Arquitectura y Entornos de desarrollo para videoconsolas

Práctica 3
Uso de la imagen en la videoconsola NDS

M. Agustí
Marzo 2016

La práctica se podrá realizar en entorno GNU/Linux en el laboratorio. Vamos a abordar el uso de la imagen en la videoconsola de estudio, tanto en lo que se refiere a las posibilidades de mostrar una imagen estática en 2D o 3D, así como del soporte a la animación y la interacción.

Recuerde que ha de guardar los resultados de sus acciones a lo largo de las prácticas para confeccionar el porfolio de prácticas, que es la forma en que se evaluará las prácticas de la asignatura. No descuide pues, hacer copias de lo que necesite del laboratorio en su espacio del servidor de la asignatura.

1 Introducción

Trabajaremos a partir de unos ejemplos de código disponibles en la documentación del SDK; *Libnds Documentation* [1]. La videoconsola hace uso de la imagen mediante la configuración del modo de trabajo de un subsistema gráfico diferente para cada pantalla, denominados MAIN y SUB. La fig. 1, resume los posibles modos de funcionamiento.

Main 2D Engine				
Mode	**BG0**	**BG1**	**BG2**	**BG3**
Mode 0	Text/3D	Text	Text	Text
Mode 1	Text/3D	Text	Text	Rotation
Mode 2	Text/3D	Text	Rotation	Rotation
Mode 3	Text/3D	Text	Text	Extended
Mode 4	Text/3D	Text	Rotation	Extended
Mode 5	Text/3D	Text	Extended	Extended
Mode 6	3D	-	Large Bitmap	-
Frame Buffer	Direct VRAM display as a bitmap			
Sub 2D Engine				
Mode	**BG0**	**BG1**	**BG2**	**BG3**
Mode 0	Text	Text	Text	Text
Mode 1	Text	Text	Text	Rotation
Mode 2	Text	Text	Rotation	Rotation
Mode 3	Text	Text	Text	Extended
Mode 4	Text	Text	Rotation	Extended
Mode 5	Text	Text	Extended	Extended

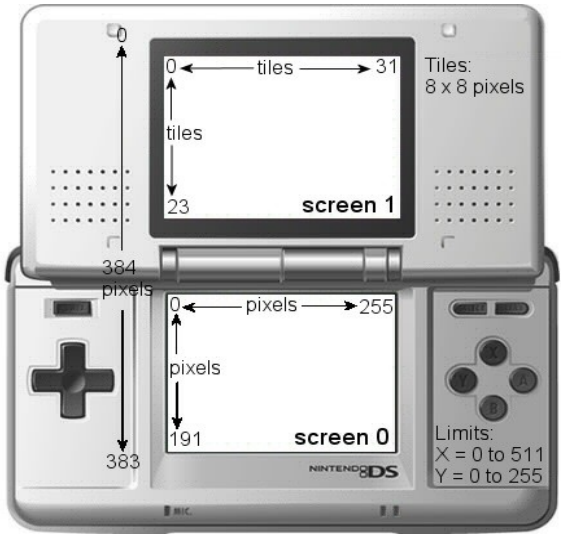


Figura 1: Modos de video y resolución del dispositivo

De manera implícita ya hemos trabajado con los dos extremos: el modo *MODE_0_2D* a través de la función *consoleDemoInit,* desde el primer ejemplo de ¡Hola, Mundo! y todos los demás que venían utilizando una interfaz de texto.

Vamos a ver los conceptos de fondo (background), el acceso directo al contenido (píxeles) en pantalla, el uso de la paleta de color, los *sprites* y las posibilidades de uso de operaciones de dibujado en 3D.

