UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Gestión de las capas en la implementación del tutorial "The Bouncing Ball" para 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 recorrer parte de la creación de Mukunda Johnson [1], uno de los grandes desarrolladores para Nintendo DS (NDS). Entre otras cosas, realizó un tutorial[1] sobre el desarrollo de aplicaciones para NDS que permite iniciarse en este complejo mundo. Sobre el trabajo de Johnson, Jaén [2] realizaría un trabajo de actualización del tutorial original, corrigiendo algunas erratas del texto inicial, introduciendo cambios en el código debidos a modificaciones de las librerías en que se basa; al tiempo que se amplían algunos apartados con pequeñas "demos" de conceptos.

Ya no es posible consultar el tutorial original en la red, así que **el presente trabajo ofrece una versión del apartado de uso de las capas, actualizada** a las versiones existentes de las librerías sobre las que se basa. Quiero rendir un homenaje a estos desarrolladores que han contribuido con su código y con sus explicaciones a que otros puedan adentrarse en este difícil campo de desarrollo de aplicaciones para videoconsolas, que son plataformas muy populares y diferentes del ordenador de sobremesa.

Para facilitar el seguimiento de este trabajo se han dispuesto todos los elementos del proyecto que aquí se comenta en *GitHub* [3]. Nos ocuparemos en este artículo de comentar las acciones necesarias para crear la escena de un juego que, utilizando la botonera de la NDS, mueva una pelota a la que el efecto de la "gravedad" y el "rozamiento con el aire" le hacen avanzar en una trayectoria casi parabólica y que se achata al caer y "botar" en el suelo.

2 Objetivos

Una vez que el lector haya leído con detenimiento este documento:

- Será capaz de entender los conceptos básicos de desarrollo de aplicaciones de videojuego en la consola NDS, centrándose en el uso de capas (también llamadas *backgrounds* o niveles).
- Será capaz de entender el proceso de transformación de los recursos de naturaleza gráfica que se incorporan a la aplicación como elementos de visualización estáticos.
- Será capaz de seguir un ejemplo de código que permite generar los elementos de fondo de una escena con dos capas o fondos.

No es un objetivo descrbir el uso de emuladores para ejecutar las aplicaciones para la NDS, como *DeSmuMe* [4]. 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 [5] y [6]. Tampoco es un objetivo de este trabajo entrar a ver las

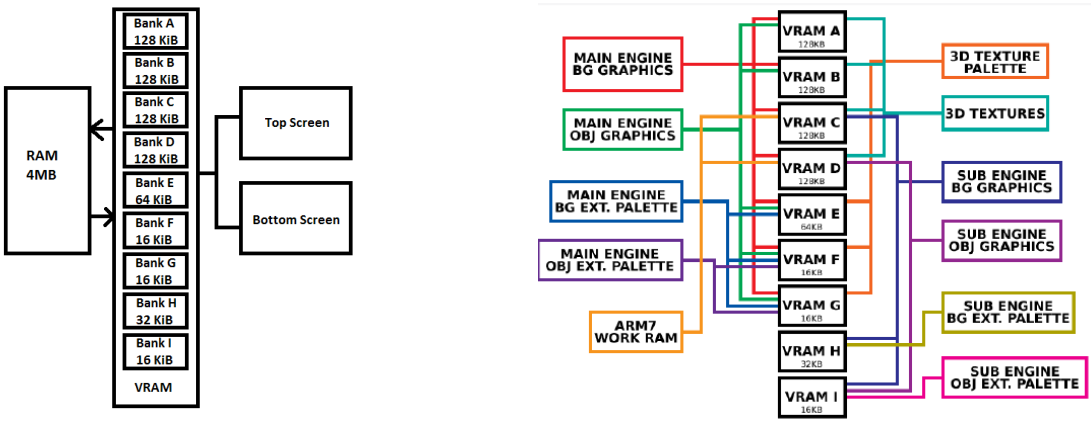
[1] Publicado originalmente en "How to Make a Bouncing Ball Game for Nintendo DS" <http://ekid.nintendev.com/bouncy/>, este trabajo ya no está disponible en la red; pero existe una copia de fecha de 2015 en [3] que es al que nos referiremos al hablar del "tutorial original".

características de la NDS. Si el lector tiene curiosidad o necesidad de profundizar en estas cuestiones es recomendable consultar [7].

