため、
コロコロ変えることによって、消えるスプライトが変化します。
これを利用することで、点滅して表示する状態を作り上げます。

では、どのようにすれば、「少ない演算でコロコロ入れ替えられるか？」が話の焦点になってきます。

「5個以上並んで消えてしまってるスプライトを調べる必要があるのでは？」とか、
「それって、画面上に複数ケースあり得るんじゃない？」とか、
「そもそも5個並んでるスプライトの上側と下側で、別の5個の組み合わせで並んでるケースもあるんじゃね？」とか、
考え出すと、なんか「探すこと自体がかなりの負荷のような気がしてきます。
なので、「いっそのこと、そんなの判定するのをやめてしまうことはできないか？」と考えてみてください。

そうです、表示中のスプライトを、毎フレーム「スプライトプレーン番号を変更して表示する」ことで、
自動的に並んでいる部分は点滅する、といった動作にすれば良いのです。
どのラインで5個以上並んでるか、なんて調べる必要は無いのです。

では、「シャッフルの演算」を 32枚も表示できるスプライト全てで演算するの？
シャッフルに乱数なんか使ったら、点滅の仕方が安定しないのでは？

とか思うかもしれませんが、素数を使えば単なる加算だけで何とかなります。

The basic theory is as follows.

【Theory】
Think of SP#n on the left as a sprite with sprite plane number n.
Let's call the screen display in 1/60 second increments a frame.
In each frame, the sprite plane number used for each character is changed from one to another.
The display priority is higher in the order of the sprite plane numbers, so the sprite planes that are used for each character are changed in order of priority.
The sprites that disappear will change as the sprite plane number is changed from one frame to the next.
By using this, we can create a state in which the sprites blink.

So, the focus of this discussion will be on the question, "How can we change sprites with a small number of operations?"

"Don't we need to check for sprites that are disappearing after more than 5 in a row?" Or
"Isn't it possible that there are multiple cases on the screen?" Or...
"Isn't it possible that the sprite that has 5 sprites in a row has another combination of 5 sprites on the top and bottom of the row?" And so on.
When you start to think about it, you start to feel that the search itself is quite burdensome.
So, "Can't we just stop judging such things?" And then think.

Yes, by "changing the sprite plane number" for each frame of the sprite being displayed, the sprite plane can be automatically re-numbered and displayed.
The sprite planes that are lined up automatically blink.
There is no need to check which line has more than 5 sprites.

Then, is the shuffle operation performed on all 32 sprites that can be displayed?
If we use random numbers for shuffling, won't the blinking be unstable?

But if you use prime numbers, you can manage it by simply adding them up.

【インデックスの総数の約数で無い素数】
【Prime number that is not a divisor of the total number of indices】

素数	19
第0フレーム	0
第1フレーム	19
第2フレーム	6
第3フレーム	25
第4フレーム	12
第5フレーム	31
第6フレーム	18
第7フレーム	5
第8フレーム	24
第9フレーム	11
第10フレーム	30
第11フレーム	17
第12フレーム	4
第13フレーム	23
第14フレーム	10
第15フレーム	29
第16フレーム	16
第17フレーム	3
第18フレーム	22
第19フレーム	9
第20フレーム	28
第21フレーム	15
第22フレーム	2
第23フレーム	21
第24フレーム	8
第25フレーム	27
第26フレーム	14
第27フレーム	1
第28フレーム	20
第29フレーム	7
第30フレーム	26
第31フレーム	13
第32フレーム	0

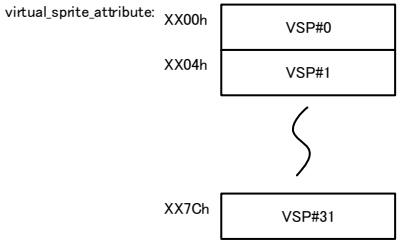
とある1つのキャラクターを表示するのに使うスプライトプレーン番号を、最初は SP#0 にしたとします。
その番号に次は、規定の素数(例えば 19)を、左記のようにフレーム毎に加算した番号に変えていきます。
スプライトプレーン番号は 0～31 の32通りなので、使う素数は 32の約数でない素数を選びます。なので 1や2はダメですね。
そういう素数を選ぶと、第32フレームで、第0フレームの値に戻ってきます。第0～第31フレームは毎回違う値になります。