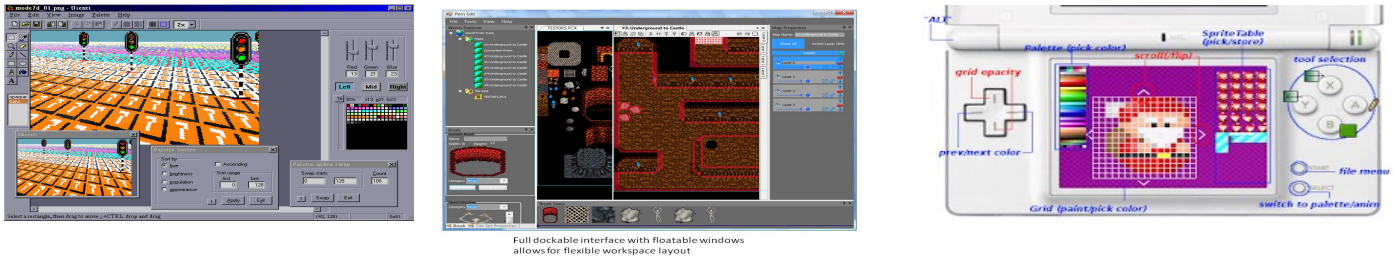
En el apartado *Examples* de la documentación se referencia a los mismos ejemplos que ya tendrá instalados en su instalación local en `${DEVKITPRO}/examples`. Vamos a ver de estos los relativos a las formas de gestionar la imagen y sus propiedades.

Asumiremos que se dispone de las herramientas necesarias: *devkitPro* para NDS, *libnds* y emuladores (*desmume* y *no$gba*). Para nuestra práctica, además, haremos uso de una aplicación de tratamiento de imágenes: Gimp [2].

Es importante hacer notar que las imágenes (los gráficos) se pueden generar en tiempo de ejecución o se pueden cargar/importar a partir de mapas de bits realizados con aplicaciones externas. Las

imágenes en formato de mapa de bits se pueden añadir (importar) en el ejecutable final. Para la conversión al formato nativo de la videoconsola de estudio y, como sucedió en el caso del audio con MMUTIL, el *Makefile* de los ejemplos que las utilizan llamará al GRIT (véase "Anexo A: GRIT" para comprobar el tipo de operaciones que puede aplicar) para que haga las conversiones que se le instruyan para los ficheros que se encuentre en un determinado subdirectorio del proyecto.

Como la producción de componentes de tipo gráfico queda fuera de los objetivos de esta práctica no lo abordaremos directamente, la fig. 2, muestra una pequeña recopilación de editores gráficos, paletas, mapas y *sprites* para la videoconsola de estudio. Pueden utilizarse herramientas genéricas como *Inkscape* o *Gimp*, así como otras más específicas como *Usenti*[1]., *Pern Editor*[2] o SEDS[3], Estas aplicaciones, externas al entorno de desarrollo, permiten la creación y postproducción de imágenes para esta plataforma. Complementando así al entorno de desarrollo en el que incidimos.



a) *b)* *c)*

Figura 2: Editores específicos para GBA/NDS: a)Usenti, b) Pern Editor y c) SEDS.

Otras aplicaciones externas interesantes son las de ayuda en la gestión de los componentes hardware para gráficos (motores 2D y 3D), así como las particularidades de estos: los modos gráficos, la configuración y la detección de problemas.

Cuando el número y de complejidad del uso de recursos gráficos crecen, la asignación de bancos de memoria puede resultar en una fuente de conflictos que se traduce en la aparición de imágenes con artefactos en la videoconsola. Aunque escapa a los objetivos de la práctica, para ayudar a la toma de decisiones en estos casos, dos herramientas externas al entorno de desarrollo utilizado hasta ahora pueden resultar de utilidad:

- El "NDS Homebrew VRAM BG Allocation Conflict Viewer", <http://mtheall.com/vram.html>.
  Permite, de manera gráfica, elegir el modo gráfico de cada uno de los dos motores 2D y, con la ayuda de los colores, se puede visualizar si hay algún conflicto. Esto es, si algún banco de memoria se ha utilizado con más de un propósito diferente.
- La "NDS Homebrew VRAM Banks Selector" <http://mtheall.com/banks.html#>. Permite la elección y asignación de los bancos de memoria al modo gráfico escogido para los motores 2D y los *slots* del motor gráfico 3D, generando las instrucciones necesarias para ello.

2 Imprimiendo o dibujando texto

El *MODE_0_2D*, con 32 apariciones y el *MODE_0_3D*, con 28, son los dos más usados de los

1 Véase la página web <http://www.coranac.com/projects/usenti/>.
2 Disponible en la dirección web de <http://pern.drunkencoders.com/>.
3 Disponible en la dirección web de <http://sylvainhb.blogspot.com.es/p/sprite-editor.html>.

ejemplos que acompañan al SDK de *libnds* para, respectivamente, poner información en "modo texto" en alguna de las pantallas de la NDS y mostrar ejemplos de código portado a la plataforma. Aunque lo llamemos "modo de texto" es un modo de vídeo especializado en dibujar los caracteres; esto es, mostrar símbolos gráficos pertenecientes a un alfabeto que se ha de cargar en memoria y que se muestra en pantalla en un modo de vídeo básico.