Antes de ver un ejemplo, hemos de hablar un poco de teoría, en concreto el uso de la memoria que hace la NDS. Veamos qué consideraciones hay que tener en cuenta para dibujar el fondo de la escena en 2D, veamos qué son las capas, las teselas y cómo se distribuyen en la memoria de la videoconsola.

3 Introducción

La NDS es un computador que comparte su memoria central (RAM, de 4MB) entre los dos procesadores (ARM9 y ARM7) que alberga en su interior. *También* comparte, véase la Figura 1a, su **memoria de vídeo** (VRAM, de 656 MiB) entre dos subsistemas o motores gráficos (denominados *Main* y *Sub*) y otros componentes que proporcionan aceleración gráfica para el uso de *sprites* (OAM) y 3D. Al estilo de los computadores actuales con dos tarjetas gráficas, esto le permite generar dos imágenes diferentes para cada una de sus dos pantallas a 60 cuadros por segundo.



a) *b)*

Figura 1: Memoria en la NDS: (a) relación entre la RAM, la VRAM y las dos pantallas (fuente [8]) y (b) bancos o particiones de la VRAM (fuente [1]).

Para dibujar con rapidez en las dos pantallas, la NDS ofrece una aceleración por *hardware* que se basa en la existencia de dos motores gráficos (que se asignan uno a cada pantalla), la división de la VRAM en bancos de memoria (cuyo propósito está preestablecido) y cuya asignación a uno de los motores es configurable.

Los nueve bancos de la NDS pueden ser usados con diferentes propósitos y cada motor solo puede acceder a parte de ellos: el *Main* a 608 KiBytes y el *Sub* a 400 KiBytes, de manera que parte de la VRAM es compartida entre ellos. Se puede consultar [10] para ver más detalles de bajo nivel. Nos vamos a limitar en este artículo a los bancos que están asignados con la etiqueta “Main Engine BG Graphics”, véase la Figura 1b, que corresponde a los bancos A al G.

En cuanto a la estrategia para **dibujar** en la NDS hay dos posibles maneras. Una es acceder a la VRAM directamente y pintar cada píxel de cada una de las dos pantallas, es lo que se conoce como “por *software*”

o modo *Frame Buffer; l*o que implica actualizar los 256x192 píxeles de cada pantalla, Figura 2a.

a) *b)*

Figura 2: Resolución de las pantallas de la NDS: en píxeles (a) y teselas (b). Imagen elaborada sobre la plantilla de [10].

Y también es posible dibujar en modo mosaico o **teselado** (*tiled*); lo que supone modificar los 32x24 teselas, de cada pantalla, véase la Figura 2b. En este modo los contenidos de cada pantalla se forman por la superposición de cuatro **capas o fondos** (*text background* o BG) y una capa superior donde se gestionan los objetos (*sprites*). Esta es la opción más habitual en la implementación de muchos videojuegos, todavía hoy, en los que el dibujo de la escena es muy repetitivo. Esto es, está compuesto por pequeñas imágenes que se observan en diferentes lugares de la escena, a veces rotadas o con algun color cambiado, que se denominan **telesas** (*tiles* o *character blocks*).

