Guía de desarrollo para Nintendo DS

Iván López Rodríguez

Índice:

PRÓLOGO

Lo que pretendo con esta guía es presentar las herramientas libres de las que se dispone cuando se quiere desarrollar *homebrew*, en este caso, para Nintendo DS.

Homebrew es el nombre que reciben las aplicaciones y juegos no oficiales creados generalmente para consolas de videojuegos propietarias.

En el primer apartado de la guía se hablará de la propia consola y su hardware, que será necesario conocer para saber qué límites tenemos al programar para ella. Después se verá cómo preparar el entorno de desarrollo tanto en sistemas Windows como Linux. Se usará devkitPro como kit de desarrollo de software (SDK) y veremos cómo configurar el entorno de desarrollo integrado (IDE) Eclipse para este SDK. Además, utilizaremos PAlib como librería de alto nivel para usar las herramientas que nos presenta devkitPro.

Conocidos el hardware de la consola e instalados y configurados el SDK y el IDE, se verán las herramientas que nos ofrecen devkitPro y PAlib. Con ella se introducirán los temas básicos para programar en NDS: texto, gráficos, entrada de usuario y sonidos. Por supuesto, cada apartado contará con pequeñas aplicaciones como ejemplo de uso.

Visto ya lo básico, para cerrar el proyecto se tratarán librerías con funciones más avanzadas, como *DSwifi* para las funciones de red o *libfat* para leer/escribir en ficheros.

1 INTRODUCCIÓN

LA CONSOLA

Historia y generalidades:

Nintendo DS es la videoconsola portátil de Nintendo perteneciente a la séptima generación de consolas (hasta ahora, las generaciones venían marcadas por las consolas de sobremesa). Salió al mercado en 2004 en Japón y EE.UU. llegando a Europa el 11 de marzo de 2005. En enero de 2011, la Nintendo DS se ha convertido en la videoconsola más vendida de la historia, adelantando a Sony PlayStation 2.

Hasta la fecha, han salido cuatro versiones de la consola (figura 1): la original; DS Lite, reduciendo el tamaño; DSi, con mejoras hardware y dos cámaras; y DSi XL, un modelo más grande de la DSi que incluye pantallas más grandes que su predecesora.

Figura 1: Nintendo DS, Nintendo DS Lite, Nintendo DSi y Nintendo DSi XL. Fuente: Wikipedia.

El diseño de la consola recuerda a los Game & Watch Multi-Screen de Nintendo de los años 80, teniendo dos pantallas en un dispositivo 'clamshell' (dispositivos electrónicos que se pliegan mediante una bisagra), siendo la característica más definitoria de la DS el hecho de que la pantalla inferior sea táctil.

Por último, los juegos de la consola vienen en el formato de 'Nintendo DS Game Card', un cartucho diseñado por Nintendo específicamente para la videoconsola. Tienen unas dimensiones de 33 mm x 35mm x 3.8mm y una capacidad desde 8MB hasta 512MB (cartuchos existentes hasta la fecha), además de una pequeña cantidad de memoria flash para almacenar partidas guardadas y otros datos de jugador. Los modelos DS y DS Lite tienen además una ranura compatible con cartuchos de Game Boy Advance, mientras que los dos modelos DSi incluyen una ranura para tarjetas SD.

En el primer trimestre de 2011 ha salido a la venta la sucesora como videoconsola portátil de Nintendo, la Nintendo 3DS, con pantalla con más potencia que toda la gama de DS y la capacidad de reproducir 3D estereoscópico sin necesidad de utilizar gafas.

Hardware:

Estas son las características que ofrece la Nintendo DS original y Lite (traducción libre del enlace **[1]**):

- Dos pantallas retroiluminadas de 3 pulgadas separadas por 21mm, con una resolución máxima de 256x192 píxeles cada una, con hasta 260000 colores (5 bits para cada canal).
- La pantalla inferior es táctil y reconoce varias pulsaciones simultáneas, devolviendo un único resultado, el baricentro.
- Dos procesadores que pueden ejecutar código simultáneamente:

  - ARM9: ARM946E-S, la CPU principal, 67 MHz, entre 200 y 300 MIPS, arquitectura RISC de 32 bits.

  - ARM7: ARM7TDMI, coprocesador, 33 MHz, sobre 20MIPS, arquitectura RISC de 16/32 bits.

- 4 megabytes de RAM integrada que en su mayor parte estan compartidos por ambos procesadores. Además, 656 KB de RAM para vídeo y un poco de memoria no volátil para las preferencias del usuario.
- Una GPU compuesta de dos sistemas de renderizado 2D (uno para cada pantalla) y un sistema de renderizado 3D que puede renderizar hasta 120000 triángulos por segundo a una tasa de 60 fps y una tasa de rellenado de 30 millones de píxeles por segundo. La GPU está integrada en el mismo chip que ambos procesadores.
- Un pad de dirección y 8 botones (A, B, X, Y en forma de cruz; L, R en los laterales; y Start y Select, a cada lado de la pantalla).
- Tarjeta de red integrada, capaz de ofrecer el protocolo propietario NiFi (Nintendo wifi) y 802.11b wifi, respectivamente para comunicación entre consolas y consola-internet, con un rango de entre 10 y 30 metros.
- Una salida de audo de 16 canales con altavoces estéreo y una salida estándar para auriculares.
- Un micrófono.
- Dos slots para cartuchos flash, el Slot-1 para cartuchos específicos de DS y el Slot-2 compatible con cartuchos de GameBoy Advance.
- Bateria de ión-litio.
- Reloj integrado de 33 MHz, que se encarga de mantener la hora y fecha incluso cuando la NDS esta apagada.
- Dos LEDs informativos, uno sobre el estado de la batería, el otro sobre la wifi.

Las versiones DSi presentan las siguientes diferencias:

- La frecuencia de reloj del procesador ARM9 aumenta a 133MHz.
- La memoria RAM integrada aumenta a 16MB.
- Desaparece el Slot 2.
- Se añade una ranura para tarjetas SD.
- Se añaden dos cámaras de 0.3 Megapíxeles, una interna y otra externa.

HOMEBREW EN NDS

Pese a que oficialmente DS significa tanto Dual Screen como Developer's System, la consola no permite desarrollar libremente aplicaciones para ella. Por ello el creador de homebrew necesitará ciertas herramientas para que sus aplicaciones se puedan ejecutar en la Nintendo DS.

Flashcarts:

Son cartuchos creados por terceras compañías, compatibles con los slot-1 ó slot-2 de la consola. Su función es cargar binarios de homebrew o copias de seguridad mediante memoria reescribible.

Los flashcarts de slot-2, más antiguos, suelen requerir parchear los ejecutables de la consola (llamados ROMs por ser imágenes del contenido de un cartucho de memoria ROM) y usar un *Passme* (una técnica creada por la *scene* de Nintendo DS para simular la autentificación del software a cargar) o alterar el firmware de la consola.

Los flashcarts de slot-1 en cambio no suelen requerir parchear las ROMs, ni necesitan técnicas adicionales para ejecutar homebrew o copias de seguridad. Además, como ventaja adicional, deja libre el slot-2, donde pueden conectarse extras, como ampliación de la memoria de la consola.

La legalidad de los flashcarts ha sido puesta en duda por Nintendo en varias ocasiones, pero (en España) los jueces han dado siempre la razón a los vendedores/usuarios de estos cartuchos.

Prácticamente la totalidad de flashcarts en el mercado actual, tanto de slot-1 como de slot-2, permiten la carga de homebrew sin problemas.

En esta guía se utilizará un cartucho R4 con el firmware Wood R4 1.23.

Figura 2: R4 for NDS, imagen encontrada en ElOtroLado.net

Emuladores:

Un emulador es un software que permite ejecutar programas en una plataforma diferente de aquella a la que el software iba dirigido. En nuestro caso, será recomendable utilizar emuladores de NDS para probar nuestras aplicaciones sin necesidad de estar volcándolas a la flashcart después de cada modificiación.

Algunos emuladores, como DeSmuMe o no$gba en su versión comercial, incluyen además herramientas útiles para la depuración del software implementado.

SDK: DEVKITPRO Y PALIB

Dado que oficialmente Nintendo no proporciona herramientas adecuadas para la programación de aplicaciones para NDS a usuarios fuera de equipos de desarrollo de juegos de NDS, recurriremos a devkitPro, un SDK disponible para Game Boy Advance, Nintendo DS, PSP, GP32, GameCube y Wii.

Instalación de devkitPro en Windows

El equipo de desarrollo de devkitPro ha creado un auto-instalador que facilita mucho la instalación del SDK en sitemas Windows. El primer paso será descargarse el ejecutable (devkitPro Updater 1.5.0) y abrirlo.

Tras comprobar si existen actualizaciones, el instalador preguntará si quieres instalar o solo descargar los archivos (Figura 3). Selecciona instalar.

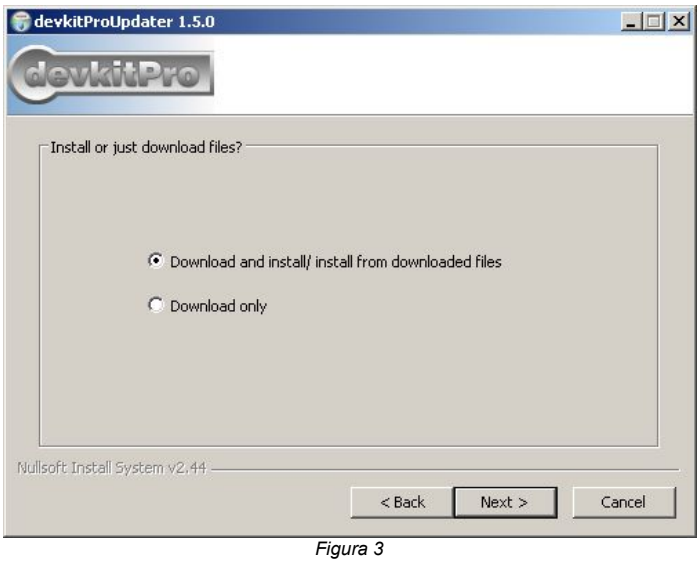


Figura 3

A continuación, preguntará si quieres conservar o eliminar los archivos descargados para la instalación, como no son necesarios, se borrarán.

En la siguiente pantalla (fig. 4) nos da a elegir qué componentes instalar. En nuestro caso nos interesa instalar únicamente el compilador DevKitArm, el compilador para NDS y GBA, además del sistema mínimo. DevKitPPC es el compilador para el homebrew de Wii y GameCube y DevKitPSP, para Sony PSP. Programmer's Notepad e Insight son un IDE y un debugger respectivamente; como más adelante configuraremos otro IDE, no hace falta instalarlo.

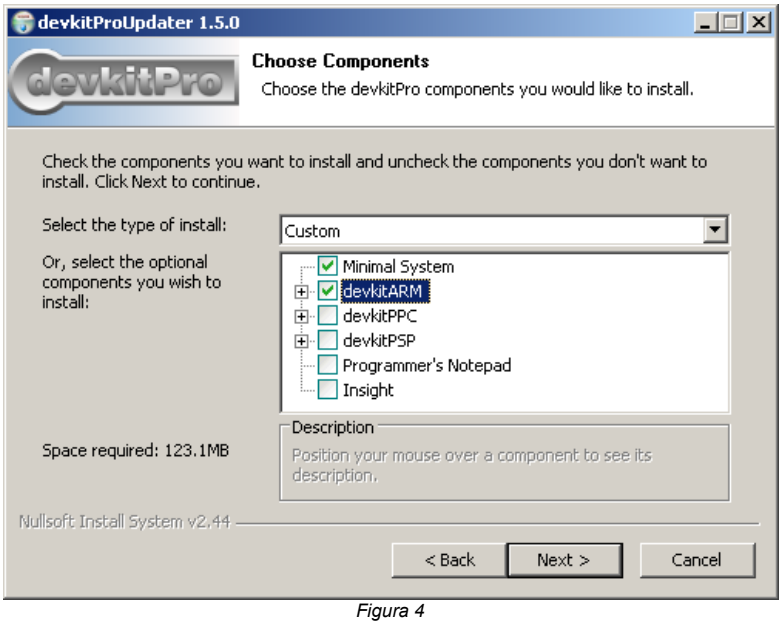
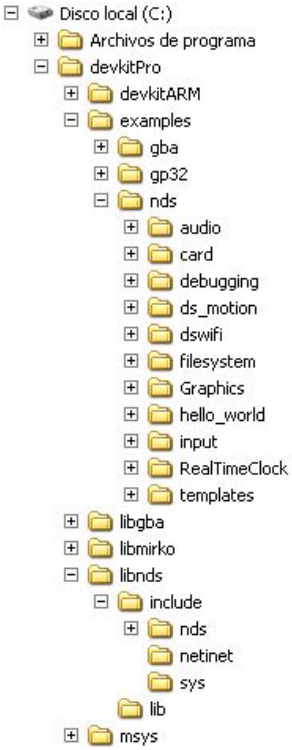


Figura 4

Tras elegir qué elementos instalar, nos solicita la ruta, por defecto *C:\devkitpro*. **Nota:** el compilador puede tener problemas si usas una ruta con espacios.

Al acabar, tendremos una estructura de directorios como la que sigue:



Para comprobar que se ha instalado correctamente, abriremos la consola de Windows, iremos a la ruta donde hemos instalado el DevKitPro, y ahí a .\examples\nds\hello_world. Desde esa ruta, ejecutaremos la orden *make* y si todo ha ido bien, el ejemplo se compilará.

