Construyendo un juego 128K con varios niveles (simple)

Este tutorial aplica a la versión 0.89 (o posteriores) de MK2

El tema de los juegos multi-nivel se da mucho a la personalización del motor. El archivo levels128.h incluido y el manejador clevels.h contienen dos ejemplos de cómo hacerlo:

- El modo "Goku Mal", en el que cada nivel es un ente compacto formado por mapa, cerrojos, comportamientos, enemigos, hotspots, tileset y spriteset contenido en un único binario.
- El modo "Ninjajar!", en el que cada nivel se compone de una serie de "assets" almacenados en la RAM extra que pueden combinarse de cualquier forma, además de incluir una estructura de secuencia de niveles por si queremos repetir un nivel varias veces (por ejemplo, las tiendas de Ninjajar!, que están todas en el mismo nivel real).

En este breve tutorial explicaremos como montar un juego de 128K con varios niveles con manejador de niveles sencillo (Goku Mal).

Configuración

En config.h tendremos que establecer ciertas directivas para lograr lo que queremos. Las primeras son las básicas, luego pongo alguna que otra más que viene bien (al menos para mí, a tí puede servirte otra cosa).

Hay que activar estas:

```
#define MODE_128K
#define COMPRESSED_LEVELS
```

Y creo que lo suyo es que, si usas scripting (¡¡porque usas scripting!!) activar esta otra:

```
#define SCRIPTED_GAME_ENDING
```

Fíjate en que NO activamos esta:

```
//#define EXTENDED_LEVELS
```

Esta lo que hace es meter toda la traya de Ninjajar! y por ahora vamos a dejar eso tranquilo. Vamos a crear un juego de manejador sencillo.

Construyendo el binario para cada nivel

Para construir el binario para cada nivel vamos a usar la utilidad buildlevel.exe que tenemos, como todas las demás, en el directorio /utils. Si lo ejecutamos a pelaco obtenemos los parámetros que toma, que son un buen tocho:

```
D:\Dropbox\nicanor\desarrollo\dev>..\utils\buildlevel.exe
buildlevel v 0.2 [MK2]
usage:

$ buildlevel mapa.map map_w map_h lock font.png work.png spriteset.png enems.ene scr_ini x_ini y_ini max_objs
enems_life behs.txt level.bin

where:
   * mapa.map is your map from mappy .map
   * map_w, map_h are map dimmensions in screens
   * lock is 15 to autodetect lock, 99 otherwise
   * font.png is a 256x16 file with 64 chars ascii 32-95
   * work.png is a 256x48 file with your 16x16 tiles
   * spriteset.png is a 256x32 file with your spriteset
   * enems.ene enems/hotspots directly from colocador.exe
   * scr_ini, scr_x, scr_y, max_objs, enems_life general level data header
   * behs.txt is a tile behaviours file
   * level.bin is the output filename.
   * decorations.spt if specified, maps are forced to 16 tiles + decorations
```

Nosotros vamos a emplear esta convención para no liarnos con los nombres de archivo (especialmente si usamos muchos niveles), siendo N el número del nivel:

- mapN.map : El mapa.
- workN.png : El tileset (256x48).
- spritesN.png : El spriteset (256x32).
- enemsN.ene : enemigos/hotspots.
- behsN.txt : Comportamientos de los tiles.
- levelN.bin : El archivo de salida.
- decorationsN.spt : Decoraciones para incluir en el script.

Vemos que necesitamos varias cosas, por tanto:

- Mapa, tileset, spriteset, enemigos, como siempre.
- Comportamientos: simplemente una lista ordenada de 48 valores, como el que solíamos poner en config.h, algo así:

```
0,24, 8, 8, 8, 8, 8, 8, 1, 1, 8, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

- font.png: Es un archivo de 256x16 con los 64 caracteres de la fuente de texto (ascii 32-97). Es un poco desperdiciar memoria, sobre todo si vamos a usar siempre la misma fuente (que es lo suyo), pero por ahora no hay otra forma de hacer un bundle omitiendo cierta parte del charset (la correspondiente a la fuente). Tampoco supondrá mucho espacio desperdiciado.

Una vez generado nuestro binario y nuestro archivo de decoraciones (si aplica), tendremos que comprimir el binario y ponerlo en /bin para el librarian y mover el archivo de decoraciones a /script para su posterior inclusión en el script principal.

Como los comandos buildlevel son muy largos y coñazos de escribir, y como tendremos que generar los niveles un montón de veces durante el desarrollo, vamos a crearnos un buildlevels.bat en /levels con las llamadas a este comando para cada nivel.

