UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Generación avanzada de escenarios mediante código y mapas de teselas para videojuegos sobre NDS

Apellidos, nombre	**Agustí Melchor, Manuel** (magusti@disca.upv.es)
Departamento	**Dpto. de Ing. De Sistemas y Computadores (DISCA)**
Centro	**Escola Tècnica Superior d'Enginyeria Informàtica** Universitat Politècnica de València

1 Resumen de las ideas clave

En este artículo vamos a abordar cómo se puede implementar un escenario de videojuego en la *Nintendo DS* (NDS), a partir de unas pequeñas imágenes de forma cuadrangular o **teselas**. Esto permite construir una imagen compleja que va a ser utilizada como un escenario para un videojuego en una plataforma con un consumo bajo de recursos.

Estos escenarios, se crean a partir de imágenes que, en los casos complejos, no pueden ser fijas o "precocinadas". Esto es, no son creadas con anticipación, puesto que parte de su contenido depende del instante de ejecución, de la acciones del usuario y, por limitaciones del *hardware* o por optimizar recursos, hay partes de su contenido que se repiten, lo que ayuda a mantener bajo el consumo de recursos y ofrecer una experiencia de usuario fluida. Así, la Figura 1a muestra una salida en las pantallas de la NDS, construida a partir de las dos imágenes de la Figura 1b: una para el mapa de caracteres y la otra para el escenario del pequeño castillo. Las proporciones de la figura son para hacernos una idea de cuanto se puede poner en las pantallas de la NDS utilizando la técnica del teselado (*tiling*).

a) *b)*

Figura 1: Situación que quiere generar: (a) resultado y (b) elementos mínimos para reproducirla.

Para facilitar el seguimiento de este trabajo se han dispuesto todos los elementos del proyecto que aquí se comenta en *GitHub* [1], donde encontraremos todos los elementos para llevar a cabo uno de los ejemplos de la interesante guía de *P. Rising* [2] sobre desarrollo *homebrew*: el que se realiza sin las herramientas oficiales, puesto que estas son de carácter propietario y sujetas a restricciones de publicidad y uso. Aunque el trabajo original de *P. Rising* sigue siendo muy bueno, no se pueden aplicar buena parte de los ejemplos de código; esto es porque han habido cambios importantes en las librerías en que se basa. Así que **nuestra aportación será** actualizar el modo de proceder que se

describe en este tutorial en cuanto al código. No esperes una traducción, es mi propia realización ampliando los detalles de la explicación original y modificando partes del código.

2 Objetivos

Una vez que el lector haya leído con detenimiento este documento, será capaz de:

- Identificar los conceptos básicos de desarrollo de escenarios para videojuegos en la consola NDS, a partir de imágenes o texturas.
- Seguir un ejemplo de código que permite generar un escenario a partir de un número mínimo de componentes gráficos en forma de imágenes en mapa de bits.

Para ver estos objetivos vamos a seguir el tutorial de *P. Rising* [2], bueno solo una parte de su extensa guía de desarrollo de aplicaciones para la NDS: es la parte que dedica al uso de teselas (*tiles*).

No es un objetivo describir el uso de emuladores para ejecutar las aplicaciones para la NDS, como *DeSmuMe* [3]. Así como tampoco instalar las herramientas que permiten la creación del ejecutable o la carga del mismo en la consola, para ello se puede recurrir a los trabajos de [4] y [5]. Tampoco es un objetivo de este trabajo entrar a ver las características de la NDS. Si el lector tiene curiosidad o necesidad de profundizar en estas cuestiones es recomendable consultar [6].