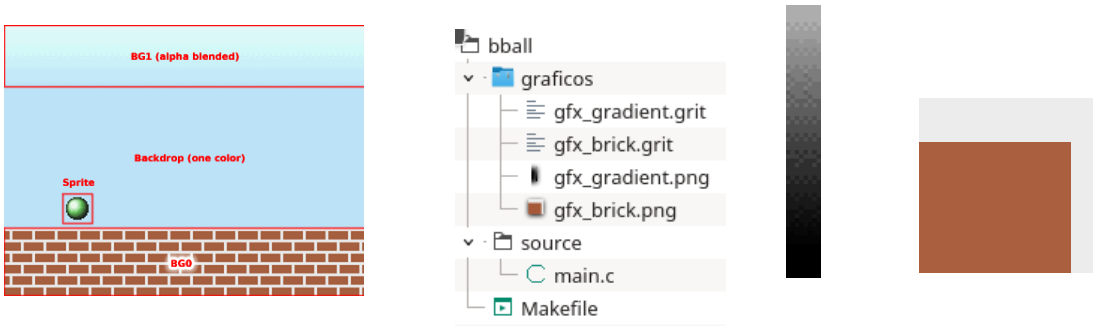
En el caso de la NDS estas teselas son imágenes cuadradas, de tamaño 8x8 píxeles (también se podría trabajar en 16x16, 32x32 y 64x64) y con una paleta de colores asociada. La pantalla se puede ver entonces como formada por una rejilla (*grid*) o tablero, cuyas casillas (*screen blocks*) son rellenadas por las teselas. Estas teselas se referencian por un número y se disponen en lo que se denomina un **mapa** que indica qué tesela se dibuja en cada coordenada del tablero. Y lo mejor, que con solo cambiar el índice de la tesela a dibujar en cada casilla, será la máquina la que actualizará el contenido de la pantalla. En este modo, se puede decir que la VRAM se comporta como un tablero de 32x24 casillas.

4 Estructura y contenido del proyecto

El tutorial en el que se basa este artículo está formado por una escena construida con dos capas estáticas y un objeto que controlará el usuario (*sprite*) y que es el único cuya posición puede variar. Vamos a centrarnos en este artículo solo en la construcción de la escena mediante capas, niveles o *backgrounds.*

Así que vamos a ver cómo se disponen las capas que forman la escena, esto es, los elementos estáticos que aparecen en la Figura 3a y que son el BG1 (que incluye el *Backdrop*) y el BG0. El BG0 contendrá la pared de ladrillos que está, en primer plano, en la parte inferior de la escena y que constituye el suelo (o plataforma) que limita por abajo el movimiento de los objetos. El BG1 es la parte de “cielo azul” que ocupa

el resto de la escena y que está en segundo plano; en su parte superior, junto a un degradado que da una aproximación a una capa de nubes.



a) *b)* *c)*

Figura 3: El proyecto: (a) imagen final del tutorial extraída de [2], (b) archivos que lo componen y (c) texturas necesarias para el proyecto (gradiente para el cielo y ladrillo para la pared).

El proyecto se estructura en la forma en que se muestra en la Figura 3b, con un subdirectorio para el código fuente y otro para los recursos gráficos en formato de mapas de bits. La Figura 3c muestra estos últimos, ampliados para que se vean los detalles y que son los únicos que utilizaremos en el desarrollo de la escena. Además, en el directorio del proyecto, hay un fichero *Makefile* que es el encargado de organizar el trabajo a realizar con los ficheros para obtener el ejecutable final. Para ello, en él se configuran los subdirectorios que contiene cada recurso (código fuente e imágenes en este caso) y las operaciones a realizar con cada herramienta para convertir y combinar todos los recursos en el fichero resultante final.

Con el uso del *Makefile*, se automatiza el proceso de conversión del formato de los ficheros gráficos a un formato basado en teselas soportado por la NDS. Veamos ahora esta conversión.

4.1 De mapas de bits a teselas

El **GRIT** [9] es una aplicación que permite convertir los ficheros gráficos que vamos a utilizar en un formato que la NDS puede leer. Para ello se le indican los parámetros que corresponden a cada imagen y que están guardados en otros tantos ficheros del mismo nombre que los gráficos (p. ej. un PNG) y con extensión "*.grit*". Cada vez que se modifiquen los ficheros gráficos, el GRIT los convertirá (como muestra la Figura 4) a código ensamblador y generará también las cabeceras en formato de C/C++ para declarar las variables que contienen los elementos gráficos. Así se les podrá referenciar en el código Todo ello será enlazado junto con el resto del código, que es una manera de importar los recursos gráficos en un ejecutable final.

Veámoslo con un poco de detalle, la Figura 4 muestra cómo se convierte el *gfx_brick.png* (Figura 4a) en una tesela (Figura 4c) más una paleta de colores para este elemento. La imagen original es un PNG indexado de 8 bits, esto es, tiene una paleta asociada de 256 colores. Pero la imagen solo utiliza dos colores, los dos últimos que se ven en la Figura 4c. El primero siempre sirve para marcar los puntos transparentes; aunque, como en este caso, no hay ninguno. La NDS trabaja con las teselas

(como hemos mencionado), como la que se ve en la Figura 4c: en la que cada punto de la mismas tiene como información asociada un número que corresponde al índice del color asociado en la paleta; y, en este caso, no hay ninguno a cero (0), puesto que no hay ningún punto transparente como acabamos de decir.