Revisemos la operativa del funcionamiento del modo de texto, que permite el funcionamiento de las pantallas de la plataforma de estudio al estilo consola de texto o terminal. Ya vimos las bases, así que ahora nos adentraremos en los conceptos avanzados de:

- Qué hay bajo la función *consoleDemoInit* que tanto hemos venido utilizando.
- Cómo configurar las dos pantallas en "modo de texto".
- Cómo se define un tipo de letra.
- Se pueden aplicar efectos geométricos al texto en pantalla.

Ejercicio: Anote la descripción del tipo de datos PrintConsole, así como las funciones y los parámetros que implementan la función *consoleDemoInit* dada por el código que se muestra y sacado del fichero fuente de *libnds source/arm9/console.c*:

```
PrintConsole* consoleDemoInit(void) {
        videoSetModeSub(MODE_0_2D);
        vramSetBankC(VRAM_C_SUB_BG);

        return consoleInit(NULL, defaultConsole.bgLayer, BgType_Text4bpp,
                           BgSize_T_256x256, defaultConsole.mapBase,
                           defaultConsole.gfxBase, false, true);
}
```

Ejercicio: Estudie el ejemplo de código *print_both_screens*, perteneciente a la sección de ejemplos ${DEVKITPRO}/examples/Graphics/Printing/. Anote, para el ejemplo señalado, la secuencia de instrucciones que definen el modo de vídeo, asignan espacio de memoria y configuran el tipo de fuente de letra a utilizar para cada pantalla.

Ejercicio: Estudie el ejemplo de código *custom_font*, perteneciente a la sección de ejemplos ${DEVKITPRO}/examples/Graphics/Printing/. Anote, para el ejemplo señalado, de dónde se obtiene la información gráfica del tipo de letra que asigna a la definición de la consola, mostrando una parte del fichero, la que muestra los caracteres ABC y abc, p. eje. ¿Qué formato tiene la imagen, su resolución espacial y la profundidad de color?

Ejercicio: Estudie el ejemplo de código *rotscale_text*, perteneciente a la sección de ejemplos ${DEVKITPRO}/examples/Graphics/Printing/. Anote, para el ejemplo señalado, cuál es el modo de vídeo escogido y la diferencia en el modo de inicializar la consola, para permitir que al texto en pantalla se le puedan aplicar efectos de traslación, rotación y escalado que se muestran en él.

3 Gráficos en 2D

A guide to homebrew development for the Nintendo DS. 2D Rendering. Overview. <http://osdl.sourceforge.net/main/documentation/misc/nintendo-DS/homebrew-guide/HomebrewForDS.html>

"... 2D memory, which is mostly made of:

- ***background** memory, which contains either tilesets (32 blocks, 16 kilobytes each) and tile maps (32 blocks), or bitmaps*
- ***sprite** memory, which contains tiles for each sprite*

As for sprite attributes, as said previously each 2D core has its own built-in dedicated memory, which contains 128 entries (one entry per possible sprite). Each entry is made of four 16-bit attributes, storing the size, shape and location of the associated sprite. Up to 32 out of the 128 entries can correspond to affine transformations (named rotsets), whose additional attributes specify rotation and scale. Hence up to 32 rotsets can be defined, but more than one sprite can be associated to a given rotset.

Base palettes have their own per-engine memory too. Each 2D engine has two base palettes, one for the background, one for the sprites. Each base palette contains 256 16-bit color entries, in x555 BGR format.

3.1 Fondos (*backgrounds*)

En este apartado exploramos con detalle el uso de los fondos, basándose en este caso en su definiciones en base a una imagen contenida en un fichero que se fija en el proyecto.

El ejemplo de código *${DEVKITPRO}/examples/Graphics/Backgrounds/*rotation permite explorar la forma en que se carga una imagen y las operaciones que se pueden realizar sobre ella con el apoyo del hardware.

Ejercicio: Anote las acciones asociadas a cada botón que utiliza esta aplicación, así como la definición de las funciones que se utilizan para las transformaciones geométricas que se aplica a la imagen cargada.

