Freedirectional scrolling using map dumps

by Achim

Here's the traditional way of scrolling. It can be found in classic games like Uridium, Paradroid, Ghosts'n Goblins etc.

The idea is to fully decode the map data and store it into RAM. In terms of memory usage this is not very efficient, of course. But by defining an origin top/left of the decoded map data the actual screen data can be displayed directly onto the screen. No RAM-shifitng routines needed.

Therefore the map data has to be stored row by row. If one row is stored in one page, the map data can be up to 256 chars wide (=6 screens wide + another 16chars). For a wider map more pages have to be reserved.

```
Example:
1. row  = $8000
2. row  = $8100
3. row  = $8200
etc.
```

The height of the map data is defined by the number of pages reserved in memory. The more you reserve, the more rows can be scrolled up and down.

Once the map data is stored this way, a display routine copies the actual screen data into screen memory. Here's an example for 16 screen rows:

```
/*-----------------------------------------
Display routine
by A.Volkers, 2011
->Kick Assembler
-----------------------------------------*/

.pc = $0400 "screen memory" virtual
screen: .fill $100,0

.pc = $1500
.var map = $fe     //16bit map pointer

display:    ldx map+1    //hi-byte of decoded map data
        stx row1+2      //self modifying code...
        inx        //always add another inx
        stx row2+2    //if two pages are reserved
        inx        //for one row
        stx row3+2
        inx
        stx row4+2
        inx
        stx row5+2
        inx
```

```
        stx row6+2
        inx
        stx row7+2
        inx
        stx row8+2
        inx
        stx row9+2
        inx
        stx row10+2
        inx
        stx row11+2
        inx
        stx row12+2
        inx
        stx row13+2
        inx
        stx row14+2
        inx
        stx row15+2
        inx
        stx row16+2
        lda map        //lo-byte map data
        sta row1+1
        sta row2+1
        sta row3+1
        sta row4+1
        sta row5+1
        sta row6+1
        sta row7+1
        sta row8+1
        sta row9+1
        sta row10+1
        sta row11+1
        sta row12+1
        sta row13+1
        sta row14+1
        sta row15+1
        sta row16+1
        ldx #$27
row1:       lda $8000,x
        sta screen,x
row2:       lda $8100,x
        sta screen+40,x
row3:       lda $8200,x
        sta screen+80,x
row4:       lda $8300,x
        sta screen+120,x
row5:       lda $8400,x
        sta screen+160,x
```

```
row6:       lda $8500,x
        sta screen+200,x
row7:       lda $8600,x
        sta screen+240,x
row8:       lda $8700,x
        sta screen+280,x
row9:       lda $8800,x
        sta screen+320,x
row10:      lda $8900,x
        sta screen+360,x
row11:      lda $8a00,x
        sta screen+400,x
row12:      lda $8b00,x
        sta screen+440,x
row13:      lda $8c00,x
        sta screen+480,x
row14:      lda $8d00,x
        sta screen+520,x
row15:      lda $8e00,x
        sta screen+560,x
row16:      lda $8f00,x
        sta screen+600,x
        dex
        bpl row1
        rts
```

The example code uses “map” as a map pointer. For scrolling this pointer has to be manipulated everytime the ScrollY-bits ($d011) and the ScrollX-bits ($d016) wrap.

- Scrolling down/moving up: dec map+1 (page hi-byte)
- Scrolling up/moving down: inc map+1
- Scrolling right/moving left: dec map (page lo-byte)
- Scrolling left/moving right: inc map

That's it. This way of scrolling has got it's limits. A lot of memory has to be reserved for the map data. It's not very handy when colour RAM shifting is needed. But there're some benefits: Map data manipulations (like picking up objects etc.) are very easy to handle with fully decoded data. The code is rather small even with two screen buffers.

It should be mentioned that many games using this technique tend to call their display routine every frame in order to display map data changes immediately. Hence the number of scrolled screen rows is usually very limited to avoid timing problems. Timing problems occur when more than 16 screen rows have to be scrolled.

This example scrolls 16 rows freely. No colour shifting, no specific map data. Display routine located at $1500: mapdumbs.zip