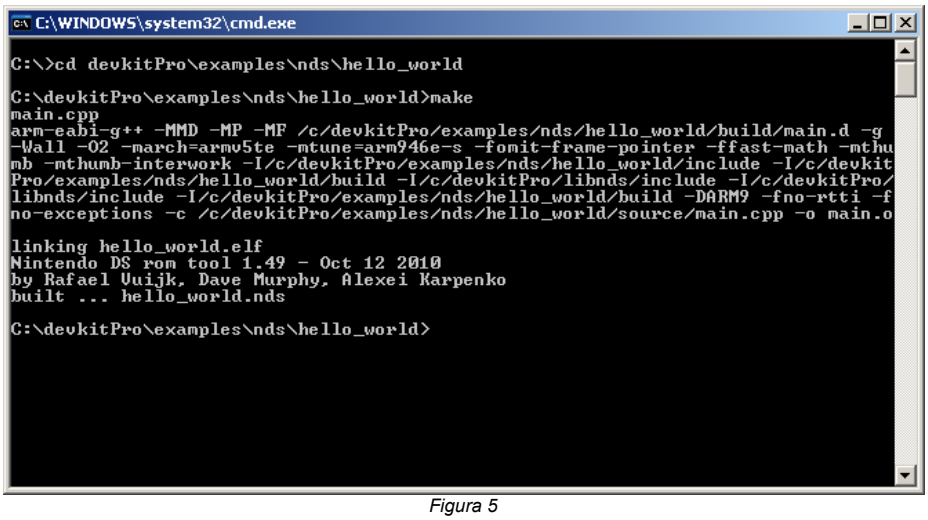


Figura 5

El hello_world.nds resultante se puede ejecutar con un emulador o en la DS con una flashcart para confirmar que efectivamente todo ha funcionado como debía.

Instalación de devkitPro en Linux

Para instalar DevKitPro en Linux, primero deberemos bajar todos los archivos necesarios para el sistema y la arquitectura donde vamos a instalarlo (enlaces en **[6]**):

- devkitArm - El compilador.
- libnds - La librería principal.
- libfat-nds - Librería para manejar ficheros.
- dswifi - Librería para las funciones de red.
- maxmod - Librería de sonido.
- libfilesystem
- Default arm7
- Ejemplos NDS

Una vez descargados, creamos la carpeta donde se alojará y descomprimimos el compilador, por ejemplo en /opt.

```
sudo mkdir -p /opt/devkitpro
sudo chmod 777 /opt/devkitpro

cd /opt/devkitpro

tar -xvjf [devkitARM].tar.bz2
```

Tras ello, creamos una carpeta llamada libnds en la ruta del devkitpro y descomprimimos en ella todas las librerías y los datos del ARM7:

```
mkdir libnds
cd libnds

tar -xvjf [libnds].tar.bz2

tar -xvjf [libfat-nds].tar.bz2

tar -xvjf [dswifi].tar.bz2

tar -xvjf [maxmod].tar.bz2

tar -xvjf [libfilesystem].tar.bz2

tar -xvjf [default arm7].tar.bz2
```

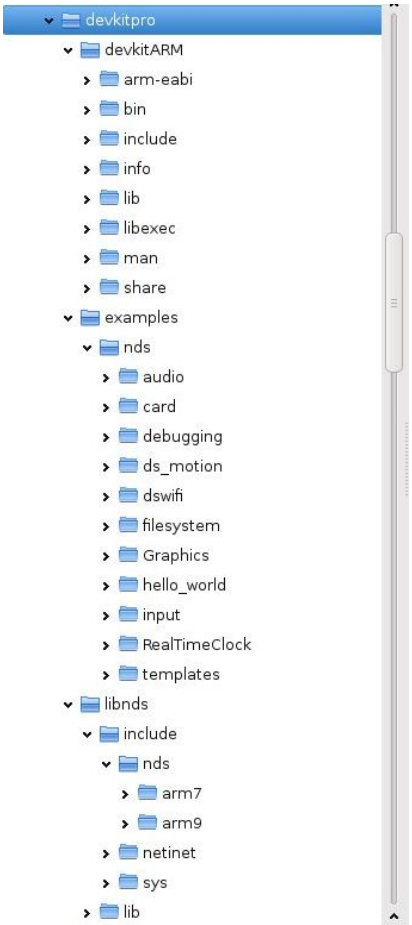
Para los ejemplos, creamos una carpeta propia y los descomprimimos en ella:

```
cd ..
mkdir -p examples/nds

cd examples/nds

tar -xvjf [ejemplos nds].tar.bz2
```

El árbol de carpetas resultante se parecerá al que sigue (puede variar entre versiones):



Por último, hay que añadir las variables al entorno, editando el .bashrc, añadiendo al final las siguientes líneas:

```
export DEVKITPRO=/opt/devkitpro
export DEVKITARM=$DEVKITPRO/devkitARM
```

Para comprobar que la instalación se ha realizado correctamente, basta con ir a $DEVKITPRO/examples/nds/hello_world/ y ejecutar la orden *make*. Si todo va bien, tendremos un *hello_world.nds* listo para probar en emulador o en consola con flashcart.

Nota: se recomienda quitar al grupo *Otros* el permiso de escribir una vez acabada la instalación.

Nota: Las versiones de las librerías para este documento son:

- libnds: 1.5.0
- libfat-nds: 1.0.9
- dswifi: 0.3.13
- maxmod: 1.0.6
- libfilesystem: 0.9.9
- Default arm7: 0.5.20

Instalación de PAlib

La instalación de PAlib es idéntica en sistemas Linux y Windows, ya que solo requiere bajarse la última versión (en nuestro caso *PAlib 100707 "QuickFix", enlace* **[7]**) y descomprir el contenido del paquete en */devkitPro/PAlib/.*

El paquete PAlib trae además una colección de herramientas útiles. A continuación comentaré PAGfx, herramienta pensada para convertir imágenes a un formato adecuado para usar con PAlib.

PAGfx

PAGfx se encarga de convertir imágenes en formato de mapas de bits a RAW, que puede ser compilado directamente en el binario .nds resultante. La aplicación está realizada con la librería .NET de Microsoft, por lo que para ejecutarla en sistemas Linux se tendrá que usar la versión para mono.

En el frontend de la aplicación (fig. 6) encontraremos pestañas para cargar bien sprites, fondos o texturas, un rectángulo coloreado que nos permitirá elegir el color que representa la transparencia y el botón de *Save and Convert*, que es el que realiza la transformación.

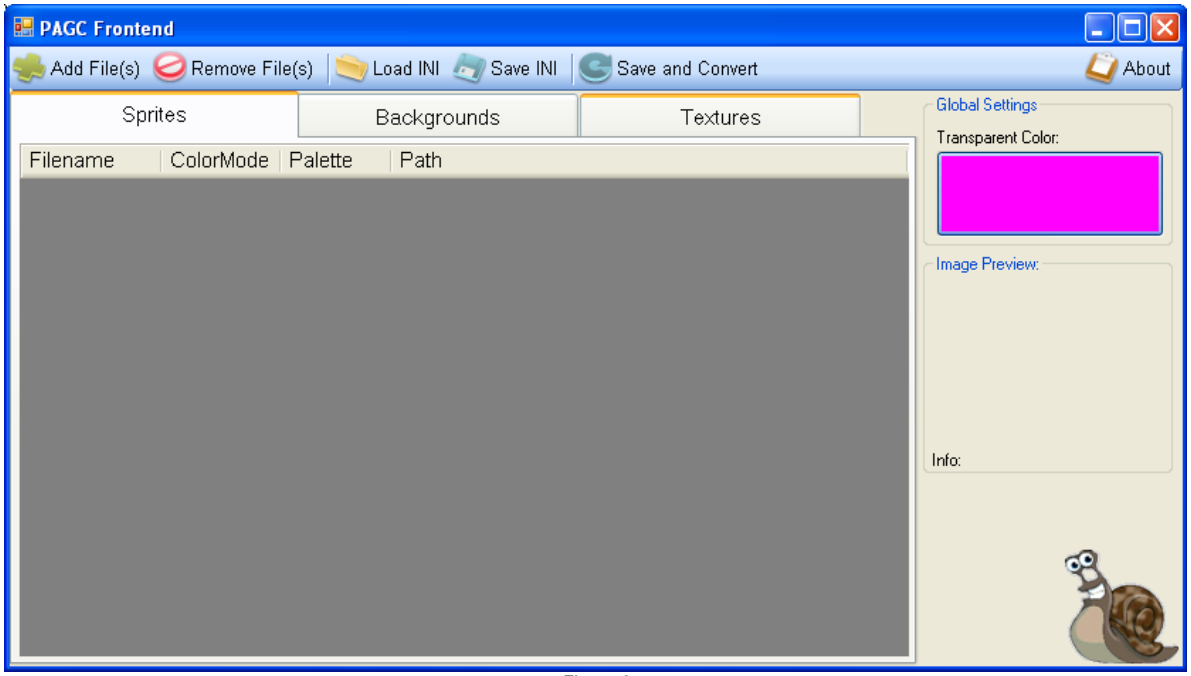


Figura 6

Al cargar una imagen en sprites o texturas, nos dará a elegir el modo de color. 256 colores suele ser el modo por defecto.

Si cargamos la imagen en fondos, podremos seleccionar el tipo de fondo, según sea por ejemplo un fondo por *tiles* o una fuente de letra.

En los tres casos, podremos definir el nombre de la paleta de colores y, dado que la DS tiene 16 paletas por pantalla, indicar que se usa la misma paleta para más de una imagen.

Al convertir todo, obtenemos en la carpeta *bin* las imágenes convertidas a formato RAW, además de una cabecera *all_gfx.h*. Colocamos la carpeta *bin* en el directorio para los gráficos (por defecto gfx), incluimos la cabecera en el código y podremos usar las imágenes en la aplicación.

ENTORNOS DE DESARROLLO

Eclipse:

Cualquier entorno de desarrollo en C/C++ sería una opción para crear proyectos para la NDS, pero suelen requerir dedicar tiempo a configurar tanto los compiladores como los ajustes necesarios de cada proyecto individual. Por ello usaremos *Eclipse* con su plugin *NDS ManagedBuilder*, que en conjunto forman un IDE pensado específicamente para el desarrollo en Nintendo DS.

Para plataformas Windows de 32 bits se puede conseguir el IDE con el plugin en la página oficial del creador del plugin **[8]**. Para otras plataformas, será necesario instalar Eclipse 3.4.0 o superior e instalar el plugin mediante el actualizador del propio Eclipse (figura 7) usando la url de actualizaciones del NDS Managed builder (http://dev.snipah.com/nds/updater).

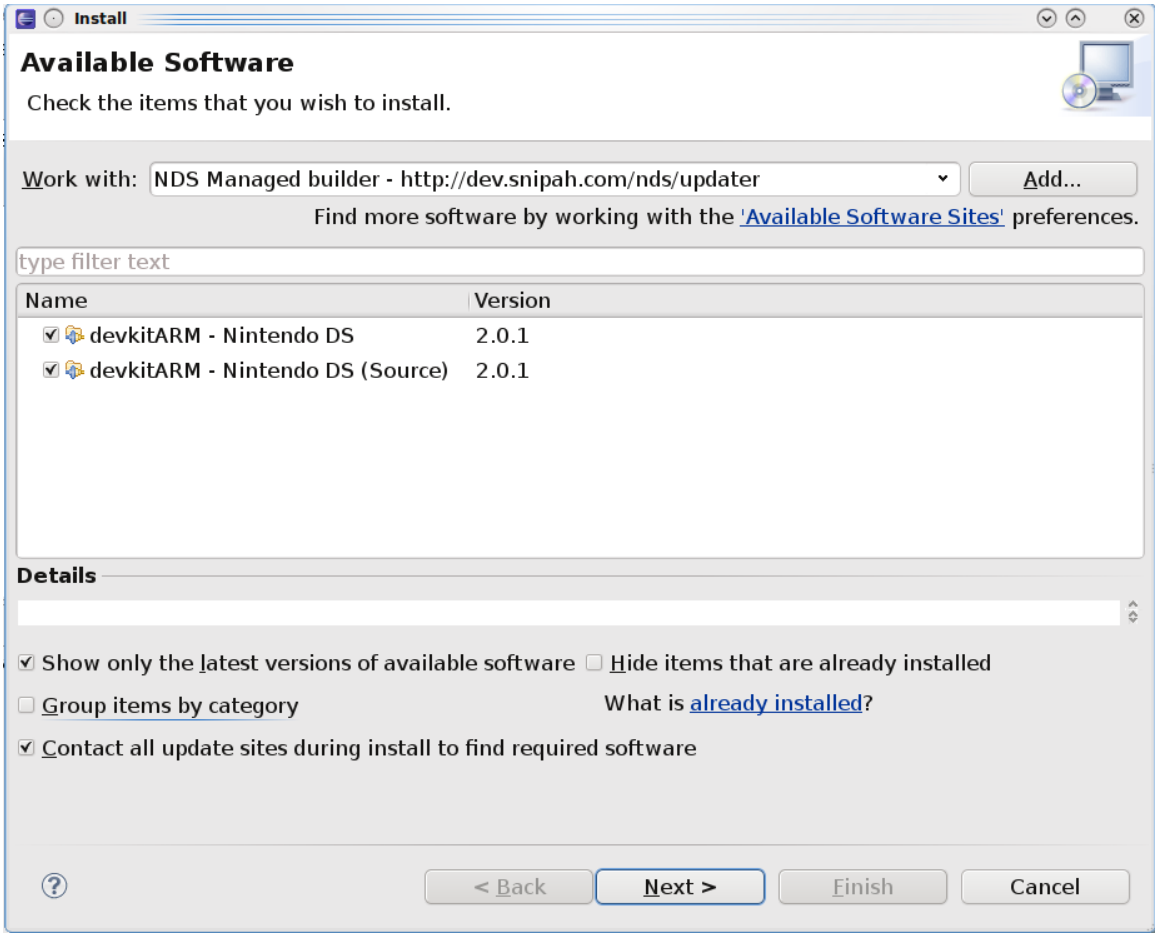


Figura 7: Instalación del plugin en la versión 3.5.1 de Eclipse para Linux. A notar que está desactivado "Agrupar por categoría".

Una vez instalado el plugin, al crear un nuevo proyecto podremos elegir crear un proyecto para NDS (Nintendo DS Rom > Simple NDS ROM Project) y tener así ya un proyecto con las dependencias necesarias para programar con devkitPro y un simple código de prueba.