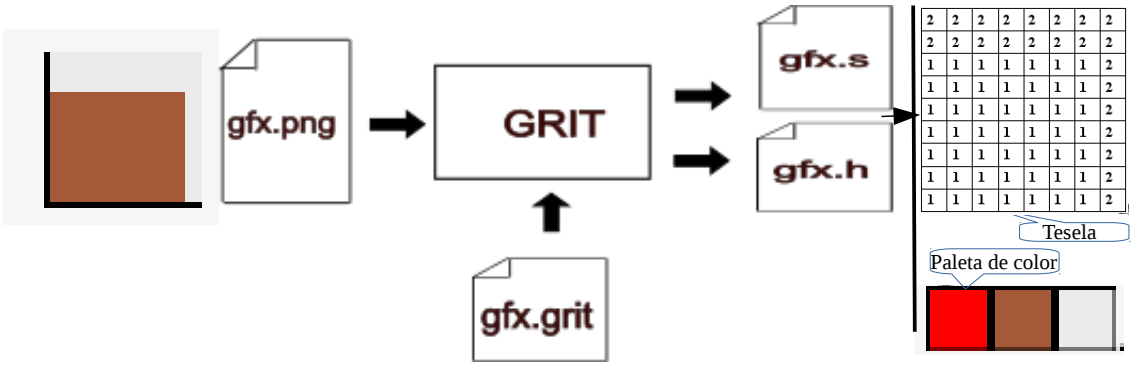


Figura 4: Conversión de ficheros gráficos con GRIT: (a) imagen en formato PNG (extraída de [1]), (b) esquema del proceso y (c) resultados obtenidos tesela y paleta.

Al utilizar el GRIT, le indicaremos que solo queremos una paleta con tres colores y una tesela de profundidad 4 bits (la mínima que admite la NDS). Estos parámetros están en el fichero *gfx_brick.grit*, como muestra la orden *cat* en la parte superior de la Figura 5. ¿Qué vemos en el resto de la Figura 5? ¡El resultado de la conversión del fichero PNG! Lo vemos como código C que es más sencillo de leer que en ensamblador y dejamos, para el lector curioso, ejecutar la orden como se hace en el proyecto, para ver la salida en formato ensamblador[2].

La Figura 5 muestra el resultado de la ejecución de la conversión y los ficheros *gfx_brick.c* y *gfx_brick.h*. El primero muestra dos variables, la primera con el nombre del fichero PNG y la terminación *Tiles* para las teselas: en este caso, una única de 8x8 píxeles de 4 bits para cada píxel, por lo que ocupa 8*8*4 bits = 2^3*2^3*2^2 bits = 2^5*2^3 bits =2^5 bytes = **32 bytes**. Si le extraña el valor es porque está en formato *Little Endian*[3], ahora seguro que lo ve ;-). Y, la segunda, con la terminación *Pal* (para la paleta), que tiene tres valores de 2 bytes cada uno, por lo que su tamaño será de seis bytes. Como estos tamaños dependen de cada imagen y de los parámetros de la conversión, se genera también un fichero .H (véase el fichero *gfx_brick.h* en la Figura 5) que contiene la declaración de las variables junto a su tamaño. Para los tamaños se declaran unas constantes con el nombre del vector cuyo tamaño contienen y la terminación *Len*: por lo que verá *gfx_brickTilesLen* y *gfx_brickPalLen*, con los tamaños mencionados.

Hay que insistir en que estas transformaciones se hacen de manera programada durante la creación del ejecutable, siguiendo las instrucciones del fichero *Makefile*. Aquí se han desglosado para ilustrar el proceso.

---

[2] Para hacerlo ejecute la orden: `$ grit gfx_brick.png -fts; cat gfx_brick.s`.

