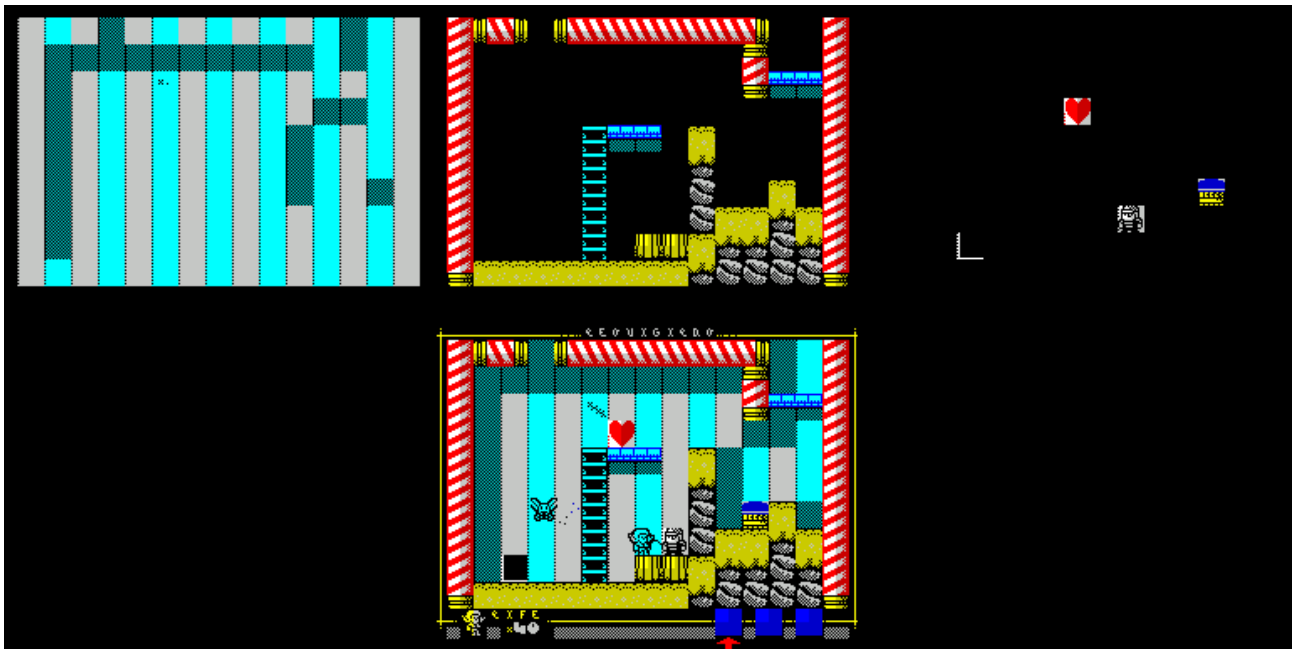
Construcción de pantallas en MK2

Este tutorial aplica a la versión 0.88c (o posteriores) de MK2

MK2 tiene un motor de construcción de pantallas bastante mejorado que nos permite obtener muy buenos resultados sin tener que usar un mapa "extendido" de 48 tiles. Para ello, se ha dividido la pantalla en tres planos lógicos: **fondo**, **mapa** y **decoraciones**. Por supuesto, puedes seguir usando mapas normales de 16 o 48 tiles exactamente igual que hacías en la Churrera (bueno, casi igual: ahora el conversor de mapas detectará si tu mapa es packed o unpacked automáticamente).

Para ver exactamente como funciona tomemos como ejemplo esta pantalla de la tercera carga de Leovigildo:



Fondo

Para usar un fondo, tendremos que dejar el tile 0 vacío y diseñar nuestro mapa teniendo en cuenta que el fondo se verá a través de las casillas donde pongamos dicho tile 0. Es como "el tile transparente".