Pero aún queda configurar PAlib en el IDE. Para ello, y es algo que se deberá hacer para cada proyecto que use PAlib, debemos abrir las propiedades del proyecto. Accedemos a *settings* en *C/C++ build* y en los desplegables de *devkitArm C Compiler* y *devkitArm C++ Compiler* añadimos "*${DEVKITPRO_DIR}/PAlib/include/nds"*.

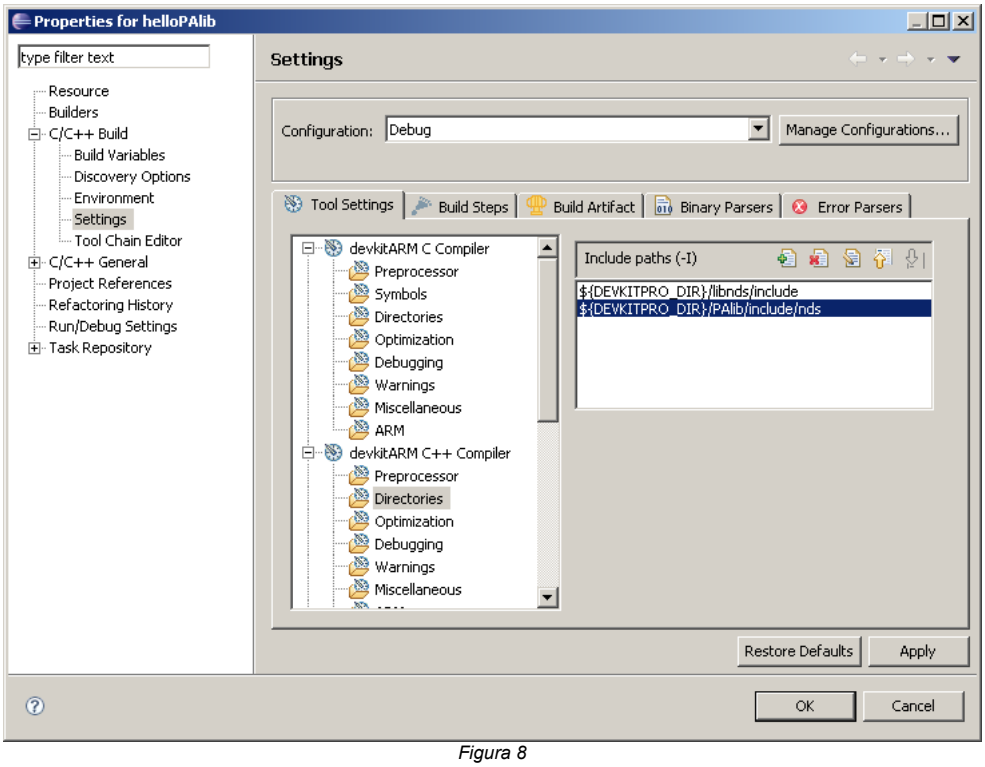


Figura 8

Para enlazar las librerías, bajamos hasta *devkitArm C++ linker* y añadimos en la ruta de búsqueda el directorio "*${DEVKITPRO_DIR}/PAlib/lib"* y en las librerías "*pa9"* y "*pa7"*, sin el prefijo *"lib"* ni la extensión ".s".

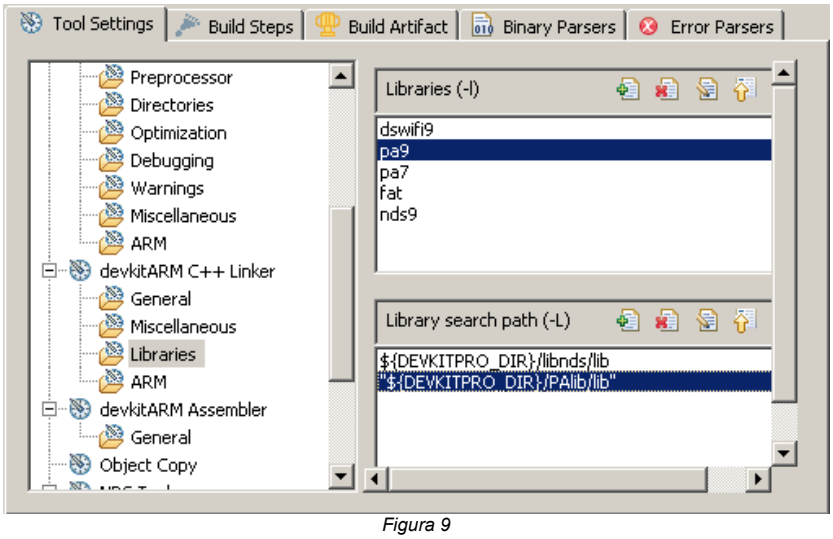


Figura 9

Para terminar, configuraremos un emulador para que desde el IDE podamos realizar las pruebas sin estar pasando constantemente la ROM al flashcart o tener que estar abriendo el emulador a mano. Para ello abriremos la configuración de herramientas externas:

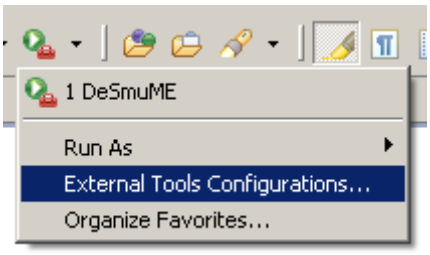


Figura 10

En la ventana emergente añadiremos en *Location* la ruta del emulador que queramos usar (en el ejemplo, DeSmuMe **[9]**), el directorio de trabajo será la carpeta *Debug* del proyecto (por variables del entorno "*${workspace_loc:/${project_name}/Debug}*") y como argumento, la ROM generada ("*${project_name}.nds*").

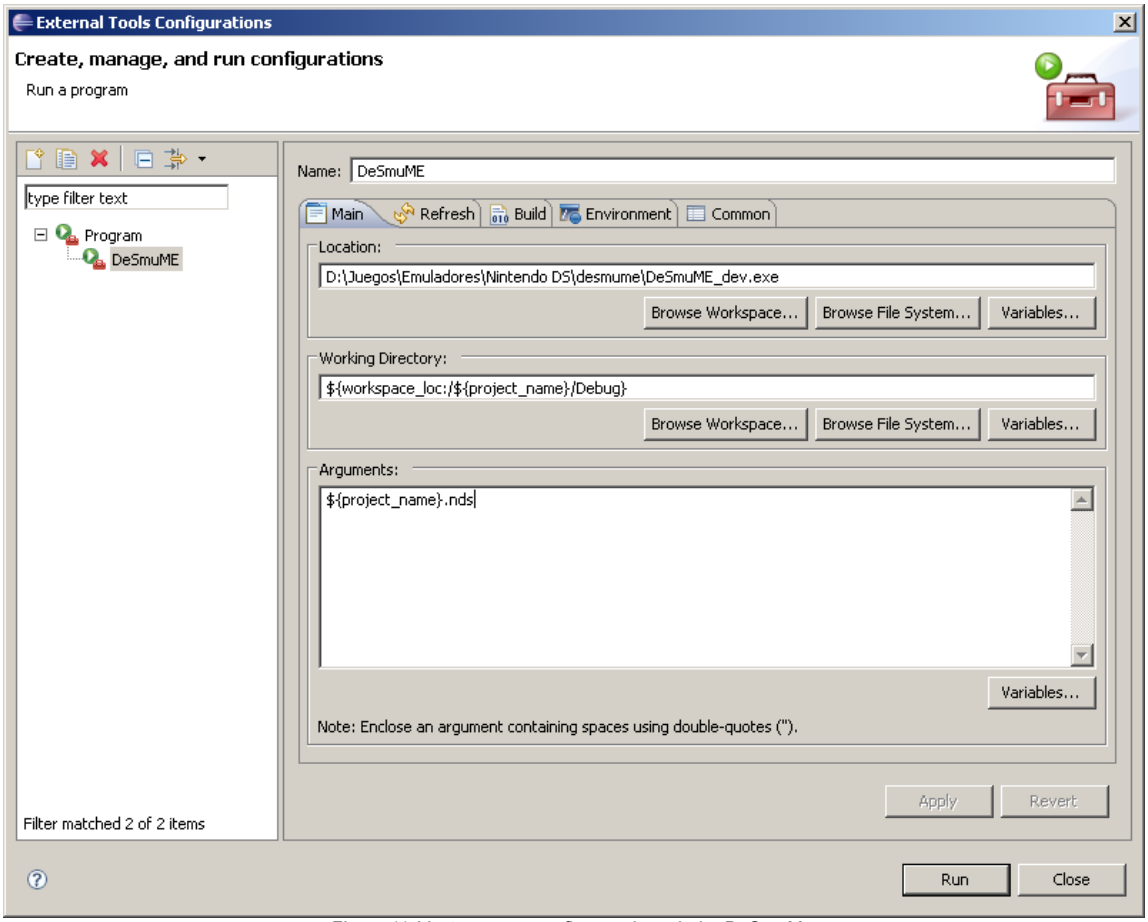


Figura 11: Ventana para configurar el emulador DeSmuMe.

Una vez acabada la configuración del IDE, para generar el proyecto basta con seleccionar *project* > *Build* o pulsar el icono del martillo.

Figura 12

Y para probar en emulador el resultado, seleccionaremos Run > External Tools > {emulador configurado} o pulsaremos directamente el botón de Play con caja de herramientas.

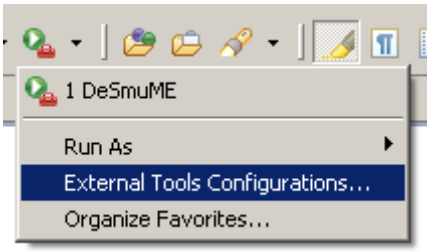


Figura 13

Nota: Si Eclipse da problemas para compilar un proyecto de PAlib por no dejar utilizar un makefile personalizado, siempre se puede recurrir a montar el proyecto siguiendo la plantilla por defecto de PAlib (se encuentra en *devkitPRO/PAlib/template/*) y ejecutando la orden *make* (la herramienta está presenteen Linux por defecto, en Windows debe estar instalado MSYS).

Si se hace de este modo, hay que respetar la organización por carpetas impuesta por PAlib, tal y como sigue:

Carpeta	Contenido
data	Archivos de datos a incluir en el proyecto.
include	Cabeceras (.h).
source	Código fuente (.c, .cpp y .s).
gfx	Gráficos generados por PAGfx.
audio	Archivos de sonido para Maxmod.
filesystem	Archivos incrustados en el .nds

El *makefile* de la plantilla, además, convierte los archivos *.raw o *.bin a las librerías y objetos apropiados para compilar y enlazar al proyecto.

Code::Blocks:

Otra opción para utilizar como entorno de desarrollo es Code::Blocks (versión 10.05). Este IDE puede obtenerse en su página oficial **[10]**. A continuación lo configuraremos para utilizar con proyectos PAlib (traducción libre de **[11]**).

En la barra de menús, seleecionamos *Tools* > *Configure tools*... con lo que nos saldrá la caja de la figura 14.

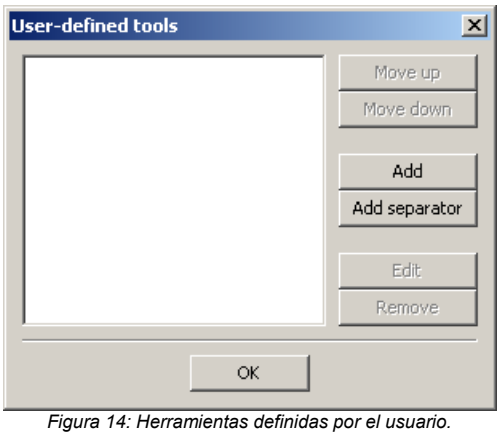


Figura 14: Herramientas definidas por el usuario.

En esta caja pulsamos el botón *Add* para añadir nuestras herramientas. Primero definimos la herramienta que actuará como compilador tal y como se muestra en la fig. 15:

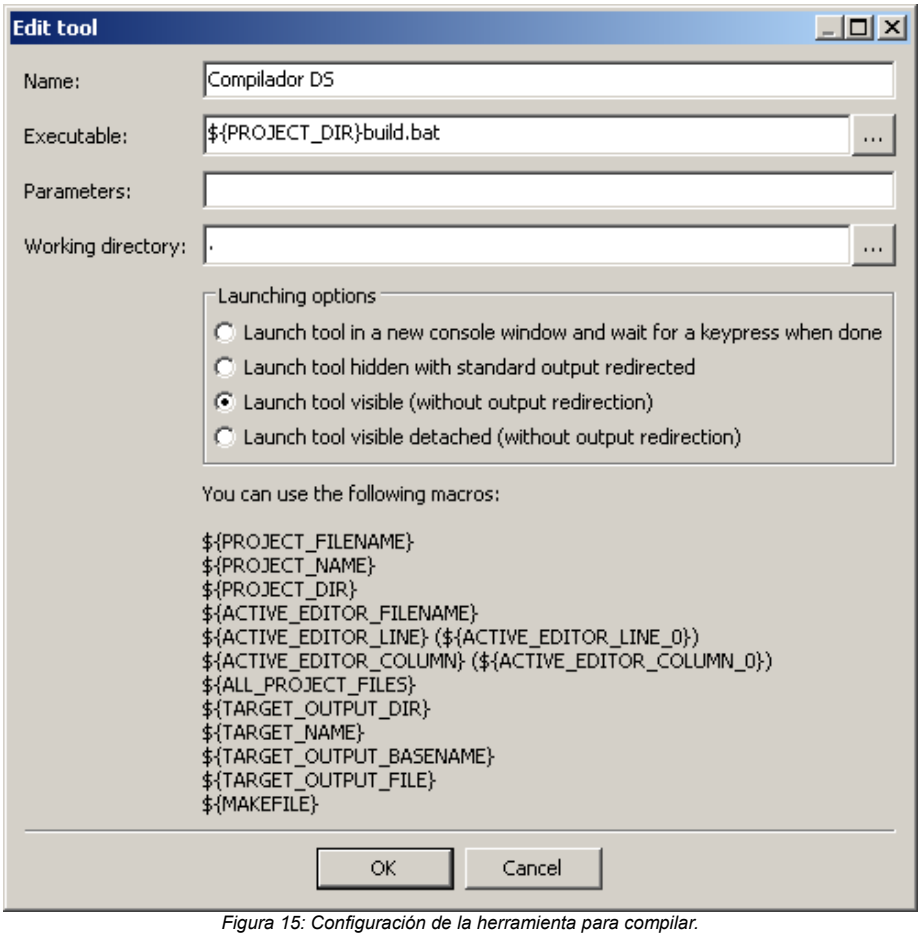


Figura 15: Configuración de la herramienta para compilar.