El ejemplo de código *${DEVKITPRO}/examples/Graphics/Backgrounds/16bit_color_bmp* muestra cómo es posible incluir estáticamente una imagen en formato mapa de bits (PNG en este caso) incrustada dentro del ejecutable que se obtiene.

Ejercicio: Cambie la imagen que se muestra en este ejemplo y obtenga una captura de pantalla del resultado. Para ello genere, con la ayuda del *Gimp,* una versión de una foto suya con las mismas características que el fichero PNG que se utiliza en este ejemplo. Sobreescriba el fichero *data/drunkenlogo.png* y reconstruya el ejecutable.

Ejercicio: Compruebe si puede cambiar también la imagen y señale en qué se diferencia este ejemplo del que puede ver en *${DEVKITPRO}/examples/Graphics/Backgrounds/256_color_bmp.*

El ejemplo de código *${DEVKITPRO}/examples/Graphics/Backgrounds/*all_in_one ya fue sugerido en otra práctica para utilizar, pero en aquel caso como manera de aglutinar una serie de ejemplos en una sola aplicación. Estudiemos ahora lo que nos aporta desde el punto de vista de los fondos.

Ejercicio: Describa la funcionalidad de los ejemplos que se muestran en este compendio y obtenga

una captura de pantalla representativa de cada categoría en el caso de las categorías *Basic*, *Bitmap* y *Scrolling*; así como una para cada ejemplo de la categoría *Advanced*. Para ello, construya y ejecute el ejemplo anotando las opciones y los resultados que se obtienen.

3.2 Sprites ("duendes")

Este término es de uso habitual en gráficos por computador,y designa una técnica de animación por computador, dando nombre también a los objetos (pequeñas imágenes, en relación con el tamaño de la escena donde se sitúan) que se superponen al fondo y, posiblemente, entre ellos. El SDK que utilizamos ofrece apoyo en forma de operaciones y estructuras de datos (de muy bajo nivel) para modificar algunas de sus propiedades (posición, geometría, escalado, rotación, paleta, visibilidad, ….). No existe un tipo *sprite*, como tal, en *libnds*. En cuanto al hardware de la videoconsola de estudio, apoya el uso de estos elementos en tanto qu e ofrece mecanismos de aceleración del tratamiento de su contenido (canales de transferencia de datos por DMA) y un área de memoria donde se almacenan los atributos: la **OAM** (*Object Attribute Memory*).

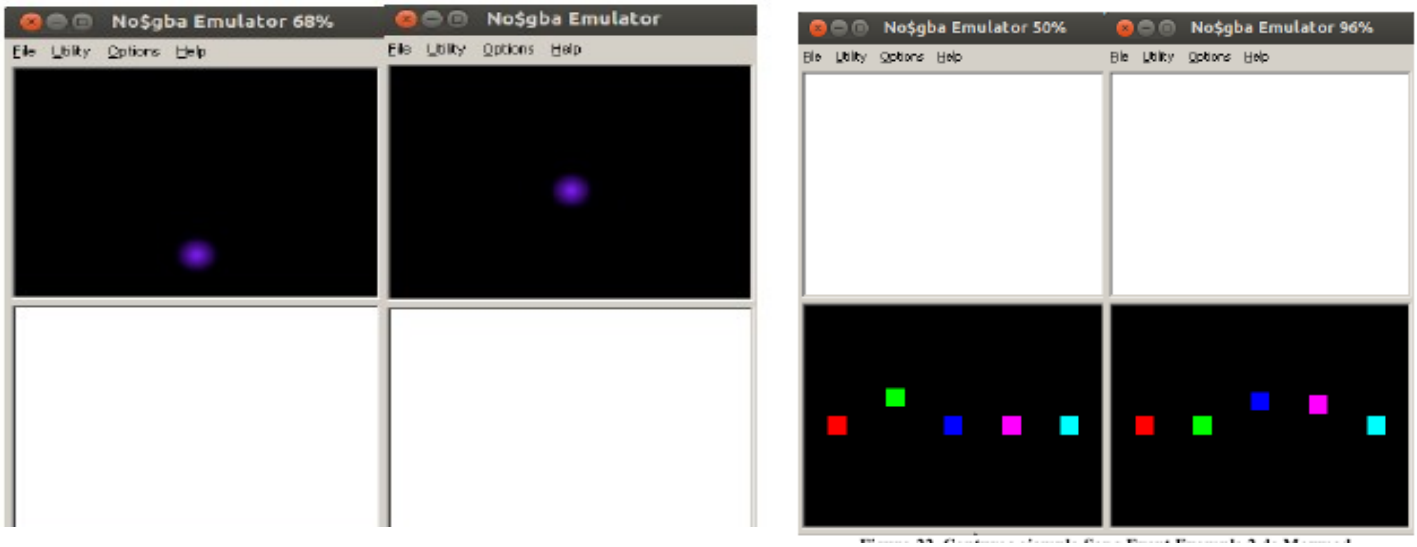


Figura 3: Capturas del movimiento en el ejemplo Song Event Example (izquierda) y Song Event Example 2 (derecha).