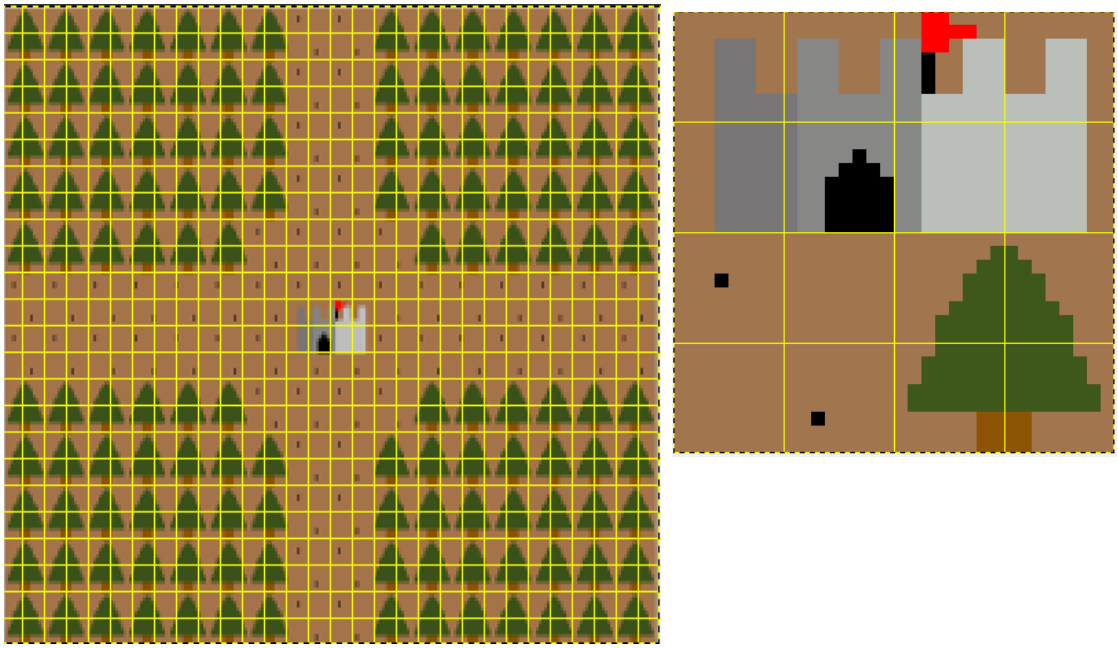
3 Introducción

En el caso de la NDS, las casillas o **teselas** son de 8x8 píxeles (aunque también se puede trabajar con algunos múltiplos, como 16x16, 32x32 y 64x64) y con una paleta de colores asociada. La resolución de la pantalla de es 240x192 píxeles, por lo que son necesarias 32 filas x 24 columnas de teselas para rellenar la pantalla. Así que podemos pensar en el resultado que aparecía en la Figura 1a como el que se obtiene al asignar a las celdas, de 8x8 píxeles, de una **rejilla** (grid) imaginaria (que se puede ver en color amarillo sobrepuesta a la imagen en la Figura 2a y que se ha generado a partir de las 16 teselas de 8x8 píxeles que se muestran en la Figura 2b).

La relación entre la imagen que contiene todas las teselas y la imagen a componer es lo que se denomina un **mapa**, una estructura de datos que describe en cada posible casilla de la imagen resultante (denominadas *screen blocks*), qué tesela (o también denominadas *character blocks*) del posible repertorio (catálogo o *texture sheet*) se utilizará, referenciándolas por un número de orden en ese posible repertorio. Y lo mejor, que con solo cambiar el índice de la tesela a dibujar en cada casilla, será el *hardware* de la NDS la que actualizará el contenido de la pantalla al instante.

Cada pantalla de la NDS puede mostrar, véase la Figura 3, hasta cuatro capas (*backgrounds* o *BG*) que se nombran como BG0 a BG3 (para una

pantalla) y SUB_BG0 hasta SUB_BG3 (para la otra). En la Figura 3[1] se puede ver una posible asignación de la correspondencia entre las pantallas y la memoria de vídeo de la NDS (VRAM) que es la que desarrollaremos a continuación. Para dibujar en las pantallas de la NDS habrá que llevar las teselas a la VRAM y asignar en el mapa (para cada pantalla y el BG correspondiente) el índice de la tesela en VRAM que va en cada una de las 32x24 casillas en que se divide el BG.



a) b)

Figura 2: Imagen y conjunto de teselas que la generan para el caso de la NDS, con teselas de 8x8 píxeles: (a) rejilla sobrepuesta y (b) catálogo de teselas de 8x8 píxeles.