Para que esta herramienta funcione, debe cumplirse que los archivos *build.bat, clean.bat* y *makefile* de la plantilla de PAlib estén en el directorio del proyecto. Además, en Windows es necesario tener instalado el paquete MSYS (viene con devkitPro) y en Linux se necesita añadir el permiso de ejecución a los archivos *.bat de la plantilla.

Para compilar usaremos *Tools* > *Compilador DS* en el menú.

Del mismo modo podemos añadir una herramienta para ejecutar la ROM compilada en el emulador de nuestra elección. Por ejemplo, en la figura 16 se muestra la configuración de una herramienta de Code::Blocks para lanzar el proyecto en DeSmuME:

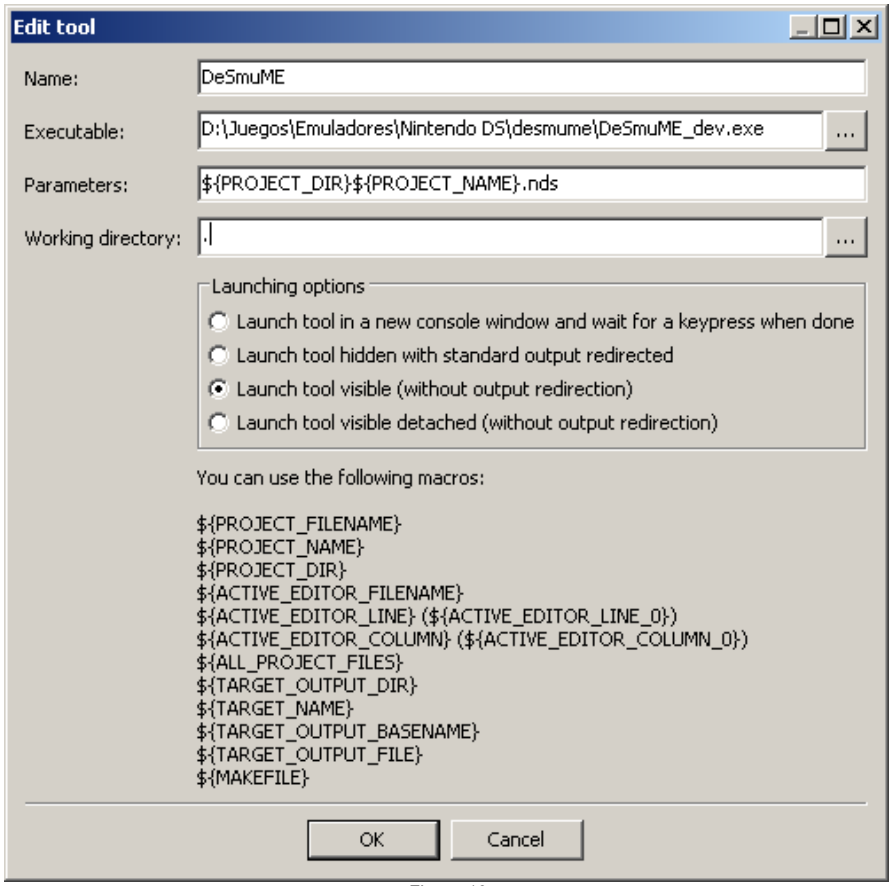


Figura 16

Fuentes y enlaces de interés sobre esta primera parte:

[1] OSDL - A guide to homebrew development for the Nintendo DS:
http://osdl.sourceforge.net/main/documentation/misc/nintendo-DS/homebrew-guide/HomebrewForDS.html
Apartado de *Hardware Resources*

[2] Wikipedia
http://www.wikipedia.org/
Artículos sobre NDS y NDSi.

[3] EOL Wiki
http://www.elotrolado.net/wiki/Portada
Artículos sobre flashcarts.

[4] devkitARM Getting started
http://devkitpro.org/wiki/Getting_Started/devkitARM

[5] devkitPro updater (para Windows)
http://sourceforge.net/projects/devkitpro/files/Automated%20Installer/

[6] Descargas devkitPro para instalación manual
devkitArm - http://sourceforge.net/projects/devkitpro/files/devkitARM/
libnds - http://sourceforge.net/projects/devkitpro/files/libnds/
libfat - http://sourceforge.net/projects/devkitpro/files/libfat/
dswifi - http://sourceforge.net/projects/devkitpro/files/dswifi/
maxmod - http://sourceforge.net/projects/devkitpro/files/maxmod/
libfilesystem - http://sourceforge.net/projects/devkitpro/files/filesystem/
default arm7 - http://sourceforge.net/projects/devkitpro/files/default%20arm7/
ejemplos - http://sourceforge.net/projects/devkitpro/files/examples/nds/

[7] PAlib
http://palib-dev.com/

[8] Eclipse - NDS Manager Builder
http://snipah.com/

[9] DeSmuMe
http://sourceforge.net/projects/desmume/files/

[10] Code::Blocks
http://www.codeblocks.org/

[11] Antigua Wiki de PAlib: Utiliser Code::Blocks comme IDE
http://www.palib.info/wikifr/doku.php?id=day1#utiliser_codeblocks_comme_ide

2 PROGRAMACIÓN BÁSICA EN NDS

En este apartado se abordarán los elementos básicos que devkitPro y PAlib presentan para poder realizar aplicaciones en NDS. Se tratarán salida de texto, entrada de usuario (botones y pantalla táctil), imágenes y salida de audio, con ejemplos para cada subapartado.

La estructura básica de todas las aplicaciones realizadas para NDS tendrán el siguiente formato:

```
#include [...] // Las librerías a utilizar.

int main( void )
{
	inicialización; // Se inicializá PAlib
			// o libNDS.
	while(1){
		// Bucle principal donde se
		// se realizarán las operaciones
		// de la aplicación.
	}

	return 0;
}
```

El motivo de colocar un bucle infinito sirve para simular el comportamiento de un videojuego al entrar en su bucle principal, si no la aplicación se ejecutaría sin permitirnos si quiera ver el resultado. El formato habitual de estos tipos de bucles es el siguiente:

Comienza el bucle:
- Comprobar entrada de usuario.
- Actualizar la lógica interna.
- Combrobar criterio de finalización del bucle.
- Redibujar.

Fin del bucle.

1. SALIDA DE TEXTO

1.1. Texto en libNDS

Hello World

El primer ejemplo que veremos será imprimir una única linea de texto en la pantalla de NDS. En este ejemplo, al ser particularmente pequeño, se mostrará y comentará todo el código (hello01.c):

```
#include <nds.h>
#include <stdio.h>

int main( void )
{
	consoleDemoInit();

	/* Print out the message */
	iprintf("Hello world");
	while(1){
	// Main Loop: Do nothing.
	}

	return 0;
}
```

Al ejecutar el binario resultante en una NDS o en un emulador, veremos simplemente la linea "Hello world" impresa en la esquina superior izquierda de la pantalla inferior.

El código, parte a parte:

consoleDemoInit()

Inicializa una consola de texto predeterminada, sin permitir elegir pantalla o capa donde imprimir el texto.

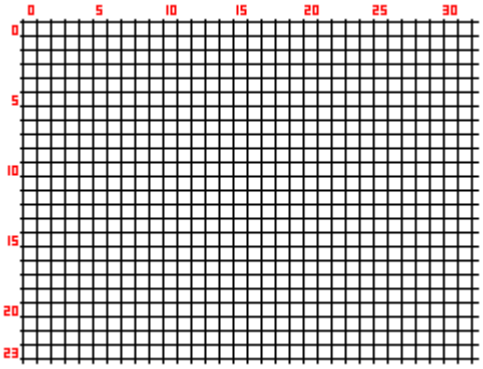
iprintf("Hello world");

Esta función es equivalente al *iprintf()* de compatibles. Imprime texto con formato, soportando solo números enteros.

while(1) { }

Cambiar la posición del texto en la consola:

Para elegir la posición del texto en la pantalla, se usará la secuencia de escape *\x1b[<linea>;<columna>H* usando como valor para la linea un entero entre 0 y 23, y el valor de la columna, un entero entre 0 y 31.



Cuadrícula de la disposición de los carácteres en la consola usando la secuencia de escape \x1b[x;yH, que coincide con la división en tiles de la pantalla.

Con este método, si queremos que la línea "Hello world" salga aproximadamente centrada, modificaremos la linea del *iprintf* de la siguiente manera:

```
iprintf("\x1b[12;10HHello world");
```

Usar una consola distinta a la por defecto:

Para crear una consola con los parámetros deseados, se usará la orden *consoleInit*(...). La función tiene ocho parámetros de entrada, como se puede ver en el ejemplo *Hello02.c* donde se crea una consola para escribir en la pantalla superior:

```
PrintConsole theScreen;
videoSetMode(MODE_0_2D);

consoleInit(&theScreen, // La consola a inicializar.
       3, // Capa del fondo en la que se imprimirá.
       BgType_Text4bpp, // Tipo de fondo.
       BgSize_T_256x256, // Tamaño del fondo.
       31, // Base del mapa.
       0, // Base del tile gráfico.
       true, // Sistema gráfico a usar (main system).
       true); // No cargar gráficos para la fuente.

iprintf("\x1b[12;10HHello world");
```

PrintConsole es el tipo que define las consolas a utilizar a la hora de imprimir contenidos en pantalla.

VideoSetMode(...) se encarga de inicializar el sistema gráfico principal (main system), que es el que se usa en el ejemplo. Si quisieramos imprimir en la pantalla inferior, debería inicializarse con *VideoSetModeSub*(...).

Los modos de video soportados dependen del sistema y el fondo que se estén utilizando, como muestra la siguientes tabla:

Modo	Background 0	Background 1	Background 2	Background 3
0	Texto	Texto	Texto	Texto
1	Texto	Texto	Texto	Rotación
2	Texto	Texto	Rotación	Rotación
3	Texto	Texto	Texto	Rot. Extendida
4	Texto	Texto	Rotación	Rot. Extendida
5	Texto	Texto	Rot. Extendida	Rot. Extendida

Habiendo además un modo extra para el sistema principal (modo 6), para fondos con mapas de bits grandes (en el fondo 1).
Los parámetros de *ConsoleInit*(...) son:

· (PrintConsole *) Un puntero a la consola a inicializar.

· (int) La capa del fondo a utilizar (vease la tabla de modos de video).

· (BgType) El tipo de fondo a utilizar, de entre los siguientes:

BgType_Text8bpp	Fondo por *tiles*, 8 bits por pixel e índices de los *tiles* de 16 bits. No permite rotación ni traslación.
BgType_Text4bpp	Fondo por *tiles*, 4 bits por pixel e índices de los *tiles* de 16 bits. No permite rotación ni traslación.
BgType_Rotation	Fondo por *tiles* con índices de los *tiles* de 8 bits. Permite rotar y escalar.
BgType_ExRotation	Fondo por *tiles* con índices de los *tiles* de 16 bits. Permite rotar y escalar.
BgType_Bmp8	Fondo de mapa de bits. Valores de 8 bits para los colores y paleta de 256 colores.
BgType_Bmp16	Fondo de mapa de bits. Valores de 16 bits para los colores. 5 bits para cada valor RGB y un bit para transparencia.

· (BgSize) El tamaño del fondo. Las variables aceptadas tienen el formato definido en console.h como sigue: BgSize_<tipo>_<resolución>, siendo los tipos posibles T para texto, R para fondo que se puede rotar y escalar, ER para rotación extendida, B8 para mapas de bits de 8bpp y B16 para mapas de bits de 16bpp; y los tipos de resolución admitidos varían desde 128x128 hasta 1024x1024 (se pueden ver todas las resoluciones en la documentación de libnds).

· (int) La base del mapa.

· (int) La base del *tile* gráfico.

· (bool) El sistema gráfico se va a usar: con el valor *true*, se utilizará el sistema principal (la pantalla superior), mientras que con el valor *false*, se imprimirá en la pantalla inferior.

· (bool) Con valor *true* los gráficos de fuente por defecto serán cargados en la capa.

Usar más de una consola:

Para usar más de una consola, se deben definir e inicializar como en el ejemplo anterior y luego usar la función *consoleSelect*(...) para indicar qué consola vamos a usar. Ejemplo *dosConsolas.c*:

```
    PrintConsole mainScreen, subScreen;
    videoSetMode(MODE_0_2D);
    videoSetModeSub(MODE_0_2D);

    consoleInit(&mainScreen, 3, BgType_Text4bpp,
            BgSize_T_256x256, 31, 0, true, true);

    consoleInit(&subScreen, 3, BgType_Text4bpp,
            BgSize_T_256x256, 31, 0, false, true);

    consoleSelect(&mainScreen);
    iprintf("Esta es la consola superior.");

    consoleSelect(&subScreen);
    iprintf("Esta es la consola inferior.");
```

1.2 Texto en Palib

Usando PAlib, con la función más básica para imprimir texto que nos ofrece ya se tiene la opción de elegir la posición de éste en la pantalla (ejemplo *helloPAlib.c*):

```
#include<PA9.h>

int main(){

      // Inicialización de PAlib.
      PA_Init();

      // Se carga la fuente por defecto.
      PA_LoadDefaultText(0, 2); // Pantalla inferior.
      PA_LoadDefaultText(1,2); // Pantalla superior

      // Y se imprime la línea.
      PA_OutputText(0, 16, 3, "World!");
      PA_OutputText(1, 10, 21, "Hello");

      while(true){
            // Main loop: do nothing.
      }
}
```

El código, parte a parte:

#include<PA9.h>

La librería *PA9.h* ya realiza una llamada a las librerías de *libNDS*, por ello, solo incluimos la librería de PAlib al principio del código.