Ya en la práctica de audio se introdujeron dos ejemplos que utilizaban *sprites*: los de asignación de eventos. En estos, la canción de fondo representada en un en formato de los que se denominan modulares (IT en estos casos), incorporaba las instrucciones que deberían ser atendidas por los elementos gráficos para demostrar que estos pueden moverse en sincronismo con la banda sonora sin estar escrito en el código del programa.

Ejercicio: ¿Utilizaban estos ejemplos un gráfico generado en tiempo de ejecución o importado desde un mapa de bits? Para ello, busque y anote, en los ejemplos *song_events_example*[4] y *song_events_example2*[5] las instrucciones relativas al uso de *sprites*: asignación de espacio de memoria con *oamAllocateGfx* y rellenado de esa misma estructura de datos.

4 Búsquelo en ${DEVKITPRO}/examples/audio/maxmod/song_events_example/.
5 Disponible en ${DEVKITPRO}/examples/audio/maxmod/song_events_example2/

En los ejemplos de código de la sección *${DEVKITPRO}/examples/Graphics/Sprites* se pueden encontrar las piezas básicas con las que explorar el modo de funcionamiento del hardware de apoyo de la videoconsola de estudio y familiarizarse con las secuencias de código usuales.

Ejercicio: En el ejemplo *allocation_test* se puede observar una forma de cómo medir el peso de los *sprites* en uso, así como también la secuencia de operaciones básicas sobre ellos. Tome una instantánea de la aplicación y anote la estructura de datos que se utiliza para llevar cuenta de los *sprites* que se han creado y las funciones con prefijo "oam" que utilizan, junto a su definición (los prototipos de las funciones incluyendo los parámetros que reciben).

Ejercicio: En el ejemplo *simple* se puede experimentar con la forma en que los *sprites* se mueven en pantalla observando la forma en que un objeto en cada pantalla se mueve hasta alcanzar la posición señalada con el puntero. Tome una instantánea de la aplicación, anote la estructura de datos que se utiliza para llevar cuenta de los *sprites* que se han creado y las funciones que hacen posible obtener el dato de posición del cursor y cambiar la posición de los *sprites*. Qué cambia de un *sprite* a otro por estar en pantallas diferentes?

Ejercicio: En el ejemplo *fire_and_sprites* se puede observar tanto sprites generados sintéticamente (en tiempo de ejecución) como otros a partir de un mapa de bits en un fichero. Tome una instantánea de la aplicación, anote la estructura de datos que se utiliza para llevar cuenta de los *sprites* que se han creado y explique cual es generado y cuál es cargado a partir de fichero.

4 Gráficos en 3D

El hardware de la videoconsola de estudio puede [6] renderizar hasta 6144 vértices por cuadro (aproximadamente 2048 triángulos o 1536 *quads,* por cuadro), a 60 cuadros por segundo (fps, *frames per second*-), en una de las dos pantallas. Se dispone de 512 kilobytes de memoria para la textura y la más grande que se puede utilizar puede ser de 1024x1024 píxeles. El funcionamiento del subsitema de 3D se comporta de manera muy similar a un autómata de estados de OpenGL, donde la aritmética de coma flotante se ha sustituido por coma fija.

En los ejemplos de código de la sección *${DEVKITPRO}/examples/Graphics/3D* se pueden encontrar las piezas básicas con las que explorar el modo de funcionamiento 3D del hardware de apoyo de la videoconsola de estudio y familiarizarse con las secuencias de código usuales.

Dada la complejidad de la representación tridimensional, limitaremos el estudio en este apartado a explorar un par de ejemplos para tener en pantalla un "icono tridimensional", un icono que puede girar sobre su eje vertical mostrando una imagen en la parte delantera y trasera, con una cierta separación entre ellas que le confiere el aspecto 3D que caracteriza a la reciente versión NDS 3D y que se ha convertido en habitual en aplicaciones y hasta en el modo de presentar las aplicaciones en las últimas versiones del firmware de la videoconsola.

