Individual sprite 'boxes' for collision detection

by Achim

The regular way of detecting a sprite to sprite collision is to define a box around the sprite, which means you add an offset to the sprite's y and x values to get y2 and x2.

In most cases the 'one size fits all' approach will work, as long as all sprites in the game have a similar size. But with different sprite sizes (e.g. a big space ship and a small bullet) it's better to work with individual boxes for accurate results.

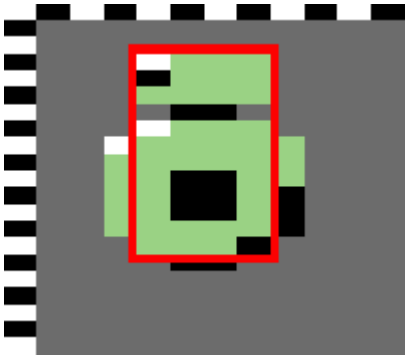
Setup

Set up sprite tables (for eight sprites):

```
spritey:    .byte    $00, $00, $00, $00, $00, $00, $00, $00
spritex:    .byte     $00, $00, $00, $00, $00, $00, $00, $00
spritemsb:  .byte    $00, $00, $00, $00, $00, $00, $00, $00
```

The main program will use these tables to move the sprites and the irq will read the values to update the VIC registers every frame.

Next step is an offset table with four offset values for figuring out y1, y2, x1 and x2. Here's an example sprite:



Spritey and spritex declare the upper left corner of the sprite.
y coordinates of red box: y1= y+3px and y2= y1+12px
x coordinates of red box: x1= x+8px and x2= x+16px (use x again to figure out x2 for easier msb handling.)

```
offsettable:    .byte    03, 12, 08, 16,        //the two y and two x offset values
                    ...                //and so on for all sprites...
```

Finally you need tables for y1, y2, x1 and x2:

```
spritey1:   .byte    $00, $00, $00, $00, $00, $00, $00, $00
spritex1:   .byte     $00, $00, $00, $00, $00, $00, $00, $00
spritemsb1: .byte    $00, $00, $00, $00, $00, $00, $00, $00
spritey2:   .byte    $00, $00, $00, $00, $00, $00, $00, $00
spritex2:   .byte     $00, $00, $00, $00, $00, $00, $00, $00
spritemsb2: .byte    $00, $00, $00, $00, $00, $00, $00, $00
```

Calculation

In this example the y-reg has to point at the correct position of 'offsettable' and x-reg at the correct position of 'spritey', 'spritey1' etc.

```
        lda spritey,x             //fetch y
        clc
        adc offsettable,y         //add first offset for y1
        sta spritey1,x
        adc offsettable+1,y       //and second offset for y2
        sta spritey2,x
        lda spritex,x
        clc
        adc offsettable+2,y
        sta spritex1,x
        lda spritemsb,x
        adc #$00
        sta spritemsb1,x
        lda spritex,x
        clc
        adc offsettabel+3,y
        sta spritex2,x
        lda spritemsb,x
        adc #$00
        sta spritemsb2,x
```

This has to be looped for all sprites.

Detection

The detection is similar to the 'one size fits all' approach mentioned above. Let's assume the first slot (in spritey1, spritey2 etc.) is reserved for the player's sprite, the other slots are used for enemy sprites.

```
    ldx #$00
loop:   lda spritey2
    cmp spritey1+1,x
    bcc skip        //player above enemy
    lda spritey2+1,x
    cmp spritey1
```

```
    bcc skip        //enemy above player
    lda spritex1+1,x
    sec
    sbc spritex2
    lda spritemsb1+1,x
    sbc spritemsb2
    bcs skip        //enemy on player's left
    lda spritex1        //player on enemy's right?
    sec
    sbc spritex2+1,x
    lda spritemsb1
    sbc spritemsb2+1,x
    bcc hitbysprite        //no, boxes hit each other
skip:   inx
    cpx #$07
    bne loop
    rts
hitbysprite:
    inc $d020
    rts
```

Using the 2*x trick makes life easier: Delete all msb tables, use 8bit additions to figure out x1 and x2 and use comparisons instead of subtractions for collision detection.