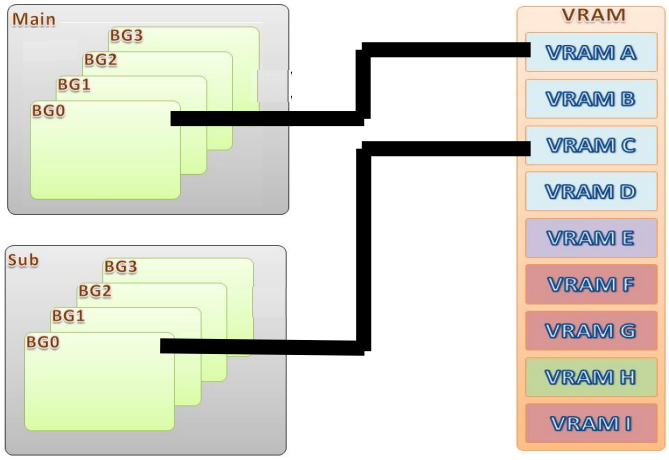


Figura 3: Ejemplo de asignación de capas o fondos a bancos de memoria de la VRAM que será utilizado en este documento.

[1] La imagen de la está basada en la de <http://4.bp.blogspot.com/_Vtd_Vdw1epI/R8f9k_A7R6I/AAAAAAAAAEs/piMMqJN18zs/s1600-h/nds_dg1.jpg >.

4 Estructura y contenido del proyecto

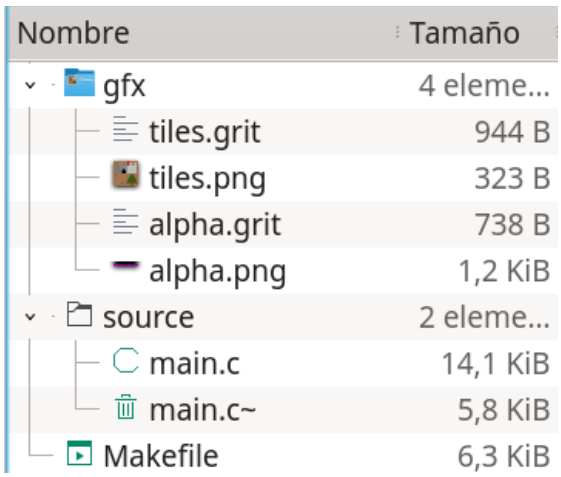
El ejemplo que estamos reconstruyendo va a mostrar un mensaje de texto en una pantalla de la NDS con un tipo de letra que vamos a diseñar y un escenario de un bosque con un castillo en el centro. Para ello, necesitamos una imagen con el conjunto de caracteres que vamos a utilizar y otra con los elementos del escenario. Vamos a suponer que estos elementos ya están construidos, el lector los tiene disponibles en el repositorio de *GitHub* [1]. Aquí nos vamos a centrar en la implementación de código que hace posible su uso, pero primero veamos cómo se incluyen las imágenes en el proyecto.

4.1 Las imágenes

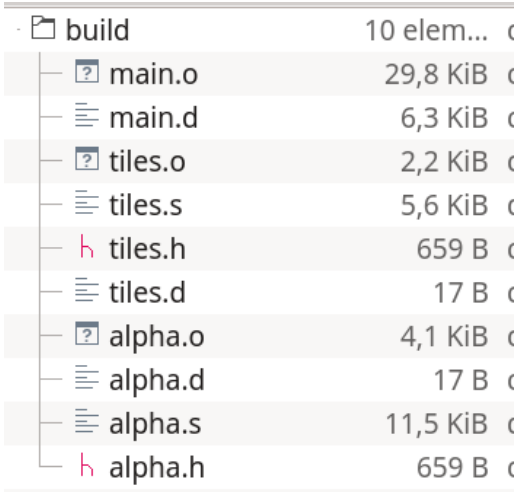
Para utilizar las imágenes en la NDS, una posibilidad es convertir estas a código. Así, dadas las dos imágenes que se describen en la Figura 4a, las ubicaremos en el directorio *gfx* (como se muestra en la Figura 4b), donde aparecen junto a unos ficheros, de extensión ".grit" que son ficheros de texto que contienen las instrucciones para dar forma a esta conversión de formatos.

```
$ file  gfx/*.png
gfx/alpha.png: PNG image data, 760 x 8, 8-bit/color RGBA, non-interlaced
gfx/tiles.png: PNG image data, 32 x 32, 8-bit/color RGB, non-interlaced
```

a)



b)



c)

Figura 4: El proyecto que desarrollamos(disponible en [1]): (a) características de las imágenes, (b) estructura de partida, (c) directorio intermedio.