PA_init();

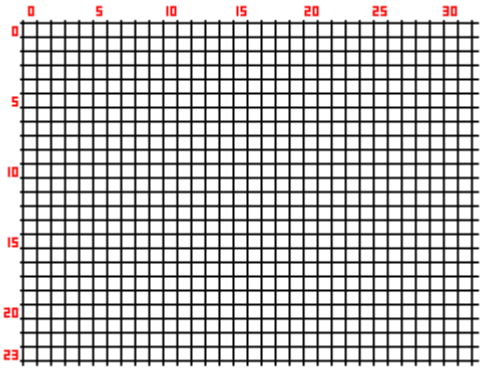
Esta función inicializa intérnamente PAlib y será la primera función a llamar en toda aplicación que haga uso de la librería PAlib.

PA_LoadDefaultText(u8 screen, int background);

Con esta función, se carga en el fondo *background* (valores entre 0 y 3) de la pantalla *screen* (por defecto, con valor 0 se indica la pantalla inferior y 1 la superior) los bitmaps de la fuente por defecto con la que escribir el texto.

*PA_OutputText (u8 screen, u16 x, u16 y, const char * text);*

Ésta es la función encargada de imprimir el texto. El primer argumento indica la pantalla como en la función anterior. Con *x* e *y* indicamos la posición del texto, en tiles. Y el último argumento es el texto a imprimir, aceptando además argumentos extra al estilo *printf()*.



Posición en tiles de cada pantalla.

Otras funciones interesantes a la hora de imprimir texto son:

PA_SetTextTileCol (u8 screen, u8 color);

Permite cambiar el color de la fuente, el primer argumento indica la fuente de qué pantalla y el segundo el color. Están definidas macros TEXT_WHITE, TEXT_RED, etc...

*PA_OutputSimpleText (u8 screen, u16 x, u16 y, const char * text);*

Funciona igual que *PA_OutputText(...)*, pero no permite variables durante el texto. A cambio, imprime texto de forma más eficiente.

*PA_BoxText(u8 screen, u16 x, u16 y, u16 maxx, u16 maxy, const char *text, u32 limit)*

Imprime el texto en una caja limitada por x,y, maxx y maxy, limitando el número de carácteres que se imprimen con el argumento limit. Útil para crear cajas de texto donde éste se muestra carácter a carácter, como en el ejemplo *boxText.c*.

```
int nletters = 0; // Número de letras a escribir.
int letter = 0; // Letra actual.

while (1){ // Bucle principal.

  if (letter == nletters){
      ++nletters; // En cada pasada, escribimos una letra más.
      letter = PA_BoxText(1, 4, 2, 27, 15, "El veloz[...]", nletters);
  }

  PA_WaitForVBL(); // Esperamos el final del frame.

}
```

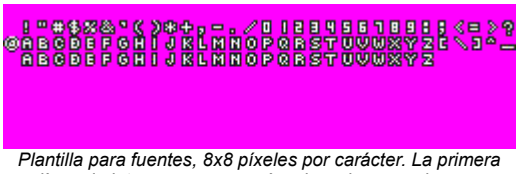
PA_WaitForVBL();

Si bien esta función no tiene relación directa con la salida de texto, ya que ha aparecido, la describimos. Lo que hace esta función es esperar a que ocurra el VBLANK, es decir, el final de frame, para así limitar las repeticiones del bucle principal a los 60 FPS a los que se mueve la NDS.

Usar una fuente personalizada

Para usar una fuente personalizada, es necesario que la fuente esté en formato de mapa de bits, que habrá además que transformar a un formato RAW con PAGfx.

Al instalar PAlib, en las herramientas también se instala la aplicación *dsFont*, que permite crear el mapa de bits de una fuente que tengas instalada en tu PC. O también se puede crear con cualquier editor de imágenes siguiendo la plantilla siguiente:



Plantilla para fuentes, 8x8 píxeles por carácter. La primera línea de letras es para mayúsculas y la segunda para minúsculas.

Con la imagen ya lista, la convertimos usando PAGfx, importándo la imagen como fondo y tipo de fondo seleccionamos *EasyBG*.

Copiamos los archivos que obtenemos con PAGfx a sus directorios correspondientes del proyecto e incluimos la cabecera en el código.

```
#include "all_gfx.h" // Cabecera generada por PAGfx.
```

Después, se carga la fuente con la función *PA_LoadText(u8 screen, u8 bg, PA_BgStruct* font);* que recibe como argumentos la pantalla en la que cargamos la fuente, el fondo de esa pantalla y la posición de memoria de la estructura de datos creada por PAGfx.

Tras ello, se puede imprimir texto con cualquiera de las funciones vistas para tal efecto y será mostrado con la nueva fuente.

```
PA_LoadText(0, 0, &font); // Cargamos los gráficos de la fuente.
PA_BoxText(0, 4, 2, 27, 15, "El veloz [...]", 113);
```

Nota: como se puede observar en el ejemplo, y a diferencia de usar la fuente por defecto, se usa el alfabeto que ofrece ASCII7, sin acentos ni carácteres no ingleses.

2. ENTRADA DE USUARIO

La consola NDS presenta estos botones como entrada de usuario:

· Una cruceta direccional a la izquierda de la pantalla inferior.
· Cuatro botones (A, B, X, Y) a la derecha de la pantalla inferior.
· Dos botones laterales (L y R) situados detrás.
· Botón de Select y botón de Start, a la derecha de la pantalla inferior.
· La pantalla inferior, táctil.

Además, el cierre de la consola tiene también un sensor para saber si está abierta o cerrada, que a efectos de programación, funciona como otro botón más. Por defecto con PAlib, al cerrar la consola, se suspende la ejecución, pero en el ejemplo de música de Maxmod (en la parte 3), veremos como evitar que la consola se suspenda cuando es cerrada.

2.1 Entrada en libNDS

Botones

La información de los botones es accesible mediante enteros sin signo, con una comprobación bit a bit.

Primero se escaneará el estado actual de los botones con *scanKeys()* en cada paso por el bucle principal. Después podemos obtener la informaciónde los botones en un entero con las siguientes funciones:

keysCurrent()	Se obtiene el estado actual de los botones.
keysDown()	Se obtienen los botones pulsados en ese instante.
keysDownRepeat()	Se obtienen los botones pulsados o repitiendo estado.
keysHeld()	Se obtienen los botones mantenidos.
keysUp()	Se obtienen los botones liberados.

El resultado de esas funciones se compara con una multiplicación bit a bit con el valor del botón del que se quiere saber su estado, estando ya definidos los bits con valores autodefinitorios.

Los cuatro botones de la derecha: *KEY_A, KEY_B, KEY_X, KEY_Y.*

L y R: *KEY_L, KEY_R*.

Select y Start: *KEY_SELECT, KEY_START*.

La cruceta de direcciones: *KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT*.

La pantalla táctil (solo como botón): *KEY_TOUCH*.

La bisagra del cierre de la consola: *KEY_LID*.

En el siguiente ejemplo, *buttons.c*, se hace un scanKeys(), se captura el estado actual de los botones en la variable keys y si el botón está pulsado, se muestra en pantalla el botón en concreto, incluyendo la pantalla táctil:

```
u32 keys;

while(1){ // Main Loop.

	scanKeys();

	keys = keysCurrent(); // Se lee el estado actual de todos los botones.

	if(keys & KEY_UP) // Comprobamos si se ha pulsado arriba en la cruceta.
		iprintf("\x1b[10;5H UP");
	else iprintf("\x1b[10;5H   ");

	if(keys & KEY_DOWN) // Comprobamos si se ha pulsado abajo en la cruceta.
		iprintf("\x1b[12;5HDOWN");
	else iprintf("\x1b[12;5H    ");

	[...]

	if(keys & KEY_START) // Comprobamos si se ha pulsado el botón Start.
		iprintf("\x1b[15;25HSTART");
	else iprintf("\x1b[15;25H     ");

	if(keys & KEY_SELECT) // Comprobamos si se ha pulsado el botón Select.
		iprintf("\x1b[16;25HSELECT");
	else iprintf("\x1b[16;25H      ");

	if(keys & KEY_TOUCH) // Comprobamos si se ha pulsado la pantalla táctil.
		iprintf("\x1b[13;10HTOUCH SCREEN");
	else iprintf("\x1b[13;10H            ");

	swiWaitForVBlank();

}
```

Como se puede ver, se comprueba el estado instantáneo de cada botón, mostrándose en pantalla sólo si esta pulsado y todo el tiempo que está pulsado. Si se quieren realizar acciones en momentos precisos (justo cuando se pulsa un botón, o cuando se libera), se podrán utilizar las otras funciones.

En el ejemplo *heldbutton.c* se usan las funciones k*eysDown(), keysHeld()* y *keysUp()* para realizar acciones específicas durante los estados del botón A.

```
	u32 keys;

	while(1){ // Main Loop.

		scanKeys();

		keys = keysDown();
		if(keys & KEY_A)
			i = 0;

		keys = keysHeld();
		if(keys & KEY_A)
			i++;

		keys = keysUp();
		if(keys & KEY_A){
			if(i < 50)
				iprintf("No suficiente tiempo.\n");
			if(i >= 50)
				iprintf("Pulsado correctamente A.\n");

		}

	swiWaitForVBlank();

	}
```

Pantalla táctil

Para reconocer la posición del puntero en la pantalla táctil basta crear una variable del tipo *touchPosition* y leer el valor de la posición como se muestra en el ejemplo *tactil.c*:

```
touchPosition touchXY;

while(1) {

	touchRead(&touchXY);  // Leemos la posición actual.

	iprintf("\x1b[1;0HPosicion x = %d,%d     \n",
			touchXY.rawx, touchXY.px);
	iprintf("Posicion y = %d,%d     ",
			touchXY.rawy, touchXY.py);

	swiWaitForVBlank();

    }
```

2.2 Entrada en PAlib

Botones

Realizando en PAlib el mismo ejemplo que en libNDS, se pueden comprobar un par de diferencias (ejemplo *buttons.c*):

```
while(1){

	if(Pad.Held.Up) // Comprobamos si se ha pulsado arriba en la cruceta.
		PA_OutputSimpleText(0, 5, 10, "UP");
	else PA_OutputSimpleText(0, 5, 10, "  ");

	// [...]

	if(Pad.Held.Select) // Comprobamos si se ha pulsado el botón Select.
		PA_OutputSimpleText(0, 25, 16, "Select");
	else PA_OutputSimpleText(0, 25, 16, "      ");

	PA_WaitForVBL();

}
```

La primera es que no necesitamos llamar a una función para ver el estado de todos los botones al comenzar el bucle. Internamente, PAlib ya realiza esa función automáticamente cada *frame*.

La segunda, es que no necesitamos realizar una comprobación a bits para saber si una tecla está ya pulsada o no. PAlib ya guarda esa información en la estructura *Pad*.

En esta misma estructura se guardan también las nuevas pulsaciones y los botones que han sido liberados:

Pad.Held.<botón>	Valor por defecto 0. Pasa a 1 mientras el botón está pulsado.
Pad.Newpress.<botón>	Cuando se pulsa el botón, vale 1 durante 1 *frame*.
Pad.Released.<botón>	Cuando se suelta el botón, vale 1 durante 1 *frame*.

Las variables de botones en la estructura *Pad* son:

- *Left, Right, Up, Down* para la cruceta de direcciones.
- *A, B, X, Y* para los botones A, B, X e Y.
- *L, R* para los gatillos izquierdo y derecho.
- *Start, Select* para los botones Start y Select.

Pantalla táctil

En libNDS, si la pantalla táctil estaba pulsada o no se manejaba exáctamente igual que el resto de botones. En PAlib han añadido esa información a la estructura *Stylus*, que contiene toda la información que nos interesa sobre la pantalla táctil:

Stylus.Held	Valor 1 mientras la pantalla táctil esté presionada.
Stylus.Newpress	Cuando se pulsa la pantalla táctil, vale 1 durante 1 *frame*.
Stylus.Released	Cuando se deja de pulsar la pantalla táctil, vale 1 durante 1 *frame*.
Stylus.X	La posición del lápiz en el eje X en la pantalla táctil.
Stylus.Y	La posición del lápiz en el eje Y en la pantalla táctil.

Se puede ver el uso de esta estructura en el ejemplo *tactil.c*, donde si la pantalla táctil está pulsada se muestran las coordenadas en texto verde y si no está pulsada, sale un aviso en texto rojo:

```
if(Stylus.Held){
	PA_SetTextTileCol(1, TEXT_GREEN);
	PA_OutputText(1, 2, 2,
		"Posicion del Stylus: %d, %d ", Stylus.X, Stylus.Y);
}

else{
	PA_SetTextTileCol(1, TEXT_RED);
	PA_OutputSimpleText(1, 2, 2,
		"Pantalla táctil no pulsada   ");
}
```

Teclado táctil

PAlib incluye además un teclado prefabricado para facilitar la entrada de usuario. El teclado se carga en la pantalla táctil y reconoce qué teclas son pulsadas con el *stylus*.

En el ejemplo *keyboard.c* se puede ver cómo cargarlo y cómo leer su entrada.

```
PA_LoadDefaultText(1, 0);
PA_InitKeyboard(2); // Cargamos el teclado.
PA_KeyboardIn(20, 100); // Coloca el teclado en posición.

int nletter = 0; // Contador para los carácteres.
char text[200];

while(1){

	text[nletter] = PA_CheckKeyboard();

	// Si hay nueva letra, la añadimos al texto.
	if (text[nletter] > 31 || text[nletter] == '\n')
		nletter++;

	// Si se pulsa backspace, borramos la última letra.
	else if ((text[nletter] == PA_BACKSPACE)&&nletter) {
		nletter--;
		text[nletter] = ' ';
	}

	// Imprimimos el texto obtenido.
	PA_OutputSimpleText(1, 0, 0, text);

	PA_WaitForVBL();

}
```

El argumento de *PA_InitKeyboard(u8 background);* sirve para indicar en qué fondo de la pantalla táctil cargamos el teclado.

