Character bullets

by Achim

Here're a few basic ideas on how to handle character bullets. Using character bullets is a bit annoying, but it's the easiest way to save sprites.

Preperation

First thing to do is to set up a lookup table that holds the screen addresses for each screen row. This will make the movements a lot easier.

```
;create lookup table for screen rows
    lda #$00
    sta tmp0        ;zp address
    sta tmp0+1        ;zp address+1
    sta rowlow        ;first row
    sta rowhi
    tax
lookup: lda tmp0
    clc
    adc #40
    sta rowlow+1,x
    sta tmp0
    bcc +
    inc tmp0+1
+   lda tmp0+1
    sta rowhi+1,x
    inx
    cpx #24             ;table for all 25 screen rows
    bne lookup

rowlow: !fill 25,0
rowhi:  !fill 25,0
```

'Rowhi' holds relative values (0 - 3) to make it versatile, e.g. for double buffering. A simple 'ora #screen_hibyte' will switch to the required screen buffer.

Now you can allocate x/y-coordinates to your bullets:

bullet_column = x, bullet_row = y

Set up two tables for x and y (columns and rows):

```
bullet_row: !fill 8,$ff    ; =y coordinate, 8 bullets, $ff=bullet not active
bullet_column:  !fill 8,0    ; =x coordinate
```

From now on you only have to manipulate the coordinates to move the bullets. Everytime your game

engine refreshes the screen data, use 'bullet_row' as an index to fetch the required screen address:

```
;x-reg pointing at current char bullet
    ldy bullet_row,x        ;fetch current screen position
    lda rowlow,y
    sta tmp0                     ;zp address
    lda rowhi,y
    ora #screen_screenhibyte    ;hi-byte screen buffer, e.g. $40 = screnn
buffer located at $4000
    sta tmp0+1                   ;zp address+1
    ldy bullet_column,x

    ;lda (tmp0),y            ;to read screen address... or
    ;sta (tmp0),y            ;to print on screen...
```

Examples

1. A simple example would be a game with following specs:

- one screen buffer
- no scrolling
- bullets moving only left or right

A procedure would look like this:

1. check if bullet can be printed on screen
2. save char underneath char-bullet
3. save colorRAM of original char
4. print char on screen and add color
5. determine bullet direction

So we need five tables:

```
bullet_row:     !fill 8,$ff    ;8 bullets, row=$ff --> bullet inactive
bullet_column:      !fill 8,0
bullet_direction:   !fill 8,0    ;e.g.: left=00, right<>00
charactertmp:       !fill 8,0    ;stored char underneath bullet
colortmp:       !fill 8,0    ;stored color of char underneath char bullet
```

In this case bullet_row=$ff means char bullet is not active and slot can be used. Each bullet is switched off by setting bullet_row to $ff.

print a new bullet on screen:

```
newbullet:          ;called if player presses fire
    ldx #7            ;maximum of 8 bullets
-   lda bullet_row,x    ;check if there's a free slot
    bmi setbullet
    dex
    bpl -
```

```
    rts

setbullet:
    lda player_row        ;precalculated screen row (=y-coordinate) of
player
    sta bullet_row,x    ;adjust this value if necessary, e.g. +/- 1 row
    lda player_column    ;precalculated screen column (=x-coordinate) of
player...
    sta bullet_column,x    ;...adjust this value if necessary, e.g. +/- 1
column
    ldy bullet_row,x    ;= y-coordinate
    lda rowlow,y        ;fetch corresponding screen address
    sta tmp0                ;zp address
    lda rowhi,y
    ora #screen_hibyte    ;hibyte of screen buffer
    sta tmp0+1

printbullet:
    ldy bullet_column,x    ;= x-coordinate
    lda (tmp0),y        ;read char underneath bullet...
    sta chararctertmp,x    ;...and store it
    lda #bullet        ;value of char bullet
    sta (tmp0),y        ;print bullet on screen
    lda tmp0+1        ;switch to colorRAM
    and #$03
    ora #$d8
    sta tmp0+1
    lda (tmp0),y        ;read color of original char
    sta colortmp,x        ;and store it
    lda #bulletcolor    ;color of char bullet
    sta (tmp0),y        ;set color
    rts
```

Moving the bullets:

```
    ldx #7
-   lda bullet_row,x
    bmi skip
    jsr move1bullet
skip    dex
    bpl -
    rts
move1bullet:
    ;restore char and color of orignal char first
    ldy bullet_row,x    ;= y-coordinate
    lda rowlow,y        ;fetch corresponding screen address
    sta tmp0
    lda rowhi,y
    ora #screen_hibyte    ;hibyte of screen buffer
    sta tmp0+1
```

```
    ldy bullet_column,x    ;= x-coordinate
    lda chararctertmp,x    ;restore original char
    sta (tmp0),y
    lda tmp0+1        ;switch to colorRAM
    and #$03
    ora #$d8
    sta tmp0+1
    lda colortmp,x        ;restore color
    sta (tmp0),y

;now move the bullet
    lda bullet_direction,x
    beq movebullet2left    ;00=move to the left
    inc bullet_column,x
    lda bullet_column,x
    cmp #40            ;hitting right border?
    bcc +
    lda #$ff        ;switch off bullet
    sta bullet_row,x
    rts
+   jmp printbullet        ;same as above...

movebullet2left:
    dec bullet_column,x
    bpl +
    lda #$ff        ;hitting left border?
    sta bullet_row,x    ;switch off bullet
    rts
+   jmp printbullet        ;same as above
```

Multiple directions

'Bullet_direction' can be used for at least 8 different directions. Idea would be to use the lower four bits for up (bit 0), down (bit 1), left (bit 2) and right (bit 3), similar to the joystick values you get from $dc00.

Moving a bullet in 8 directions could look like this:

```
;x-reg pointing at current bullet
movebulletup:
    lda bullet_direction,x
    and #1                ;bit 0 set?
    beq movebulletdown
    dec bullet_row,x
    bpl movebulletleft        ;hitting top border? if not check left & right
    lda #$ff
    sta bullet_row,x        ;then switch off bullet
    bmi movebulletleft        ;continue with left&right
movebulletdown:
```

```
    lda bullet_direction,x
    and #2                ;bit 1 set?
    beq movebulletleft
    inc bullet_row,x
    lda bullet_row,x
    cmp #25                ;hitting bottom border?
    bcc movebulletleft
    lda #$ff            ;then switch off bullet
    sta bullet_row,x        ;and continue with left&right
movebulletleft:
    lda bullet_direction,x
    and #4                ;bit 2 set?
    beq movebulletright
    dec bullet_column,x
    bpl +
    lda #$ff            ;hitting left border?
    sta bullet_row,x        ;switch off bullet
+   rts
movebulletright:
    lda bullet_direction,x
    and #8                ;bit 3 set?
    beq +
    inc bullet_column,x
    lda bullet_column,x
    cmp #40                ;hitting right border?
    bcc +
    lda #$ff            ;switch off bullet
    sta bullet_row,x
+   rts
```

Manipulate the coordinates like this and then again refresh the screen data ('printbullet', see above)

Scrolling screen

Scrolling the screen data automatically moves the char bullets as well, of course. The effect will be a line of char bullets on screen. So the game engine has to delete the char bullets left behind. This should be done when the hardscrolling is finished. Read the bullet tables to figue out which bullet is on screen. Adjust the coordinates according to the scrolling direction and restore the original char of the backdrop.