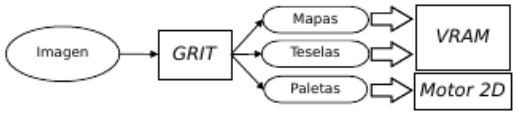
Para incorporar las imágenes al proyecto utilizaremos una aplicación, GRIT [7], que se encargará de ello: aplicando a cada imagen el conjunto de acciones que se recogen en el fichero del mismo nombre que la imagen y con extensión ".grit". Como resultado de la ejecución de GRIT sobre cada imagen, se generarán unos archivos intermedios en un directorio que se denomina "build", Figura 4c. Allí se generarán las versiones en código ensamblador de las imágenes en forma de

estructuras de datos que pueden ser accedidas desde el código C del programa.

Vemos los contenidos de los archivos ".grit" en la Figura 5. Ambos podrían haber tenido los mismos valores, pero como el mapa de caracteres tiene menos colores podemos optimizarlo un poco más y por eso tiene los parámetros que se especifican.

```
$ cat gfx/alpha.grit
# graphics in tile format
-gt
# output first 16 colors of the palette
-pw16
# no tile reduction
-mR!
# no map output
-m!
# graphics bit depth is 4 (16 color)
-gB4
$
$ cat gfx/tiles.grit
# -gB{n} specifies the output graphic format: 8-> paleta de 256 colors
-gB8
```

a)



b)

Figura 5: Ficheros GRIT: (a) contenido para las imágenes del proyecto y (b) elementos generados con ellos (fuente [6])

En el caso de la imagen para la generación del escenario vemos que solo se ha especificado el "-gB8" para configurar que la imagen utiliza 8 bits por pixel (bpp). ¿Dónde han ido a parar el resto de parámetros? Simplemente no se han especificado porque son los valores por defecto:

- "-p" que indicaría que se ha de generar la paleta de color.
- "-gt" que indicaría que se ha de tratar como una imagen de teselas.
- "-m!" indica que no se ha de generar un mapa (lo haremos en el código más tarde).

Ambas imágenes que componen el proyecto son descompuestas en sus correspondientes texturas, paleta y mapa, Figura 5b, a partir de la información en los ficheros GRIT. Además para poder ser referenciadas como teselas independientes deben estar copiadas en unas determinadas áreas de memoria VRAM de la NDS y configurado el motor gráfico para ello, veremos esto cómo se hace en el apartado 5.

4.2 El uso de las teselas

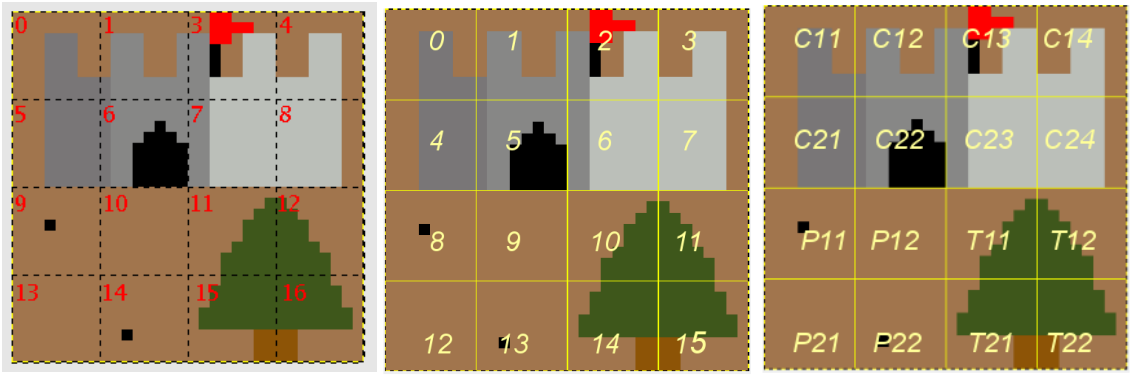
El ejemplo en el que se basa este artículo utiliza los dos conjuntos de teselas: el mapa de caracteres para escribir en una de las pantallas y el mapa de teselas del nivel, para dibujar el escenario sobre el que se moverían los personajes.