El fondo se genera programáticamente modificando cierta sección de la función draw_scr_background que se define en el archivo /dev/engine/drawscr.h. Lo que se hace es detectar si el tile que hemos extraído del mapa vale 0, en cuyo caso se sustituye por un tile de fondo. Básicamente, se trata de añadir código en este trozo:

```
// CUSTOM {
// Construcción del fondo.
      if (0 == gpd) {
      }
// } END OF CUSTOM
```

Aquí podemos hacer prácticamente lo que nos de la gana. Usando gpx y gpy, que son las coordenadas del tile que se está dibujando, sólo tendremos que escribir en gpd el valor correcto. Por ejemplo, podemos usar un tilemap de 15x10 tiles con un fondo prediseñado, como se hace en Sir Ababol 2 48K, algo así (definido, por ejemplo, justo al principio de drawscr.h):

```
// Un tilemap de 15x10 tiles:

unsigned char backdrop [] = {
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,19,21, 0,
    0, 0, 0,19,21, 0, 0,19,21, 0, 0,19,20,20,21,
    0, 0,19,20,20,21,19,20,20,21,19,20,20,20,20,
    0,19,20,20,20,20,21,20,20,19,20,20,20,20,20,
   19,20,20,20,20,20,20,21,19,20,20,20,20,20,20
};
```

En cuyo caso, nuestro código de construcción del fondo sería algo tan sencillo como:

```
// CUSTOM {
// Construcción del fondo.
      if (0 == gpd) {
         if (gpy >= 5) {
            gpd = backdrop [gpx + gpy * 15];
         }
      }
// } END OF CUSTOM
```

Podemos complicar el tema todo lo que queramos, siempre que el resultado termine en gpd (que indica el número de tile que terminará pintandose, en lugar del 0). Por ejemplo, el mencionado Leovigildo III usa un código que pinta las franjas verticales, añade imperfecciones aleatoriamente, y coloca sombras para los tiles del mapa que sean obstáculos:

```
// CUSTOM {
// Construcción del fondo.
```

```
      if (0 == gpd) {
         gpd = (attr (gpx - 1, gpy) > 4 || attr (gpx, gpy - 1) > 4
|| attr (gpx - 1, gpy - 1) > 4) ? 22 : 18 + (gpx & 1) + (gpjt ?
0 : 22);
      }
// } END OF CUSTOM
```

Las posibilidades son infinitas.

Mapa

La capa del mapa se forma con los tiles que se extraen del mapa que has creado en Mappy, ni más ni menos. Si has añadido código para sustituir el tile 0 por un fondo, estos tiles no se dibujarán.

Si vamos a usar el nuevo motor de composición de pantallas, es tontería usar mapas de 48 tiles, ya que en estos podemos hacerlo todo directamente sobre el mapa. Con dibujar nuestro mapa en mappy utilizando únicamente los 16 primeros tiles, el conversor de mapas lo detectará y creará un mapa packed de 16 tiles.

Decoraciones

Las decoraciones son tiles extra que se imprimen sobre las dos anteriores capas al final del proceso. Esto puede hacerse por scripting o usando el nuevo módulo /dev/engine/extraprints.h si no estamos usando scripting en nuestro juego (como ocurre en la cuarta carga de Leovigildo).

En cualquier caso, no tendremos que introducir las decoraciones a mano, sino que el propio conversor de mapa las detectará en nuestro archivo .map y las exportará. Para ello sólo tenemos que pasarle el parámetro "forced". Si usamos este parámetro, cuando el conversor detecte un tile >= 16, en vez de exportar un mapa en formato "unpacked", ignorará esos tiles en el mapa exportado (que será, por tanto, "packed") y los irá añadiendo a una lista que posteriormente exportará separadamente:

```
..\utils\map2bin.exe ..\map\mapa.map 3 3 99 map.bin bolts.bin
force
```

Decoraciones, usando scripting

map2bin generará un archivo map.bin.spt (si hemos especificado que la salida se escriba en map.bin al invocarlo) **que deberemos copiar en la carpeta /script**, junto con nuestro script principal. El archivo map.bin.spt incluirá una sección ENTERING SCREEN xx para cada pantalla con decoraciones. Dentro, tendrá un bloque IF TRUE THEN ... END con el nuevo comando