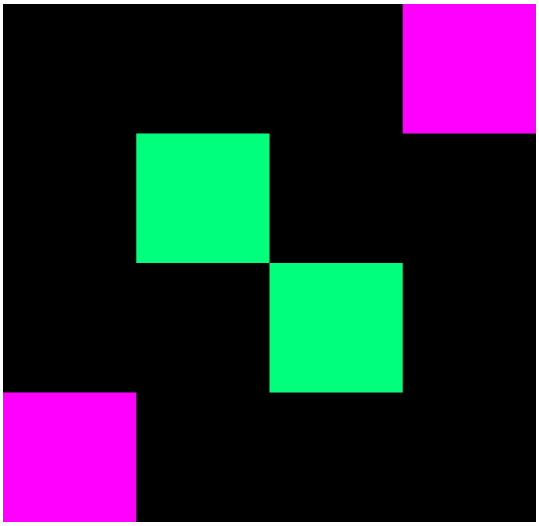
Ejercicio: En el ejemplo *Textured_Quad* se puede explorar la forma de cargar una imagen en formato *raw 128x128 16 bit*, mientras que en el ejemplo *Ortho*, se puede explorar sobre la carga de un contenido gráfico en PCX sobre las caras de un cubo.

Se puede modificar para mostrar una imagen propia si sustituimos el fichero gráfico en cada caso por uno propio en el formato adecuado. Analice los ejemplos buscando y anotando los prototipos de las funciones que hacen posible la carga de la imagen en cada caso, así como la secuencia común de instrucciones en los dos casos para inicializar la representación en 3D. genes Tome una instantánea

de los ejemplos originales y del resultado obtenido al cambiar la imagen que muestran.

5 Trabajo autónomo

En el apartado de descripción de formatos de almacenamiento, visto en las sesiones de teoría, se ha mostrado un ejemplo de fichero de texto (también existe la variante en binario) de los que componen la familia *Netpbm*: [4] en concreto, la fig. 4, recuerda el contenido de un *portable pixmap format* (PPM). El interés de este formato reside en que al poderse codificar en texto plano y tener una estructura muy sencilla, permite escribir código que los genere o los decodifique con facilidad y sin atender a cuestiones particulares de la plataforma de trabajo.



```
P3
# feep.ppm
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
```

Figura 4: Ejemplo de imagen (izquierda) y el contenido del fichero de texto que la codificada en PPM.

Ejercicio: Realizar una versión muy simple de una aplicación de dibujo para la NDS, que llamaremos *simplePaintDS*. De forma que ofrezca:

- En la pantalla inferior, un área de dibujo que implementaremos mediante un modo de acceso directo a la memoria de vídeo (*framebuffer*). Se sugiere reutilizar el código del ejemplo *starfield* visto en clase de teoría [5].
- Inicialmente y en respuesta a la pulsación del botón **B**, rellenada en blanco.
- Que va a guardar la interacción con el puntero (*stylus*) en un color dado por defecto.
- El color por defecto, se podrá cambiar con la pulsación del botón **A**, que generará un nuevo valor de color mediante la función aleatoria *rand*.
- En respuesta a la pulsación del botón **SELECT**, la aplicación guardará el contenido actual del dibujo en un fichero externo, situado en el directorio raíz (“/”) del sistema de archivos FAT de la tarjeta de memoria SD. El formato del mismo será PPM en texto plano y el nombre lo deberá introducir el usuario mediante el teclado que ofrece el API de *libnds*..

Compruebe si el dibujo resultante puede ser abierto por una aplicación en un computador como *Gimp*.

6 Bibliografía

[1] Libnds Documentation. [Disponible en línea] <http://libnds.devkitpro.org/index.html>.

[2] Gimp, sitio web. [Disponible en línea] <http://www.gimp.org/>

[3] E. Asensio. 2013. Estudio práctico de los API no oficiales de desarrollo e interacción con audio para Nintendo DS sobre GNU/L inux. PFC DISCA -.ETS d'Enginyeria Informàtica.