La numeración de las teselas del mapa de caracteres es unidimensional, la Figura 6 muestra los primeros veintiún caracteres de este mapa. Para indexarlos se utilizará el código ASCII de cada carácter, tomando como inicial el espacio en blanco que es el número treinta y dos del código ASCII y, como los anteriores no son imprimibles, utilizamos este como el primero (con el índice cero) de esta lista.

Figura 6: Numeración de las teselas del mapa de caracteres.

Para el caso del escenario, se utiliza una numeración por filas y columnas, como muestra la Figura 7. El tutorial original asigna los valores en rojo de la Figura 7a, que como podemos apreciar se ha saltado el número '2', por lo que en Figura 7b hemos corregido la numeración. Para generar el mapa del escenario se utilizará una estructura de datos que recogerá (con la nomenclatura de la Figura 7c), qué índice de estas dieciséis teselas le corresponde a cada casilla de la pantalla.

En el uso de este escenario, para permitir el movimiento de los personajes, el índice de tesela se puede utilizar para decidir cuando se puede avanzar por una casilla (casilla "libre"), cuando está sobre un objeto recolectable o cuando "tropieza" y no puede realizar el movimiento.



a) *b)* *c)*

Figura 7: Numeración de las teselas del escenario: (a) original [2], (b) numeración corregida y (c) nomenclatura empleada en el código.

Hemos introducido este y algún cambio más, para asemejar más el código a la forma de proceder actual de las herramientas que se utilizan para estos desarrollos. Así que vamos a echarle un vistazo al código y comentaremos algunos detalles del mismo, especialmente lo que hemos modificado.

5 El código del ejemplo práctico revisado

El primer fragmento del código que muestra la Figura 8, contiene la mayor parte de declaraciones y la función que carga el mapa de teselas del único nivel de que consta este ejemplo.

```
1       /*  Recreació de "Homebrew Programmers
2       Guide to the Nintendo DS 0.1 . Part II: Graphics Basisc. Using Tiles"
 <http://dspassme.jzdocs.com/programmers_guide/tutorial/using_tiles.html>  */
3       #include <nds.h>
4       #include <stdio.h>
5
6       #include "alpha.h" // La versión original no utilizaba GRIT i tenía que  poner
7       #include "tiles.h"   // aquí las estructuras de datos directamente.
8
9       #define C11  0
10      #define C12  1
11      #define C13  2
12      #define C14  3
13      #define C21  4
14      #define C22  5
15      #define C23  6
16      #define C24  7
17      #define P11  8
18      #define P12  9
19      #define T11 10
20      #define T12 11
21      #define P21 12
22      #define P22 13
23      #define T21 14
24      #define T22 15
25
26      void drawMap( u16* map ) // Cambios respecto a la versión original
27      {
28       u16 original_toDraw[] = {
29       // El tipo de letra es intencionadamente menor para que quepa
30          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
31          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
32          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
33          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
34          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
35          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
36          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
37          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
38          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
39          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
40          P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,
41          P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,C11,C12,C13,C14,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,
42          P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,C21,C22,C23,C24,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,P11,P12,
43          P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,P21,P22,
44          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
45          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
46          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
47          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
48          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
49          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
50          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
51          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
52          T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,P11,P12,P11,P12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,T11,T12,
53          T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,P21,P22,P21,P22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,T21,T22,
54       };
55
56       for(u16 i = 0; i < sizeof(toDraw) / sizeof(u16); ++i) {
57         map[i] = original_toDraw[i];
58       }
59      } // drawMap
60      ...
```