DECORATIONS ... END en el que se define una lista de tiles que cambian. De este modo, cada decoración utiliza sólo 2 bytes del script (frente a los 4 que ocupaban en la Churrera o en MK2 < 0.8, por ejemplo), algo así:

```
[...]
ENTERING SCREEN 2
   IF TRUE
   THEN
      DECORATIONS
         7, 1, 16
         8, 1, 18
         7, 3, 22
         8, 3, 23
         7, 4, 25
         8, 4, 24
         1, 7, 27
         1, 8, 26
         7, 8, 19
         8, 8, 20
         9, 8, 20
         10, 8, 20
         11, 8, 21
      END
   END
END
[...]
```

(Cada linea de Decorations define X, Y, T para un tile de decoración y podemos usarlo nosotros en cualquier sitio del script que admita comandos - esto viene genial para modificar de un plumazo una parte tocha del escenario - es como una secuencia de SET_TILE (X, Y) = T).

Hecho esto, sólo tendremos que añadir esta linea al principio del script (en scripts para juegos multi-nivel, tendremos que añadirla al principio de la sección del script para el nivel correspondiente):

```
INC_DECORATIONS map.bin.spt
```

Cuando msc3 encuentre esta linea, incluirá el código generado por map2bin y almacenado en map.bin.spt en el script principal al principio de cada bloque ENTERING SCREEN.

Decoraciones, sin usar scripting

En el archivo map.bin.spt, al final, habrá código C con definiciones de arrays, algo así:

```
// If you use extraprints.h, trim this bit and use it!
const unsigned char ep_scr_00 [] = { 0x21, 16, 0x31, 17, 0x41, 18,
0xC1, 16, 0xD1, 18, 0x23, 22, 0x33, 23, 0x43, 22, 0xA3, 22, 0xB3,
23, 0x24, 25, 0x34, 24, 0xA4, 25, 0xB4, 24, 0x76, 28, 0x47, 29,
0x38, 19, 0x48, 21, 0xA8, 19, 0xB8, 20, 0xC8, 21 };
const unsigned char ep_scr_01 [] = { 0xC3, 18, 0x74, 16, 0x84, 18,
0x25, 27, 0xC5, 27, 0x26, 28, 0x76, 27, 0xC6, 26, 0x77, 29, 0xC7,
28, 0x88, 19, 0x98, 20, 0xA8, 20, 0xB8, 21 };
const unsigned char ep_scr_02 [] = { 0x71, 16, 0x81, 18, 0x73, 22,
0x83, 23, 0x74, 25, 0x84, 24, 0x17, 27, 0x18, 26, 0x78, 19, 0x88,
20, 0x98, 20, 0xA8, 20, 0xB8, 21 };
const unsigned char ep_scr_03 [] = { 0x42, 23, 0x33, 25, 0x43, 22,
0x53, 24, 0x73, 18, 0x44, 24, 0x84, 18, 0xA4, 16, 0x95, 17, 0x18,
19, 0x28, 21, 0x48, 19, 0x58, 20, 0x68, 21 };
const unsigned char *prints [] = { ep_scr_00, ep_scr_01,
ep_scr_02, ep_scr_03, 0, 0, 0, 0, 0 };
```

Si queremos usar decoraciones con juegos sin scripting, lo primero que tendremos que hacer es configurar esta característica en nuestro archivo /dev/config.h:

```
#define ENABLE_EXTRA_PRINTS
```

Una vez hecho esto, tendremos que recortar el código incluido en map.bin.spt y pegarlo al principio de /dev/engine/extraprints.h, justo donde pone...

```
// [[[ PASTE HERE THE OUTPUT AT THE END OF THE .SPT FILE THAT
MAP2BIN PRODUCES ]]]
```

Puedes ver extraprints.h en acción (y otras cosas que te ayudarán a crear videoaventuras simples sin scripting) en la cuarta carga de Leovigildo.