PA_CheckKeyboard(); devuelve el carácter pulsado en el teclado.

PA_KeyBoardIn(int X, int, Y); nos permite definir la posición final del teclado, con un scroll hasta alcanzarla.

3. IMÁGENES

Como videoconsola que es, la NDS está especialmente preparada para mostrar gráficos en sus pantallas. Si bien sus 4MB de memoria principal y 656KB de memoria gráfica no permite mostrar gráficos de última generación, su arquitectura facilita el acceso a la memoria gráfica sin pasar por la CPU, con lo que realiza el dibujado de forma eficiente.

Dada la dificultad que requiere la gestión de memoria para dibujar gráficos en libNDS, en este apartado solo trabajaremos a más alto nivel, con la librería PAlib.

Dado que los gráficos se sitúan en pantalla, hay que tener en cuenta las dimensiones de éstas:

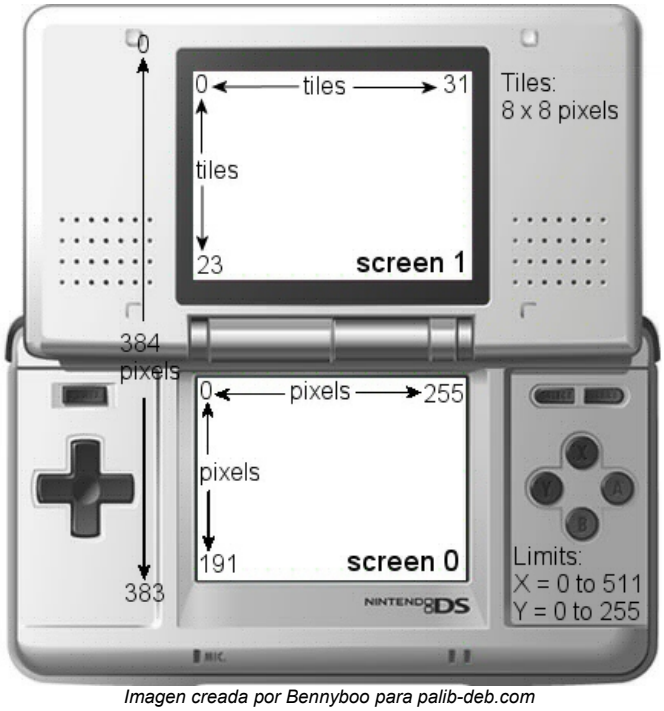


Imagen creada por Bennyboo para palib-deb.com

Las pantallas tienen 768 *tiles* (32 ancho x 24 alto) y una resolución de 256x191 píxeles, pero los límites llegan hasta 511 para X y 255 para Y. Poner una imágen en X = 512 equivaldría a ponerla en X = 0.

Hay dos tipos de imágenes 2D en NDS: fondos y sprites. La consola puede mostrar hasta 4 fondos o 128 sprites por motor gráfico.

3.1 Fondos

Los fondos en PAlib tienen definida una estructura *PA_BgStruct*, que PAGfx rellena automáticamente con los datos de nuestras imágenes. Entre los datos de la estructura, se encuentran el tipo de fondo que es, el tamaño, los *tiles*, el tamaño de los *tiles*, el mapa, el tamaño del mapa y la paleta.

Cargar fondos

Cargar fondos es fácil y rápido con PAlib. Primero convertimos las imágenes a RAW usando PAGfx. El modo de fondo seleccionado será EasyBg, que convierte el fondo en *tiles*, mapa y paleta.

En el ejemplo *loadingbg* vemos cómo cargar fondos en ambas pantallas:

```
PA_LoadBackground(1, 3, &fondo1);
PA_LoadBackground(0, 3, &fondo2);
```

Dando como resultado:

Los argumentos de *PA_LoadBackground(u8 screen, u8 bg, PA_BgStruct* img);* son, primero la pantalla en la que cargamos el fondo (con 1 pantalla superior, con 0 pantalla inferior), el fondo de la pantalla donde va nuestra imagen (entre 0 y 3) y por último, un puntero a la estructura del fondo, con sus campos rellenados por PAGfx. La estructura tendrá el mismo nombre que la imagen usada, sin extensión.

Trasladar fondos (*scrolling*)

En el ejemplo *scrollbg* vemos cómo es posible añadir traslación a los fondos cargados en cada pantalla. Se puede observar el efecto de *wrapping* que realiza la consola al llegar al límite de cada fondo. Esto se debe a que la función elige el tipo de scroll dependiendo del tamaño del fondo, en nuestro caso, 256*256.

En este ejemplo, además, se carga contenido en dos fondos de la pantalla (texto en el fondo 0 y la imagen en el fondo 3), con lo que se puede comprobar cómo el fondo 0 se dibuja delante del fondo 3 por defecto.

```
// Inicializamos el texto.
PA_InitText(1, 0);
PA_InitText(0, 0);

// Cargamos los fondos.
PA_LoadBackground(1, 3, &fondo1);
PA_LoadBackground(0, 3, &fondo2);

// Variables para el scroll.
int scrolltopx = 0;
int scrolltopy = 0;

int scrollsubx = 0;
int scrollsuby = 0;

while (1)
      {
    // Actualizamos las variables según las teclas pulsadas.
    scrolltopx += (Pad.Held.Left - Pad.Held.Right) * 4;
    scrolltopy += (Pad.Held.Up - Pad.Held.Down) * 4;

    scrollsubx += (Pad.Held.Y - Pad.Held.A) * 4;
    scrollsuby += (Pad.Held.X - Pad.Held.B) * 4;

    // Movemos los fondos.
    PA_EasyBgScrollXY(1, 3, scrolltopx, scrolltopy);
    PA_EasyBgScrollXY(0, 3, scrollsubx, scrollsuby);

    // Imprimimos la posición en texto.
    PA_OutputText(1, 0, 0, "x : %d   \ny : %d   ", scrolltopx, scrolltopy);
    PA_OutputText(0, 0, 0, "x : %d   \ny : %d   ", scrollsubx, scrollsuby);

    PA_WaitForVBL();
}
```

La función que permite el scroll es *PA_EasyBgScrollXY(u8 screen, u8 bg, u32 x u32 y)*. Los dos primeros argumentos son la pantalla y el fondo tal y como se viene viendo en todos los ejemplos y los dos últimos argumentos son las coordenadas donde colocar el fondo en píxeles.

Como, por el tamaño del fondo, hay *wrapping*, si la posición se sale del tamaño de la pantalla el fondo será colocado inmediatamente al final de su propio borde, repitiéndose indefinidamente. Si la imagen fuese más pequeña que la pantalla, rellenaría hasta el borde con píxeles negros y después del vacío, la imagen se repetiría otra vez.

Sobre las operación para calcular el scroll:

```
scrolltopx += (Pad.Held.Left - Pad.Held.Right) * 4;
scrolltopy += (Pad.Held.Up - Pad.Held.Down) * 4;

scrollsubx += (Pad.Held.Y - Pad.Held.A) * 4;
scrollsuby += (Pad.Held.X - Pad.Held.B) * 4;
```

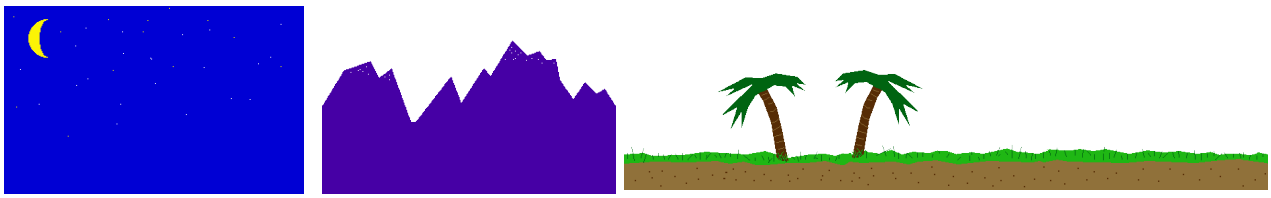
El 4 del final es la velocidad con la que se moverá el fondo mientras se mantenga pulsado el botón, siendo esta velocidad en píxeles por frame (y recordemos que la NDS imprime imágenes a 60 frames por segundo).

El pulsar un botón o su opuesto dara valor negativo o positivo a la velocidad, ya que al pulsarlo, tomará valor 1 y, en el caso de la cruceta, no se puede pulsar simultáneamente el contrario. En el caso de la pantalla inferior, sí que se podría, pero ambos valores se anulan, dando como resultado 0 x 4 = 0.

Trasladar varios fondos con paralaje

El *parallax scrolling* es una técnica que, haciendo que los distintos fondos se muevan a distintas velocidades, consigue un efecto de profundidad. Como es un método muy utilizado en videojuegos, PAlib tiene implementadas funciones para facilitar la tarea de realizar paralaje entre fondos.

Para ello, cargaremos tres fondos distintos:



bg3.bmp *bg2.bmp* *bg1.bmp*

Imágenes obtenidas en http://en.wikipedia.org/wiki/Parallax_scrolling

El código de este ejemplo:

```
// Cargamos los fondos.
PA_LoadBackground(0, 1, &bg1);
PA_LoadBackground(0, 2, &bg2);
PA_LoadBackground(0, 3, &bg3);

// Inicializamos la paralaje.
PA_InitParallaxX(0, 0, 256, 128, 0);

int scroll = 0;

while (true)
{
      scroll += (Pad.Held.Right - Pad.Held.Left) * 4;

      // Se realiza el scroll con paralaje.
      PA_ParallaxScrollX(0, scroll);
      PA_WaitForVBL();
}
```

Como se vio en el ejemplo del desplazamiento normal, el orden de los fondos importa, siendo el fondo 0 el que se muestra más al frente y el 3 el que más atrás, por defecto. En el ejemplo, el nombre de las imágenes está puesto para coincidir con el fondo que van a ocupar, así la imagen bg1.bmp irá en el fondo 1.

Las funciones para inicializar el paralaje son:

PA_InitParallaxX(u8 screen, int bg0, int bg1, int bg2, int bg3); para el eje X y
PA_InitParallaxY(u8 screen, int bg0, int bg1, int bg2, int bg3); para el eje Y.

Sus argumentos son, primero la pantalla y luego la velocidad para cada uno de los fondos de dicha pantalla. Un valor de 256 indica velocidad normal, de tal forma que valores menores supondrán velocidades menores (por ejemplo, 128 para que el fondo use la mitad de la velocidad normal) y valores mayores indicarán velocidades mayores (ejemplo, 512 para el doble). Con un valor de cero, el fondo no se ve afectado por la paralaje.

Al llamar a la función *PA_ParallaxScrollX(u8 screen, int scroll)*; se desplazarán en el eje X todos los fondos con velocidades definidas distintas a cero de la pantalla seleccionada, aplicando la variación de velocidad a cada fondo.

Mapas de bits como fondos

Hasta ahora hemos cargado fondos como mapas de *tiles*, que es el uso habitual de los fondos, pero a veces es necesario cargar directamente una imagen en formato de mapa de bits.

El dibujar un mapa de bits en lugar de un mapa de *tiles* presenta ciertas desventajas:

· El consumo de memoria gráfica es muy elevado. Un mapa de bits de 8bpp necesita un tercio de la VRAM, mientras que uno de 16bpp necesita dos tercios.

· Al dibujar bit a bit en lugar de *tile* a *tile* (un *tile* son 8x8 bits), consume 64 veces más tiempo realizar el dibujado.

· Los mapas de bits sólo pueden colocarse en el fondo 3 de cada motor gráfico.

La principal ventaja de los fondos de 16bpp es que no necesitan paleta, pues usan todos los colores disponibles en NDS.

El primer paso para cargar el fondo será convertir las imágenes con PAGfx, eligiendo como tipo de fondo '8bit' para las imágenes de 8bpp y '16bit' para las de 16bpp.

En el ejemplo *bmpbgs* podemos ver los pasos para cargar imágenes, tanto de 8bpp:

```
// Inicializamos el fondo para cargar imágenes de 8bpp.
PA_Init8bitBg(0,3);

// Cargamos la imagen de 8bpp en la pantalla inferior.
PA_Load8bitBgPal(0,(void*)fondo8bpp_Pal);
PA_Load8bitBitmap(0, fondo8bpp_Bitmap);
```

Como de 16bpp:

```
// Inicializamos el fondo para cargar imágenes de 16bpp.
PA_Init16bitBg(1,3);

// Cargamos la imagen de 16bpp en la pantalla superior.
PA_Load16bitBitmap(1, fondo16bpp_Bitmap);
```

En ambos casos primero se inicializa el fondo 3 de la pantalla correspondiente para cargar mapas de bits y después ya metemos la imagen en sí en memoria. En el caso de la imagen de 8bpp, además, se debe cargar la paleta.

Fondos para dibujar: Primitivas 2D

Los fondos para cargar mapas de bits de PAlib permiten también dibujar primitivas 2D sobre ellos. El fondo lo inicializamos como en el apartado anterior, bien en 8 bits, bien en 16 bits. Las principales primitivas permitidas y las funciones para dibujarlas son las siguientes:

PA_Put16bitPixel (screen, x, y, color)	Dibuja un píxel en un fondo de 16 bits.
PA_Draw16bitLine (screen, x1, y1, x2, y2, color)	Dibuja una línea en un fondo de 16 bits.
PA_Draw16bitRect (screen, x1, y1, x1, y1, color)	Dibuja un rectángulo relleno entre los dos puntos lados en un fondo de 16 bits.
PA_Put8bitPixel (screen, x, y, color)	Dibuja un píxel en un fondo de 8 bits.
PA_Draw8bitLine (screen, x1, y1, x2, y2, color)	Dibuja una línea en un fondo de 8 bits.