Figura 8: Código del ejemplo, parte 1 . Partes del código extraído de [2]

Los que hayan visto la película *Matrix*[2], seguro que ya pueden ver en la variable *original__toDraw* el escenario del pequeño castillo, ¿a que sí? ;-). ¡Correcto, es la pantalla de abajo de la Figura 1a! Se define de manera estática y dentro del código por brevedad. No lo vamos a poner en letra más grande que se come mucho espacio, el código está en el repositorio de este proyecto en *Github* [1].

```
61      ...
62      int main( void ) {
63        unsigned int i, j;
64        const int char_base = 0;
65        const int screen_base = 20;
66        int       elFondo;
67
68        lcdMainOnBottom();
69
70        vramSetBankA(VRAM_A_MAIN_BG);
71        videoSetMode( MODE_0_2D | DISPLAY_BG0_ACTIVE );
72        elFondo = bgInit(0,                                    // índice del BG
73                    BgType_Text8bpp, BgSize_T_256x256,   // tipo i tamany
74                    screen_base,                         // map
75                    char_base);                          // tile
76        u16* tile = (u16*)CHAR_BASE_BLOCK(char_base);
77        u16* map = (u16*)SCREEN_BASE_BLOCK(screen_base);
78        //Cargar teseles
79        dmaCopy(tilesTiles,  bgGetGfxPtr(elFondo), tilesTilesLen);
80        //Carrgar las paletas
81        dmaCopy(tilesPal,  BG_PALETTE, tilesPalLen);
82        // Draw the forest scene: copiar els gràfics a la VRAM
83        drawMap( bgGetMapPtr(elFondo) );
84      ...
```

Figura 9: Código del ejemplo, parte 2 . Partes del código extraído de [2].

En la Figura 9 vemos como el tutorial original había activado la memoria de vídeo asociada a cada pantalla, como se había indicado en la Figura 3, con las instrucciones *vramSetBank* y *videoSetMode (líneas 69 y 70)*. Y, a diferencia del ejemplo original que accede a los registros directamente, aquí por portabilidad, legibilidad y actualización a las librerías hemos utilizado:

- Línea 68, en el tutorial original aparece abajo el texto y arriba el dibujo del escenario. Aquí hemos utilizado *lcdMainOnBottom*, para especificar dónde queremos; si no nos gusta podríamos especificar *lcdMainOnTop* para escoger la otra pantalla de la NDS.
- Línea 72, la nueva función *bgInit*, para escoger el BG0 (recordemos que habíamos indicado esta elección de la capa 0 ya en la Figura 3) al motor gráfico principal, que es el encargado de pintar la pantalla que alberga el escenario. Esta función agrupa los parámetros para configurar esta capa del motor principal, con el valor 0 para indicar el número de BG, el tipo y tamaño del BG y la dirección del mapa y de las teselas.

[2] Véase la descripción de la misma en <https://es.wikipedia.org/wiki/The_Matrix >.

- Observamos también aquí cómo utilizamos el DMA para cargar las teselas (línea 79), las paletas (línea 81) y el mapa (línea 83) en las zonas de memoria correspondientes. La función *drawMap,* en nuestra versión, recibe el puntero a la zona de memoria donde ha de llevarla.

