MANUAL DEL USUARIO

ASÍ SE HIZO

Explicaciones del desarrollo

ANIMACIONES

MÁSCARA

DOBLE BUFFER

FUENTE

Lecciones aprendidas

MANUAL DEL USUARIO

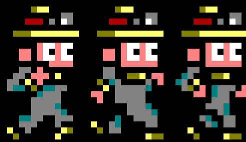
ASÍ SE HIZO

Explicaciones del desarrollo

ANIMACIONES

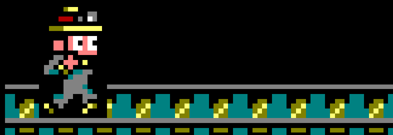
El sistema de animaciones que usamos es muy sencillo. Definimos unos frames que se irán recorriendo y cambiando según un contador de tics, por defecto para cada entidad distinta.

```
frames_minero::
 .dw #_spr_minerov2_1
 .dw #_spr_minerov2_2
 .dw #_spr_minerov2_3
```



MÁSCARA

El tener unos carriles en constante cambio, para que tuviera el efecto de que se estuvieran moviendo, y que hubiera un minero encima de ellos, generaba un problema, y es que, se dibujaba un cuadrado con las dimensiones del minero y hacía que el juego pareciera poco profesional:



Para ello, se implementaron máscaras para determinadas entidades.
La manera en la que se implementaron las máscaras fueron realizando operaciones AND y OR de la siguiente manera:

En la imagen se muestra que primero se hace un AND del sprite del fondo con la máscara, para obtener un sprite del fondo pero con la parte donde se rellena nuestro minero en negro (0x00), para que al hacer luego un OR del sprite resultante con el sprite del minero, quede reflejado el minero en la parte que quedó de negro en la máscara.

A continuación se muestran los siguientes sprites según frame:

No es la manera más eficiente de realizar las máscaras ya que habría que crear una máscara para cada sprite, pero durante el desarrollo se optó hacerlo de esa manera.

Esta misma técnica se realiza también para los trenes, pero se reutilizan las mismas máscaras.

El resultado de esta máscara es bastante bueno y es el siguiente:

DOBLE BUFFER

Luego, como definimos unas máscaras para dar una sensación de transparencia a los sprites, tenía un tiempo de cálculo para dichas operaciones. Por tanto se producía una sensación de parpadeo ya que no le daba tiempo en frames seguidos, pintar el sprite que queríamos. Para ello la solución fue crear un doble buffer, una segunda pantalla donde se dibujarán los sprites y resto de entidades. Este doble buffer se iría alternando en cada iteración del bucle, para dar una sensación de fluidez en los frames, ya que siempre habrá una pantalla que ya está dibujada y el raster no pilla el frame a medio dibujo.

Implementar este doble buffer tenía sus ventajas y desventajas, como perder 16KB de memoria a cambio de eliminar el parpadeo o mayor fluidez en el juego, pero valió la pena.

También fue todo un reto y fue bastante interesante implementar esta funcionalidad

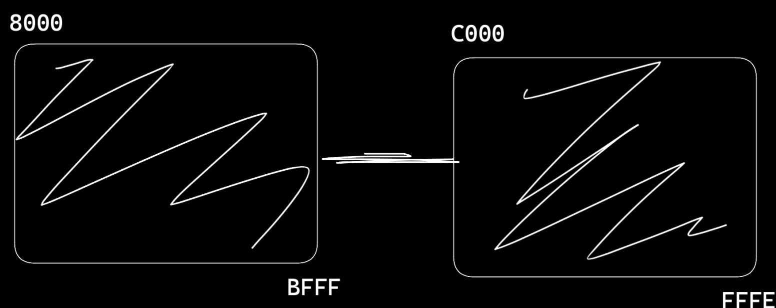
Para implementarlo, reservamos memoria para almacenar un puntero, que nos dirá dónde apunta la pantalla actualmente.

```
start_mem:
 .dw #0x8000
```

```
=============================================
0xC0 |     0xC000     |  cpct_pageC0     |  0x30
0x80 |     0x8000     |  cpct_page80     |  0x20
```

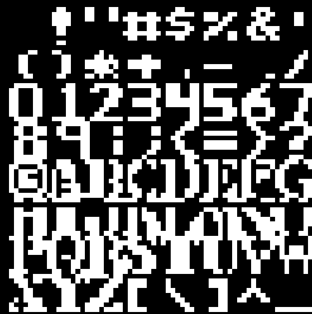
Y luego cambiamos la dirección de memoria de vídeo:

```
switch_buf::
 ld      hl, #start_mem+1
 ld        l, (hl)
 srl     l
 srl     l
 call   cpct_setVideoMemoryPage_asm

 ld      hl, #start_mem+1
 ld      a, #0x40
 xor     (hl)
 ld      (start_mem+1), a

ret
```



FUENTE

Para no depender de la función, *cpct_drawStringM0_asm,* ya que nos veíamos limitados a que el código no podía estar por debajo de 0x4000, hicimos nuestra propia fuente para el juego.



Al usar una fuente por nuestra cuenta, tuvimos que crear nuestra propia función para dibujar sprites, `_myDrawCharM0_inner y _myDrawStringM0`

Los cuales se ocupan de pintar el string que le pases como parámetro sin usar el ROM para los caracteres ni desactivar las interrupciones.

Lecciones aprendidas

Hemos aprendido mucho más acerca de las operaciones y la lógica que hay detrás de los lenguajes de programación de alto nivel, hay muchos conceptos que nosotros mismos pensábamos que entendíamos y nos hemos dado cuenta que con muchos conceptos nuestro modelo mental no era el real.

Especialmente nos ha parecido muy interesante y nos ha gustado el control absoluto de todo lo que ocurren en el programa/juego. Normalmente estamos acostumbrados a manejar la memoria a través de lenguajes de alto nivel, lo que hace que no tenemos el mismo mismo control y en muchas ocasiones no entendemos el funcionamiento real