[3] Una explicación sobre *Little Endian* y *Big Endian* se puede consultar en <http://www.coranac.com/tonc/text/bitmaps.htm#sec-data>.

```
$ cat gfx_brick.grit
# the -pn{n} rule specifies the number of entries in the palette: 3 colors
-pn3
# the -gB{n} rule specifies the output graphic format: 4-bit (16 color palette)
-gB4
$
$ grit gfx_brick.png -ftc ; cat gfx_brick.c
//{{BLOCK(gfx_brick)
//======================================================
//      gfx_brick, 8x8@4,
//      + palette 3 entries, not compressed
//      + 1 tiles not compressed
//      Total size: 6 + 32 = 38
//      Time-stamp: 2019-12-12, 15:17:48
//      Exported by Cearn's GBA Image Transmogrifier, v0.8.15
//      ( http://www.coranac.com/projects/#grit )
//======================================================
const        unsigned        int        gfx_brickTiles[8]        __attribute__((aligned(4)))
__attribute__((visibility("hidden")))= {
      0x22222222,0x22222222,0x21111111,0x21111111,0x21111111,0x21111111
,0x21111111,0x21111111, };
const        unsigned        short        gfx_brickPal[4]        __attribute__((aligned(4)))
__attribute__((visibility("hidden")))= {
      0x001F,0x1D74,0x77BD, };
//}}BLOCK(gfx_brick)
$
$ cat gfx_brick.h
//{{BLOCK(gfx_brick)
//======================================================
//        gfx_brick, 8x8@4,
//        + palette 3 entries, not compressed
//        + 1 tiles not compressed
//        Total size: 6 + 32 = 38
//        Time-stamp: 2019-12-12, 15:28:18
//        Exported by Cearn's GBA Image Transmogrifier, v0.8.15
//        ( http://www.coranac.com/projects/#grit )
//======================================================
#ifndef GRIT_GFX_BRICK_H
#define GRIT_GFX_BRICK_H
#define gfx_brickTilesLen 32
extern const unsigned int gfx_brickTiles[8];
#define gfx_brickPalLen 6
extern const unsigned short gfx_brickPal[4];
#endif // GRIT_GFX_BRICK_H
//}}BLOCK(gfx_brick)
$
$ cat gfx_gradient.grit
# the gradient has 16 colors
-pn16
# the gradient is a 4-bit image
-gB4
```

Figura 5: Ejemplos de conversión de gráficos con GRIT.

De manera similar, para la imagen del gradiente de las nubes (*gfx_gradient.png, en la Figura 3c)* de 8x64 píxeles y con una paleta de 16 colores, se convertirá a un vector de teselas (ocho en ese caso), utilizando los parámetros que figuran en el fichero *gfx_gradient.grit* y que se puede ver al final de la Figura 5. Al cargarla en la VRAM, a diferencia del gfx_brick que ocupaba 1 tesela, esta imagen ocupará 8 teselas, por lo que se puede imaginar como en la *Figura 6a*: cada una de

las “celdas” de esta memoria (es la unidad en la que accede a su contenido) está ocupada por una tesela (pintadas en la *Figura 6a* como la textura de imagen que las ha producido para reconocerlas, en lugar de los índices a la paleta de color que realmente es su contenido), podemos ver la primera a ceros, después la del ladrillo y las ocho del degradado para el cielo de nubes. El resto están marcadas con interrogantes para expresar que no han recibido contenido.

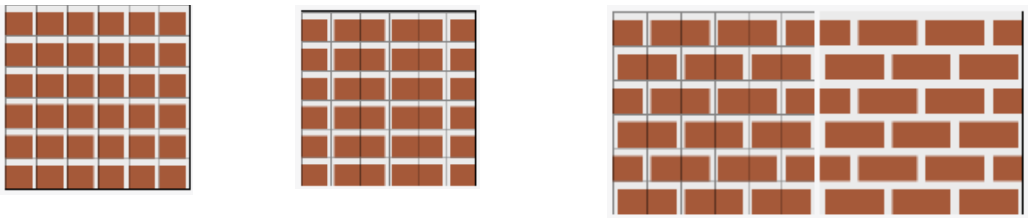
Además, recordemos, hemos generado unas paletas que podemos verlas representadas en la Figura 6b, una por fila, que corresponde a cada elemento cargado; así vemos en la primera fila la asociada al *gfx_brick* y en la segunda la asociada a *gfx_gradient*.

a) *b)*

Figura 6: Contenido de parte de la memoria de vídeo: a) banco con las teselas de la pared y el gradiente cargados [1] y b) banco con las paletas.