```
@echo off
echo BUILDING LEVELS!

echo LEVEL 0
..\utils\buildlevel.exe ..\map\map0.map 4 4 99 ..\gfx\font.png ..\gfx\work0.png ..\gfx\sprites0.png
..\enems\enems0.ene 0 2 7 99 1 behs0.txt level0.bin decorations0.spt
..\utils\apack.exe level0.bin ..\bin\level0c.bin
move decorations0.spt ..\script

... (etc)
```

Como véis, al comprimir levelN.bin lo generamos directamente en \bin con el nombre de levelNc.bin

Modificaciones a make.bat

En primer lugar habrá que quitar todo lo que sea construir el mapa, exportar enemigos, gráficos, etcétera e incluir una llamada a nuestro nuevo buildlevels.bat que debería ser algo así:

```
cd ..\levels
call buildlevels.bat
cd ..\dev
```

Ponemos call para que cuando termine buildlevels.bat siga la ejecución de make.bat.

Librarian

Supuestamente ya tendríamos que tener lo básico para el librarian. Recuerda que si tienes binarios pre-generados para alguna página específica de RAM (como, por ejemplo, textos y script) debes colocarlos en un archivo preloadN.bin donde N es la página de RAM donde deben ir para que librarian los tome en cuenta. A la lista de archivos binarios /bin/list.txt añadimos nuestras fases:

```
title.bin
marco.bin
ending.bin
level0c.bin
...
```

Según esta configuración, la fase N-esima estará en el recurso N+2-ésimo: la primera fase en el recurso 3, la segunda en el 4...

El script

Esto se merece si propio tutorial, pero lo mencionaremos aquí: tu archivo de script contendrá scripts para todos los niveles. Estos scripts se separan en el archivo con END_OF_LEVEL:

```
### level 0

ENTERING GAME
    IF TRUE
    THEN
       ...
    END
    ...
END

...

END_OF_LEVEL

### level 1

ENTERING GAME
    IF TRUE
    THEN
       ...
    END
    ...
END

...

END_OF_LEVEL

...
```

Ahora hay que decirle al motor dónde encontrar el script para cada nivel. msc3.exe, en el archivo msc-config.h que genera, crea unas cuantas constantes que nos vendrán muy bien:

```
#define SCRIPT_0 0x0000
#define SCRIPT_1 0x0179
#define SCRIPT_2 0x01CD
...
```

Además, esta linea de make.bat:

```
..\utils\sizeof.exe ..\bin\texts.bin 49152 "#define SCRIPT_INIT" >> msc-config.h
```

Añade otra constante SCRIPT_INIT que nos indica dónde empiezan los scripts dentro de la memoria. Así, tendremos que el primer script empieza en SCRIPT_INIT + SCRIPT_0, y así sucesivamente. Pues bien, estos son los valores que tenemos que poner en el array de niveles que aparece al final de levels128.h:

```
LEVEL levels [] = {
   {3, 3, SCRIPT_INIT + SCRIPT_0},
   {4, 4, SCRIPT_INIT + SCRIPT_1},
   ...
};
```

- El primer número es el número de recurso de librarian que contiene el bundle del nivel que sea. Como ves, empezamos en el 3.
- El segundo número es el número de la música dentro de la OGT que debe sonar de fondow.
- Por último, tenemos la expresión que calcula dónde empieza el script concreto que hay que ejecutar para cada fase.

Con esto y un bizcocho, el tema debería funcionar medio qué. Y si no, como es complicao la primera vez (y la segunda), siempre está el foro para ir haciéndolo despacito y pasito a pasito.