En el ejemplo *2Dbasics* dibujamos un cuadrado relleno usando *PA_Draw16bitRect*:

```
PA_Draw16bitRect(0, x, y, x + 16, y + 16, PA_RGB(r, g, b));
```

Para definir el color, PAlib tiene creada la macro que se ve en el ejemplo (*PA_RGB*), que permite definir el color usando valores independientes para rojo, verde y azul, entre 0 y 31.

Para dibujar otro tipo de figuras aparte de rectángulos rellenos, debemos usar las funciones para dibujar líneas o píxeles, como el rectángulo en el que rebota el anterior cuadrado del ejemplo, que está creado con cuatro líneas independientes:

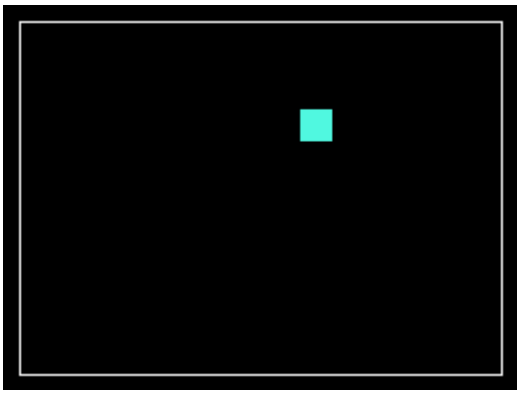
```
PA_Draw16bitLine(0, 8, 8, 8, 184, PA_RGB(31,31,31));
PA_Draw16bitLine(0, 8, 8, 248, 8, PA_RGB(31,31,31));
PA_Draw16bitLine(0, 248, 8, 248, 184, PA_RGB(31,31,31));
PA_Draw16bitLine(0, 8, 184, 248, 184, PA_RGB(31,31,31));
```

Conociendo la posición de los vértices, se puede generar cualquier figura geométrica.

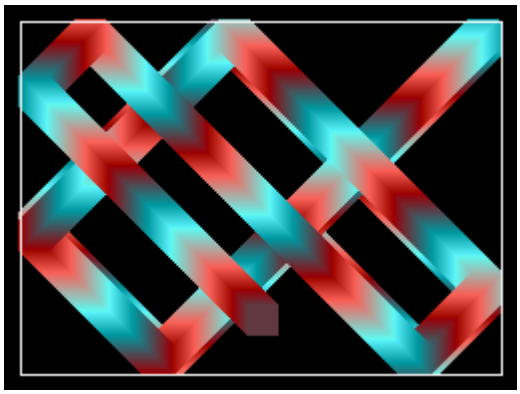
Otra función interesante del mismo ejemplo es:

```
PA_Clear16bitBg(0);
```

Esta macro lo que hace es limpiar la memoria gráfica, borrando el contenido de pantalla. Para crear efecto de animación es necesario que esté, o los nuevos fotogramas se dibujarán directamente encima de los anteriores. El resultado de limpiar o no el fotrograma anterior de la pantalla se puede apreciar en las siguientes imágenes:



*Ejemplo **con** PA_Clear16bitBg(0)*



*Ejemplo **sin** PA_Clear16bitBg(0)*

Fondos para dibujar: Stylus

Con lo visto en el apartado anterior, podríamos dibujar ya usando el Stylus, leyendo la posición en cada momento y coloreando un píxel con esas coordenadas. Pero PAlib tiene ya las funciones *PA_8bitDraw* y PA_16bitDraw que se encargan de hacer eso en fondos de 8 y 16 bits respectivamente.

Fragmento del ejemplo *DSPaint*:

```
PA_SetDrawSize(0, pincel);
PA_16bitDraw (0, rgb);
```

Los argumentos de *PA_16bitDraw* son, primero la pantalla en la que vamos a dibujar, que al usar el stylus debe ser siempre la inferior y el segundo argumento, el color de los píxeles.

La otra función, *PA_SetDrawSize*, sirve para definir el tamaño del pincel a utilizar.

3.2 Sprites

Los sprites son imágenes 2D, habitualmente pequeñas y parcialmente transparentes utilizados en programación gráfica (y, especialmente, videojuegos) para representar prácticamente la totalidad de objetos que hay en pantalla.

Los sprites pueden trasladarse por la pantalla, ser rotados, escalados o incluso animados actualizando su imagen a mostrar. En el caso particular de la NDS, hay 32 'rotsets' disponibles, eso quiere decir que aunque se pueden mostrar un total de 256 sprites (128 por pantalla), éstos sólo pueden rotarse o escalarse de 32 maneras distintas simultáneamente, por lo que hay que agruparlos por comportamiento.

En NDS, los sprites presentan 3 posibles modos de color:

· El modo de Gameboy Advance: 16 paletas de 16 colores cada una, pudiendo un sprite usar distintas paletas para crear sprites de distinta apariencia.

· 16 paletas de 256 colores, el modo recomendado por la relación memoria-calidad.

· Sprites con gráficos en 16bpp, que consumen muchos recursos.

Así mismo, también están limitadas las posibles dimensiones de los sprites tal y como muestra la siguiente tabla:

8x8	16x8	32x8	
8x16	16x16	32x16	
8x32	16x32	32x32	64x32
		32x64	64x64

De tal forma que si tuviéramos un sprite de, por ejemplo, 42x42, deberíamos usar el tamaño de sprite 64x64 y usar el color de transparencia para rellenar el espacio sobrante.

Eso no quiere decir que la imagen deba tener una de esas dimensiones. Si estamos trabajando con sprites animados, la imagen del sprite tendrá como altura el número de frames multiplicado por uno de los valores posibles (8,16, 32 o 64).

Carga de sprites

Tras transformar los sprites con PAGfx, lo primero que realizaremos con respecto a ellos en el código es cargar la paleta:

```
PA_LoadSpritePal(0, 0, (void*)sprite_nave_Pal);
```

El primer argumento, como de costumbre, es la pantalla en cuya memoria vamos a cargar la paleta. Con un valor de 0 se carga en la pantalla inferior y con 1 en la superior.

El segundo argumento indica qué paleta estamos cargando, de las 16 que podemos cargar en cada pantalla (valores de 0 a 15). Será el número que usemos para identificar la paleta al cargar los sprites.

Por último se envía el puntero a los datos de la paleta generados por PAGfx.

Si no se cargase la paleta, los sprites que la usen resultarían invisibles o, en caso de haber cargado previamente otra paleta, con colores erróneos.

A continuación, cargaremos los propios sprites:

```
PA_CreateSprite(0, 0, (void*)nave_Sprite, OBJ_SIZE_16X16, 1, 0, x, y);
```

Otra vez tenemos la pantalla como primer argumento. El siguiente valor identifica al sprite entre los 128 que puede contener dicha pantalla (acepta valores entre 0 y 127). Tras ello, se cargan los datos del sprite generados por PAGfx.

El argumento del tamaño es una macro que encapsula las dos dimensiones del sprite, pudiendo ser los valores vistos en la tabla de la página anterior.

Tras el tamaño, el siguiente argumento es un booleano para indicar la profundidad de color. Con 0 se indica que el sprite es de 16 colores y con 1, que es de 256.

Luego cargamos la paleta que usa este sprite con el identificador que le pusimos a la paleta al cargarla en memoria.

Y para terminar, con los dos últimos argumentos indicamos la posición en pantalla del sprite. Sus valores van de 0 a 511 para el eje X y de 0 a 255 para el eje Y, siendo visibles únicamente los valores que caigan dentro de 256x192.

Transformar sprites

Traslación

Para cambiar de posición el sprite, basta con usar la función:

```
PA_SetSpriteXY(0, 0, x, y);
```

Siendo el primer argumento la pantalla; el segundo, el sprite a mover y los dos últimos indican la nueva localización.

Si estamos moviendo el sprite de forma fluida, la consola realiza *wrapping* al llegar a los límites, pero estos límites no coinciden por abajo y por la derecha con los bordes de la pantalla como hemos dicho antes, si no que se va en X hasta la posición 511 y en Y hasta la posición 255. Por lo tanto, si queremos que el sprite no desaparezca un buen rato detrás de la pantalla, deberemos corregir las coordenadas a mano.

En el ejemplo *spritebasics* limitamos el movimiento de un sprite de tamaño 16x16 al interior de la pantalla, parándose al llegar al límite. Hay que recordar que el origen de coordenadas local del sprite está situado en su esquina superior izquierda.

```
if(x < 0) x = 0;
if(x > 256 - 16) x = 256 - 16;
if(y < 0) y = 0;
if(y > 192 - 16) y = 192 - 16;
```

Si, por otro lado, quisieramos que el sprite realizase un wrapping con respecto a los bordes de la pantalla, basta jugar con los valores para que se comporte como queramos:

```
if(x < 0 - 16) x = 255;
if(x > 256) x = 0 - 16;
if(y < 0 -16) y = 191;
if(y > 192) y = 0 - 16;
```

Nota: también están disponibles las funciones *PA_SetSpriteX(...)* y *PA_SetSpriteY(...)*, en las que sólo se modifica el valor en la coordenada dada por el nombre de la función.

Trasladar con el Stylus

PAlib nos ofrece ya una función que mueve el sprite que el stylus está tocando.

```
PA_MoveSprite(i);
```

Siendo i el sprite a mover. La función comprueba si el sprite i está siendo tocado por el Stylus y lo mueve centrado con respecto a su posición, pero no tiene en cuenta transparencias.

Por ello, también tenemos disponibles otras dos funciones: *PA_SpriteTouched(sprite)* y *PA_SpriteTouchedPix(sprite)*. En el siguiente fragmento de código se ve lo que tenemos que hacer para conseguir el mismo efecto con estas funciones que con la anterior:

```
if(Stylus.Held && PA_SpriteTouched(0))
      PA_SetSpriteXY(0, 0, Stylus.X - 16/2, Stylus.Y - 16/2);
```

Al igual que antes, los 16 que se ven hacen referencia al tamaño del sprite del ejemplo, que es de 16x16.

El comprobar si está pulsada la pantalla táctil es recomendable para evitar comportamientos indeseados, ya que *PA_SpriteTouched(sprite)* no comprueba eso.

Rotación

Lo primero que debemos hacer tanto para rotar como para escalar un sprite es añadirlo a un *rotset* de los 32 que NDS tiene disponibles. Cada *rotset* es una agrupación de sprites que se rotarán y escalarán de forma idéntica, por lo que aunque puedan mostrarse hasta 256 sprites, solo podrán estar rotados o escalados de 32 maneras distintas.

Para añadir un sprite a un *rotset* hay que indicarlo como se muestra:

```
PA_SetSpriteRotEnable(screen, sprite, rotset);
```

El primer argumento indica otra vez la pantalla. Con el segundo argumento indicamos el sprite con la id asignada en su creación y con el último argumento indicamos entre 0 y 31 el *rotset* al que estamos asignando el sprite.

Una vez añadido el sprite a un *rotset* ya podemos rotarlo. En el ejemplo rotamos el sprite con los botones L y R y mantenemos el ángulo con un valor entre 0 y 511.

```
u16 angle = 0;

while (1){

      angle += Pad.Held.L - Pad.Held.R;
      angle &= 511;

      PA_SetRotsetNoZoom(0,0,angle);
}
```

Los argumentos indican, en orden, la pantalla, el *rotset* al que aplicamos el cambio de ángulo y el ángulo que rotamos todos los elementos de ese *rotset*.

Escalado

Como en la rotación, el sprite debe estar asignado a un *rotset*.

```
PA_SetRotsetNoAngle(0, 0, zoomx, zoomy);
```

Los dos primeros parámetros coinciden con la función de rotación. Los siguientes indican el escalado a aplicar a cada eje de forma independiente. Si quisieramos que el objeto se escalase de forma uniforme, habría que usar la misma variable para ambos argumentos.

Los valores para el zoom son de 256 para que no haya ningún tipo de escalado, valores menores para aumentar su tamaño (por ejemplo, 128 haría el sprite el doble de grande) y valores mayores reducirían el tamaño del sprite (512, la mitad del tamaño).

El escalado hace zoom al sprite dentro de su ventana, de tal forma que si hemos definido el sprite de, por ejemplo, 16x16 píxeles, todo lo que se salga del rectángulo formado por esos 16x16 píxeles no sé dibujará.

Para ello, PAlib nos ofrece la función *PA_SetSpriteDblsize(screen, sprite, enable/disable)* que doblará el tamaño de la ventana del sprite definida desde el centro, colocando un marco transparente alrededor de nuestro sprite. Es importante notar que el aumento de la ventana se realiza centrado, porque eso cambia el origen de coordenadas local del sprite más arriba y más a la izquierda.

Nota: Hay una función para realizar simultáneamente la rotación y el escalado. Esta función es *PA_SetRotset(screen, rotset, angle, zoomx, zoomy);* donde indicamos primero el ángulo y después el escalado a los ejes X e Y (después de indicar la pantalla y el *rotset* al que realizamos la modificación).

Sprites animados

A la hora de animar sprites, debemos preparar la imagen que contendrá todos los cuadros de la animación de forma vertical, como se muestra en la imagen siguiente:

Todos los cuadros de la animación deben tener el mismo tamaño (en este ejemplo - *anim1* - es 32x32 píxeles) pues a la hora de cargar el sprite, se debe usar uno de los tamaños permitidos vistos antes. Esto dará como resultado imágenes que de anchura tendrán una de las permitidas por PAlib y de altura, otra de las permitidas (no necesariamente la misma) multiplicada por el número de frames de la animación.

La imagen se convierte y se carga como ya se ha visto en el apartado 'Carga de sprites', y a continuación se comienza el bucle de la animación:

```
// Comenzamos la animacion entre los frames 0 y 6 a 7 fps.
PA_StartSpriteAnim(0, 0, 0, 6, 7);
```

Los dos primeros argumentos son los identificadores de la pantalla y el sprite. Después indicamos los sprites inicial y final de la animación (se empieza a contar por 0) y por último, la velocidad de reproducción del sprite en cuadros por segundo.