4.2 Etapas del proyecto

Ya hemos dicho que los contenidos del tutorial original y las versiones del código actualizadas se han ido disponiendo en *GitHub* [3]. El resultado final es el que se puede ver en este artículo entre la Figura 8 y la Figura 9, acompañado de las explicaciones o comentarios junto a las instrucciones.



a) b) c)

Figura 7: Creando el suelo en la capa 0. a) teselas aplicadas sin modificar, b) teselas alternadas con reflexión vertical y c) alternando la reflexión de las columnas impares. Imágenes extraídas de [1].

Para ver cómo el proyecto va creciendo se han guardado las etapas intermedias que corresponden a este artículo en el directorio *tutorial_BouncingBal/bball__versions/bball__subVersionsBackgrounds/* y corresponden a:

- bball__v01. Es la inicial, muestra la carga de recursos y la generación del fondo de cielo liso.
- bball__v02. Muestra cómo se genera el cielo con su banda de “nubes“ y el suelo (o la plataforma) sobre el que se limitará el movimiento de los objetos. La Figura 7 muestra su generación algorítmica, que corresponde a las líneas 54 a la 57 de la Figura 9.

El “suelo” se genera a partir de la imagen de un ladrillo alternándola con la versión del mismo reflejada en vertical y, también, en función de la paridad de cada fila.

- bball__v03. Es la versión con los elementos estáticos de la escena terminados. Es la que se muestra aquí el código y corresponde al planteamiento completo del tutorial original con las simplificaciones y normalizaciones de código que hemos introducido. Se muestra en la Error: no se encontró el origen de la referencia y Error: no se encontró el origen de la referencia, donde lo comentaremos brevemente junto a las líneas de código.

```
1       #include <nds.h>
2       #include <stdio.h>
3       // Referencias a los gráficos, se generan "automágicamente" por GRIT
4       #include "gfx_brick.h"
5       #include "gfx_gradient.h"
6
7       void setupGraphics( int *bg0, int *bg1 );
8       void update_logic() {};
9       void update_graphics( int bg0, int bg1 ) {};
10
11      int main( void ) {
12              int bg0, bg1;
13              setupGraphics( &bg0, &bg1 );
14              while(1)        {
15                      processLogic();         // Interacción y cálculo de animaciones
16                      swiWaitForVBlank();             // Espera al “vblank”
17                      updateGraphics( bg0, bg1 );// Actualizar graficos
18              }
19      }
20      //Teselas (tiles)
21      #define tile_empty              0 //tile 0 = empty
22      #define tile_brick              1 //tile 1 = brick
23      #define tile_gradient           2 //tile 2 = gradient
24      //Paletas
25      #define pal_bricks              0 //paleta brick (entrada 0->15)
26      #define pal_gradient            1 //paleta gradient (entrada 16->31)
27      #define backdrop_colour     RGB8( 190, 255, 255 )
28
29      u16 *bg0map, *bg1map;
...
```

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

5 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.