[4] Netpbm format. 2014. [Disponible en línea] <http://en.wikipedia.org/wiki/Netpbm_format>.-

[5] NDS/Tutorials Day 3. [Disponible en línea]
<http://www.dev-scene.com/NDS/Tutorials_Day_3>

[6] "OSDL. A guide to homebrew development for the Nintendo DS". [Disponible en línea]
<http://osdl.sourceforge.net/main/documentation/misc/nintendo-DS/homebrew-guide/HomebrewForDS.html>

[7] H. Cuñat. 2011. MoonSeeDS. Didáctica espacial en la Nintendo DS. PFC DISCA - ETS d'Enginyeria Informàtica.

[8] Charas, rpgmaker, the chara generator <http://charas-project.net/resources.php?lang=en

7 Anexo A: GRIT

> Once we have our Sprite Strip created, just like with Tiled Backgrounds, we put it in GRIT's bmp folder. Once our Sprite or Sprites are in it, we run the Convert_Sprites batch file, and after the process is complete, the resulting binary files representing the Tiles and the Palette data can be recovered from the sprites folder. Alternatively, you can convert Sprites using Joint Palettes – this way, you can use several Sprites using just one Palette. Keep in mind though that it limits the total number of colours you can use to 256 rather than 256 per Sprite, however when using a large number of them, you may end up having to resort to that.
>
> Foxi4 <http://gbatemp.net/threads/ds-programming-for-newbies.322106/page-8>

Descripción obtenida de la página[6] del autor, Jasper Vijn: *Grit and its GUI version Wingrit (fig. 5) are my image converters for the GBA (and NDS I guess). They can do most of the simple things like reading an image (pretty much any type of bitmap thanks to FreeImage) and converting it to binary data of various bitdepths which can be directly put into VRAM, but also more complicated matters such as tiling and metatiling (for 1D object mapping for example), making a tilemap along with a reduced tileset (or using an external tileset), popular map layouts, and compression compatible with the GBA's BIOS routines. The capability for an NDS alphabit has been added recently as well. Output can be C/asm arrays, raw binary, GBFS, and a RIFF-based format called GRF.*

Not good enough? Well, the source code is available too, so you're free to modify it. The code should be platform independent right now, or at least very nearly so; the catch being that you might have add or remove some type definitions and maybe create your own makefile for compilation.

6 GRIT | Coranac <http://www.coranac.com/projects/grit/>.

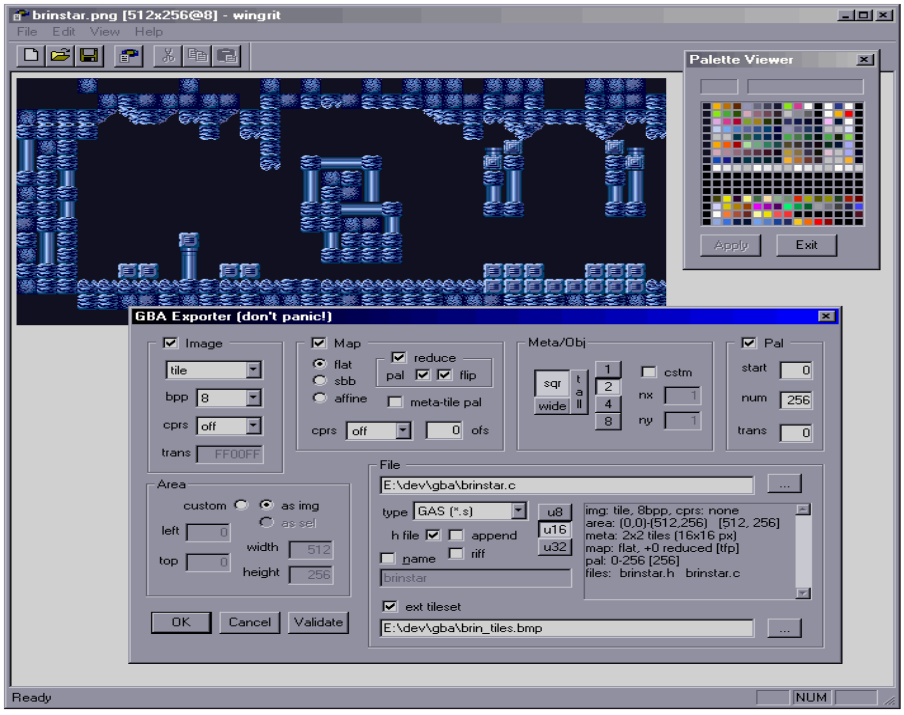


Figura 5: GRIT 0.8.6 - GBA Raster Image Transmogrifier en su versión gràfica para MS/Windows.