Y, en la Figura 10, veremos que para asignar el fondo al segundo motor, en lugar de utilizar *bgInitSub*, (al estilo de la otra pantalla que hemos generado ya) se utilizan las instrucciones de *consoleInit* y *consoleSetFont* (líneas 88 y 101, respectivamente), para inicializar la pantalla en modo texto, con nuestro propio mapa de caracteres. A partir de entonces los mensajes que aparecen en pantalla con cualquier *printf*, como en la línea 114.

```
85      ...
86       vramSetBankC(VRAM_C_SUB_BG);
87       videoSetModeSub( MODE_0_2D | DISPLAY_BG0_ACTIVE );
88       PrintConsole *pantallaMsg = consoleInit(0,0,
89                                    BgType_Text4bpp, BgSize_T_256x256,
90                                    screen_base, char_base,
91                                    false, false); // CORE_SUB, NO_FONT_PER_DEFECTE
92       // Asignar el mapa de caracteres
93              ConsoleFont font;
94              font.gfx = (u16*)alphaTiles;
95              font.pal = (u16*)alphaPal;
96              font.numChars = 95;
97              font.numColors =  alphaPalLen / 2;
98              font.bpp = 4;
99              font.asciiOffset = 32;
100             font.convertSingleColor = false;
101             consoleSetFont(pantallaMsg, &font);
102      // Texto para mensajes en pantalla
103      char* msg[] = {
104       "",
105       "Tile Demo.",
106       "tile01 project.",
107       "",
108       "http://www.dspassme.com/programmers_guide/",
109       "http://dspassme.jzdocs.com/programmers_guide/tutorial/", // Este sí
110       "",
111       "PhoenixRising",
112      };
113       for(i = 0; i < 8; ++i) {
114              printf("%s\n", msg[i]);
115       }
116       while(1) {
117                      swiWaitForVBlank();
118                      scanKeys();
119                      int keys = keysDown();
120                      if(keys & KEY_START) break;
121              }
122      return 0;
123     }
```

Figura 10: Código del ejemplo, parte 3 . Partes del código extraído de [2].

Y, por último, las líneas 115 a 120 muestran el típico bucle de espera a que se pulse una tecla, que aquí no es realmente preciso porque ya no

se hace nada más, pero creo que hace el código más claro, actualizado y abre la puerta a introducir tus propias modificaciones.

6 Conclusiones y cierre

Con este artículo hemos vuelto a poner a disposición de los interesados en el desarrollo de aplicaciones *homebrew* para la NDS (las que utilizan el kit de desarrollo no oficial) un recurso que creemos es de utilidad para servir de apoyo a los que se inician en este complejo contexto de desarrollo de aplicaciones para plataformas de videoconsolas. Las actualizaciones necesarias del código y la ubicación en un repositorio en *GitHub* son nuestra modesta aportación. Quiero agradecer desde aquí el trabajo del autor original del artículo, así como la intervención de J. D. Jaén en la anterior revisión y ampliación del mismo. Sirva este trabajo como homenaje a estos desarrolladores que han contribuido con su código y con sus explicaciones a que otros puedan adentrarse en este apasionante y difícil campo de desarrollo de aplicaciones para computadores con una arquitectura tan especializada y poco documentada.

Esperamos que el lector se anime a descargar el proyecto desde el *GitHub* [1], leer el original [2] y probar algunas modificaciones. Con el emulador *DeSmuME* [3] es posible ejecutar los desarrollos expuestos. Una sugerencia, después de la línea 100 de la Figura 10, para cambiar el color del texto y el fondo, no diremos lo que resulta, podemos probar con:

```
BG_PALETTE_SUB[0] = RGB15(0, 0, 0);
BG_PALETTE_SUB[1] = RGB15(0, 31, 0);
```

7 Bibliografía

[1] Ejemplos de desarrollo para NDS. Repositorio en GitHub. Disponible en <https://github.com/magusti/NDS-hombrew-development/tree/master/hw_progGuideToTheNDS_usingTiles >.

[2] Phoenix Rising. (2005). Homebrew Programmers Guide to the Nintendo DS . Disponible en <http://dspassme.jzdocs.com/programmers_guide/tutorial/index.html >.

[3] DeSmuME. Página web del proyecto. Disponible en <http://desmume.org/>.

[4] J. Amero (Patater). (2008). Introduction to Nintendo DS Programming. Disponible en <https://patater.com/files/projects/manual/manual.html >.

[5] Boudeville, O. (2008). A guide to homebrew development for the Nintendo DS. Disponible en <http://osdl.sourceforge.net/main/documentation/misc/nintendo-DS/homebrew-guide/HomebrewForDS.html>.

[6] F. Moya y M. J. Santofimia. (2011). Laboratorio de Estructura de Computadores empleando videoconsolas Nintendo DS. Ed. Bubok Publishing. ISBN. 978-84-9981-039-3.

[7] Vijn,J. (2007). grit - GBA Raster Image Transmogrifier. Disponible en <https://www.coranac.com/projects/grit/>.