Suppose the sprite plane number used to display a certain character is initially set to SP#0.
Next, the number is changed to a number that is added to a specified number of primes (e.g., 19) for each frame as shown on the left.
Since there are 32 different sprite plane numbers from 0 to 31, the prime number to use should be a prime number that is not a divisor of 32. So 1 or 2 is not allowed.
If you choose such a prime number, the 32nd frame will return to the value of the 0th frame. Frames 0-31 will have a different value each time.

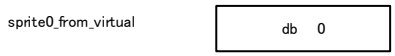
$$N(i+1) = MOD(N(i) + 19, 32)$$

【仮想スプライトアトリビュートテーブル】

DRAM上の、下位8bitが00hなアドレスから、スプライトアトリビュートテーブルと全く同じ並び（つまり、Y座標・X座標・パターン番号・色番号の4byte で一組。それが32組連なるテーブル。）の仮想スプライトアトリビュートテーブルを用意します。
この 中のスプライトプレーン番号を VSP#0～VSP#31 と呼ぶことにします。



VSP#0 を、本物の SP#n にコピーして使うわけですが、VRAM にはなるべく連続アクセスしたいので、「本物の SP#0 にコピーする VSP#n の n はどれか」を変数 sprite0_from_virtual に格納しておきます。初期値は 0 で良いです。



下記のコードで、VRAM に VSP#0 の内容を、SP#0 に転送できます。
（※実際は、連続する OUTI の間に適切なウェイトを入れてください。NOPとか。）

```
; VDPのアドレス設定は済ませておく
  LD    C, VDP_PORT0
  LD    A, [sprite0_from_virtual]
  LD    H, XXh
  LD    L, A
  OUTI
  OUTI
  OUTI
  OUTI
```

そして、素数（例えば 7）を加算して、VSP#7 に着目し、SP#1 へ転送します。
足したときに、VSP#32 とかにならないように、32*4-1 = 7Fh でマスクして循環させます。
+7 は VSP のインデックスとして加算するので、VSP 1つ分のサイズ 4 を掛けた値を足します。

```
  ADD  A, 7 * 4
  AND  A, 7Fh
  LD    L, A
  OUTI
  OUTI
  OUTI
  OUTI
```

使わないスプライトは、Y=200とか画面外にしておけばOKです。
これを、32回繰り返せば、前ページの理由により VSP#0～VSP#31 が全て処理されますし、VRAM上の SP#0～SP#31 には、VSP#0～VSP#31 とは異なる順序で格納されます。

Aの値が sprite0_from_virtual の値に戻ってくるので、ここに別の素数（例えば19）を加算して、sprite0_from_virtual に入れておきます。
そうすることで、次の更新時には、また違った順番にシャッフルされます。

```
  ADD   A, 19 * 4
  AND   A, 7Fh
  LD    [sprite0_from_virtual], A
```

【全貌】

```
    LD          HL, sprite_attribute_table | 0x4000
    LD          A, L
    OUT         [vdp_port1], A
    LD          A, H
    OUT         [vdp_port1], A

    LD          A, [sprite0_from_virtual]
    LD          B, 32
    LD          C, vdp_port0
    LD          H, virtual_sprite_attribute_table >> 8
loop:
    LD          L, A
    OUTI
    INC         B
    NOP
    OUTI
    INC         B
    NOP
    OUTI
    INC         B
    NOP
    OUTI
    INC         B
    NOP
    ADD         A, 7 * 4
    AND         A, 7Fh
    DJNZ        loop
    ADD         A, 19 * 4
    AND         A, 7Fh
    LD          [sprite0_from_virtual], A
    RET
```

VDPが画面の更新をしているタイミングで、表示している最中のスプライトのアトリビュートを変えるとテアリングが発生します。
VRAM に、sprite_attribute_table を表示用と更新用の2つ設置して、更新用に書き込んで、V-BLANKING で表示用と更新用を切り替えると、綺麗な表示になります。