GRIT (como MMUTIL que se utilizó para el caso del sonido) es parte de las herramientas de la distribución de *devkitARM*. Se ejecuta al compilar un proyecto que tiene en su *Makefile* una regla que lo utiliza para convertir los ficheros de formato gráfico a uno de los nativos (soportados directamente por el hardware) de la consola. Se puede ejecutar directamente, obteniendo la profusa descripción que se transcribe a continuación.

```
$  ${DEVKITPRO}/devkitARM/bin/grit
---grit v0.8.8 ---
GRIT: GBA Raster Image Transmogrifier. (grit v0.8.8, 20110622)
  Converts bitmap files into something the GBA can use.
usage: grit srcfile(s) [args]


--- Graphics options (base: "-g") ---
-g | -g!       Include  or exclude gfx data [inc]
-gu(8|16|32)   Gfx data type: u8, u16, u32 [u32]
-gz[!lhr0]     Gfx compression: off, lz77, huff, RLE, off+header [off]
-gb | -gt      Gfx format, bitmap or tile [tile]
-gB{n}         Gfx bit depth (1, 2, 4, 8, 16) [img bpp]
-gS            Shared graphics
-gT{n}         Transparent color; rrggbb hex or 16bit BGR hex [FF00FF]
-al{n}         Area left [0]
-ar{n}         Area right (exclusive) [img width]
-aw{n}         Area width [img width]. Overrides -ar
-at{n}         Area top [0]
-ab{n}         Area bottom (exclusive) [img height]
-ah{n}         Area height [img height]. Overrides -ab

--- Map options (base: "-m") ---
-m | -m!       Include or exclude map data [exc]
```

```
-mu(8|16|32)   Map data type: u8, u16, u32 [u16]
-mz[!lhr0]     Map compression: off, lz77, huff, RLE, off+header [off]
-ma{n}         Map-entry offset n (non-zero entries) [0]
-mp{n}         NEW: Force mapsel palette to n
-mB{n}:{(iphv[n])+}     NEW: Custom mapsel bitformat
-mR{t,p,f}     Tile reduction: (t)iles, (p)al, (f)lipped
                 options can be combined [-mRtpf]
-mR[48a]       Common tile reduction combos: reg 4bpp (-mRtpf),
                 reg 8bpp (-mRtf), affine (-mRt), respectively
-mR!           No tile reduction (not advised)
-mL[fsa]       Map layout: reg flat, reg sbb, affine [reg flat]

--- Palette options (base: "-p") ---
-p | -p!       Include or exclude pal data [inc]
-pu(8|16|32)   Pal data-type: u8, u16 , u32 [u16]
-pz[!lhr0]     Pal compression: off, lz77, huff, RLE, off+header [off]
-ps{n}         Pal range start [0]
-pe{n}         Pal range end (exclusive) [pal size]
-pn{n}         Pal count [pal size]. Overrides -pe
-pS            shared palette
-pT{n}         Transparent palette index; swaps with index 0 [0]
--- Meta/Obj options (base: "-M") ---
-Mh{n}         Metatile height (in tiles!) [1]
-Mw{n}         Metatile width (in tiles!) [1]
-MRp           Metatile reduction (pal only) [none]

--- File / var options ---
-ft[!csbgr]    File type (no output, C, GNU asm, bin, gbfs, grf) [.s]
-fr            Enable GRF-format for .c or .s
-fa            File append
-fh | -fh!     Create header or not [create header]
-ff{name}      Additional options read from flag file [dst-name.grit]
-fx{name}      External tileset file
-o{name}       Destination filename [based on source]
-s{name}       Symbol base name [based from dst]
-O{name}       Destination file for shared data
-S{name}       Symbol base name for shared data

--- Misc ---
-tc            Tiling in column-major order.
-tw            NEW(?): base tile width [8].
-th            NEW(?): base tile height [8].
-U(8|16|32)    All data type: u8, u16, u32
-W{n}          Warning/log level 1, 2 or 3 [1]
-Z[!lhr0]      All compression: off, lz77, huff, RLE, off+header [off]

New options: -fr, -ftr, -gS, -O, -pS, -S, -Z0 (et al)
```