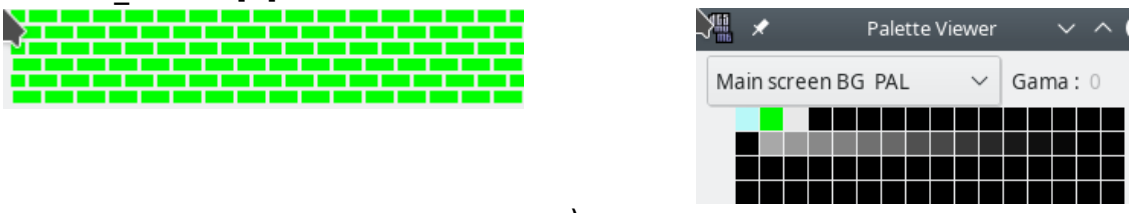
```
...
31      void setupGraphics( int bg0, int *bg1   ) {
32        int n, x, y, bg0, bg1;
33        vramSetBankE( VRAM_E_MAIN_BG );
34        vramSetBankF( VRAM_F_MAIN_SPRITE );
35        // Inicializar bgs (fondos, capas oniveles)
36        *bg0 = bgInit(0, BgType_Text4bpp, BgSize_T_256x256, 1, 0);
37        *bg1 = bgInit(1, BgType_Text4bpp, BgSize_T_256x256, 2, 1);
38        //Generar el primer banco de tile por borrado a cero
39        for( n = 0; n < 16; n++ )     {       BG_GFX[n] = 0;    }
40        //Cargando los graficos para las teselas de las capas
41        dmaCopy( gfx_brickTiles, (BG_GFX + (tile_brick * 16)), gfx_brickTilesLen );
42        dmaCopy( gfx_gradientTiles, (BG_GFX + (tile_gradient * 16)),
                    gfx_gradientTilesLen );
43        //Cargando las paletas para las teselas de las capas
44        dmaCopy( gfx_brickPal, BG_PALETTE, gfx_brickPalLen );
45        dmaCopy( gfx_gradientPal, BG_PALETTE+(pal_gradient*16),
                                  gfx_gradientPalLen );
46         //Asignar color de fondo
47         BG_PALETTE[0] = backdrop_colour;
48        //Construir los mapas
49        bg0map = bgGetMapPtr( *bg0 );
50        bg1map = bgGetMapPtr( *bg1 );
51        for( n = 0; n < 1024; n++ ) bg0map[n] = 0;
52        for( x = 0; x < 32; x++ )    {
53          for( y = 18; y < 24; y++ )      {
54            // magical formula to calculate if the tile needs to be flipped.
55            int hflip = (x & 1) ^ (y & 1); // ^ es la XOR → 1^0 y 0^1 → 1
56            // set the tilemap entry
57            bg0map[x + y * 32] = tile_brick | (hflip << 10) | (pal_bricks << 12);
58          }
59        }
60        for( n = 0; n < 1024; n++ ) bg1map[n] = 0;
61        for( x = 0; x < 32; x++ ) {
62          for( y = 0; y < 8; y++ )   {
63              int tile = tile_gradient + y;
64              bg1map[ x + y * 32 ] = tile | (pal_gradient << 12);
65          }
66        }
67      // Combinar el gradiente (BG1) con el "backdrop".
68        REG_BLDALPHA = (16<<8) + (4); //SRC_BG1*16 + BACKDROP*4
69        // Inicializar el modo de vídeo
70        videoSetMode( MODE_0_2D|DISPLAY_BG0_ACTIVE|DISPLAY_BG1_ACTIVE );
71      } // Fi de setupGraphics
```

Figura 9: Código del ejemplo, parte 2. Partes del código de [1] y [2].

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* [3], leer el original y probar las ampliaciones propuestas en el

trabajo de Jaén [2]. Con el emulador DeSmuME [4] es posible ejecutar los desarrollos expuestos. Y, por supuesto, experimentar y modificar el código que se publica en el repositorio del proyecto. La Figura 10 muestra cómo se puede cambiar el color de la plataforma, cambiando la segunda posición de su paleta. Pruebe cambiarlo accediendo a "BG_PALETE[1]".



a)

b)

Figura 10: Puede cambiar el color de los ladrillos, cambiando el color en la posición "pal_bricks" (Figura 8) de la paleta.

6 Bibliografía

[1] M. Johnson. Sitio web. Disponible en <http://mukunda.com/projects.htm >.

[2] J. D. Jaén Gomariz. (2015). Tutorial práctico para desarrollo de videojuegos sobre plataforma Nintendo NDS. Disponible en <http://hdl.handle.net/10251/56433 >.

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

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

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

[6] O. Boudeville. (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>.

[7] 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.

[8] Foxi4. (2012). *DS Programming for Newbies!* Disponible en <https://gbatemp.net/threads/ds-programming-for-newbies.322106/page-6 >.

[9] *Grit. GBA Raster Image Transmogrifier.* Disponible en <http://www.coranac.com/projects/grit >.

[10] S. Stair. (2006). *GBATEK. Gameboy Advance / Nintendo DS - Technical Info*. Disponible en <https://www.akkit.org/info/gbatek.htm>.