Con esto, la animación del sprite se reproducirá en bucle de forma indefinida, si bien podemos pausarlo o pararlo en cualquier momento. En nuestro ejemplo pausamos o volvemos a reproducir cada vez que se pulsa el botón A.

```
// Al pulsar A, la animación se pausa pone de nuevo en marcha.
if(Pad.Released.A){
      if(!pause) pause = 1;
      else pause = 0;
      PA_SpriteAnimPause(0, 0, pause);
}
```

Los argumentos iniciales de la función de pausa vuelven a ser los identificadores de pantalla y sprite. El último argumento pausará la animación si tiene un valor distinto a cero, o la volverá a poner en marcha con cero.

Para parar del todo la animación está la función *PA_StopSpriteAnim(screen, sprite)*. Con esta función no se puede reanudar la animación de forma automática.

Si queremos actualizar el cuadro de forma manual, se deberá usar la siguiente función:

```
PA_SetSpriteAnim(screen, sprite, frame);
```

Para ello, nos puede interesar también saber en qué frame estamos:

```
PA_GetSpriteAnimFrame(screen, sprite);
```

Esta última función nos devuelve un entero sin signo de 16 bits indicándonos el cuadro actual de la animación. En el ejemplo *anim1* vemos el uso de estas dos funciones:

```
// Con R avanzamos un frame.

if(pause && Pad.Released.R)
{
      int frame = PA_GetSpriteAnimFrame(0, 0) + 1;
      if(frame == 7) frame = 0;
      PA_SetSpriteAnim(0, 0, frame);
}

// Con L retrocedemos un frame.

if(pause && Pad.Released.L)
{
      int frame = PA_GetSpriteAnimFrame(0, 0) - 1;
      if(frame == -1) frame = 6;
      PA_SetSpriteAnim(0, 0, frame);
}
```

PAlib no corrige automáticamente si pones valores mayores o menores de los permitidos por el sprite en el frame, por lo que debemos corregirlo a mano para evitar los fallos gráficos que ocurrirían si pusieramos valores fuera de rango (en el ejemplo, mayores a 6 o menores a 0).

Si después de actualizar manualmente un sprite reanudamos la animación de forma automática, continuará desde el nuevo cuadro.

En una misma imagen se pueden incluir varias animaciones, como por ejemplo, las distintas direcciones de un personaje al moverse (ejemplo *anim2*). Ésto se puede realizar porque la función para indicar el principio de la animación nos permite indicar cuáles son los frames iniciales y finales de dicha animación.

```
// Las animaciones en movimiento.
if(Pad.Newpress.Up)
      PA_StartSpriteAnim(0, 0, 0, 3, 6);

if(Pad.Newpress.Down)
      PA_StartSpriteAnim(0, 0, 8, 11, 6);
```

En este ejemplo, la animación del personaje caminando hacia arriba iría entre los cuadros 0 y 3, ambos incluidos; mientras que la animación del personaje yendo hacia abajo, entre los frames 8 y 11, siendo ambas animaciones de 4 frames.

Cuando una animación es simétrica hacia un lado con respecto al otro, se suelen realizar las imágenes de únicamente un lado. En nuestro ejemplo, la animación del personaje yendo hacia la izquierda o hacia la derecha es idéntica, salvo por la dirección en la que mira. Para voltear la imágen simétricamente con respecto al eje Y y así poder usar los mismos cuadros, usamos la siguiente función:

```
PA_SetSpriteHflip(screen, sprite, hflip)
```

Siendo el valor del argumento hflip 1 si queremos voltear o 0 si queremos devolverlo a su orientación original.

Para voltear verticalmente, la función es idéntica:

```
PA_SetSpriteVflip(screen, sprite, vflip)
```

Sprites como botones

Al trasladar sprites con stylus vimos una función que nos permitía saber cuándo se había pulsado sobre un sprite.

```
PA_SpriteTouched(i);
```

Con ayuda de esta función, podemos hacer botones para nuestras aplicaciones a partir de sprites (ejemplo *spritebuttons*). Necesitamos vigilar todos los botones en el código, a la espera de que sean pulsados:

```
if(Stylus.Held && PA_SpriteTouched(0)){
      PA_SetTextTileCol(1, TEXT_RED);
      PA_OutputSimpleText(1, 2, 11, "Has pulsado el boton rojo.");
      PA_SetSpriteAnim(0, 0, 1);
}

if(Stylus.Held && PA_SpriteTouched(1)){
      PA_SetTextTileCol(1, TEXT_GREEN);
      PA_OutputSimpleText(1, 2, 11, "Has pulsado el boton verde.");
      PA_SetSpriteAnim(0, 1, 1);
}
```

Recordemos que la aplicación funciona en un bucle principal, por lo que para vigilar si pulsamos los botones o no, se debe hacer con una sentencia if en lugar de sentencias de bucles para evitar que la aplicación se quede bloqueada esperando una acción del usuario.

En el ejemplo, pulsar el botón muestra un texto en la pantalla superior y cambia el fotograma del sprite a mostrar. Para devolver al fotograma inicial, vigilamos cuándo dejamos de pulsar la pantalla táctil con el stylus.

```
if(Stylus.Released){
      PA_SetSpriteAnim(0, 0, 0);
      PA_SetSpriteAnim(0, 1, 0);
      PA_SetSpriteAnim(0, 2, 0);
}
```

4. SONIDO

En este apartado comentaremos la librería de reproducción de sonido por defecto de PAlib, ésta es, ASlib (Advanced Sound library). ASlib tiene soporte para reproducir efectos de sonido RAW, que es lo que veremos en este apartado, y para reproducir MP3, de una manera tan costosa en recursos que no se va a ver. En su lugar, veremos en la parte 3 cómo reproducir otros formatos de música con Maxmod.

Al igual que ocurría con las imágenes, el sonido debe estar en formato RAW para poder ser incluido directamente en el binario. El *makefile* se encargará de convertir los archivos *.raw en las respectivas librerías y objetos y enlazarlo al proyecto.

Para crear el archivo de sonido RAW se pueden utilizar varios programas de edición de audio. A la hora de convertirlo es bueno tener en cuenta la frecuencia de muestreo de los altavoces de la DS: 32768 Hz. De cara a minimizar el *aliasing* en el sonido, es recomendable usar una fracción de esa frecuencia al descomprimir el audio a RAW, como por ejemplo 16384 Hz (un medio de la frecuencia de muestreo del altavoz).

En el ejemplo *sounds* se ve la forma más fácil de reproducir sonidos con ASlib. Primero incluimos los sonidos como cabeceras (las cabeceras se generarán automáticamente al ejecutar make):

```
#include "sfxa.h"
#include "sfxb.h"
```

Cuando inicializamos los sistemas, entre ellos inicializamos los de audio:

```
AS_Init(AS_MODE_16CH);
AS_SetDefaultSettings(AS_PCM_8BIT, 16384, 0);
```

Los modos en los que se puede inicializar ASlib son los siguientes:

AS_MODE_MP3	Usar mp3.
AS_MODE_SURROUND	Usar surround.
AS_MODE_16CH	Usar los 16 canales de la DS.
AS_MODE_8CH	Usar únicamente los canales 1-8 de la DS.

La otra función de inicialización indica la configuración por defecto de los efectos de audio: bitrate (AS_PCM_8BIT ó AS_PCM_16BIT, dependiendo de nuestros archivos), frecuencia de muestreo y retraso (por defecto, mejor dejarlo a cero).

Con esto, para reproducir un sonido, se usará la siguiente macro, que es la forma más sencilla de reproducir efectos de sonido con ASlib.

```
AS_SoundQuickPlay(sfxa);
```

AS_SoundQuickPlay(sound); es una macro de la auténtica función para reproducir sonidos, *AS_SoundDefaultPlay(u8 *data, u32 size, u8 volume, u8 pan, u8 loop, u8 prio);* asignando los valores de volumen a 127 (el máximo), dirección a 64 (centrado), bucle a 0 (desactivado) y prioridad a 0.

Esta función nos devuelve el canal en el que se reproduce el efecto de sonido en un entero, que podemos almacenar para modificaciones del sonido.

Otras funciones interesantes de ASlib para los efectos de sonido son:

AS_SetSoundPan(u8 channel, u8 pan);	Cambia la dirección del sonido, 0 indica el altavoz de la izquierda; 64, centrado y 127 el altavoz de la derecha.
AS_SetSoundVolume(u8 channel, u8 volume);	Cambia el volumen del sonido, con valores entre 0 y 127.
AS_SoundStop(u8 channel);	Detiene la reproducción del sonido del canal indicado.

5. OTROS

En esta sección se abordan ciertos detalles básicos de la programación en NDS que no encajan en ninguno de los apartados anteriores.

Punteros

Los punteros en NDS funcionan de forma distinta a como lo hacen cuando se programa para compatibles. La diferencia radica en que en un PC, la memoria física está compartida por varias aplicaciones dejando la gestión de ésta al sistema operativo, mientras que en la NDS solo se ejecuta una aplicación, que tiene a su disposición toda la memoria de la máquina.

Esto hace que la reserva de memoria con funciones tipo *alloc* o *malloc*, como las funciones para liberar memoria, sean innecesarias, alojando directamente los datos en una posición de memoria.

```
int* puntero = 0x03001000;
```

Así mismo, para asignar valores a esa zona de memoria no se utilizará el operador &, si no que directamente usaremos el nombre de la variable.

```
puntero = 10;
```

Las direcciones de la memoria principal en la NDS están entre los rangos 0x03000000 y 0x033FFFFF, que son 4 MB. Otras direcciones de memoria fuera de estos pueden apuntar a registros con información importante, por lo que si se quieren modificar se debe mirar la tabla de registros en la documentación de las librerías.

Véase **[3]** para ampliar información sobre gestión de memoria programación de muy bajo nivel en NDS.

Estructuras de datos de PAlib

Existen en PAlib un par de estructuras interesantes. Estas son PA_RTC para consultar el tiempo del sistema y PA_UserInfo para consultar la información del usuario que éste ha rellenado en la propia consola.

Los campos de la estructura PA_RTC son:

PA_RTC.Day	El día, con valores entre 1 y 31.
PA_RTC.Month	El mes, con valores entre 1 y 12.
PA_RTC.Year	El año.
PA_RTC.Hour	La hora, entre 0 y 23.
PA_RTC.Minutes	Los minutos, entre 0 y 59.
PA_RTC.Seconds	Los segundos, entre 0 y 59.

Los de la estructura PA_UserInfo:

PA_UserInfo.Name	El nombre del usuario.
PA_UserInfo.BdayDay	El día de su cumpleaños.
PA_UserInfo.BdayMonth	El mes de su cumpleaños.
PA_UserInfo.Language	El idioma, de 0 a 5: 0 → Japonés. 1 → Inglés. 2 → Francés. 3 → Alemán. 4 → Italiano. 5 → Castellano.
PA_UserInfo.Message	Mensaje definido por el usuario.
PA_UserInfo.AlarmHour	La hora de la alarma configurada en la DS.
PA_UserInfo.AlarmMinute	Los minutos de la alarma configurada en la DS.
PA_UserInfo.Color	Color elegido por el usuario, de 0 a 15.

En el ejemplo *datastructs* se puede ver el acceso a estas estructuras (**nota:** el reloj no funciona en emulador, pero sí en la propia consola).

Cambiar nombre e icono a la ROM

Para cambiar el nombre y la descripción de la ROM resultante, hay que editar el Makefile. Al abrirlo con cualquier editor de textos, encontraremos las siguientes líneas:

```
TEXT1 := PAlib Project
TEXT2 := Change this text
TEXT3 := for your project!
```

Cambiando el contenido de esas tres líneas, cambiaremos el texto que se verá en la consola, visible en el menú de la FlashCart que estemos usando, pero también cambiaría en el propio menú de la NDS si compilásemos la ROM en un cartucho. Cada variable representa una línea del texto a mostrar, siendo TEXT1 la primera línea, TEXT2 la segunda y TEXT3 la tercera.

Para cambiar el icono que se ve junto a ese texto, basta con incluir una imagen *logo.bmp* en el directorio raiz del proyecto que sea un mapa de bits de 256 colores, con unas dimensiones de 32x32 y 16 colores en su paleta. El color del primer píxel (arriba a la izquierda) será el color transparente para ese icono.

Fuentes para esta segunda parte

[1] Documentación de libNDS:

http://libnds.devkitpro.org/index.html

Se pueden encontrar todas las funciones, estructuras, definiciones de datos y todos los ejemplos de LibNDS, pero no incluye tutoriales.

[2] Documentación de PAlib:
http://www.palib-dev.com/wiki/doku.php

Aquí se encuentra desde la documentación donde se definen todas las particularidades de PAlib, como tutoriales para el uso de estas funciones.

[3] Manual de programación de NDS por Thefer:

Este es un manual orientado a programar la NDS a muy bajo nivel con LibNDS. La parte de punteros está basada (es un resumen muy superficial) en el texto de este manual.

No tiene página oficial, pero se puede encontrar en lugares como http://nds.scenebeta.com/

3 LIBRERÍAS ESPECÍFICAS

En esta parte de la guía, veremos ciertas librerías que cumplen una función específica, aumentando así las capacidades del homebrew.

Las librerías que se comentarán son:

- **Maxmod:** librería de sonido, más completa que ASlib.
- **LibFat:** librería para navegar por archivos y directorios.
- **DsWifi:** librería que permite la conexión wifi de NDS.

Las tres librerías, si bien fueron proyectos independientes, actualmente están incluidas en devkitPro en su versión más actual, por lo que no es necesario instalar ni descargar nada más.

Además, al venir con devkitPro, la plantilla de PAlib observa en su *Makefile* la opción de que estés usando tanto Maxmod como DsWifi para compilar adecuadamente el material del ARM7. Si no usamos ninguna de las librerías, el modo del ARM7 será:

```
ARM7_SELECTED := ARM7_MP3
```

Si usamos solo DsWifi:

```
ARM7_SELECTED := ARM7_MP3_DSWIFI
```

Y si usamos Maxmod (y además DsWifi):

```
ARM7_SELECTED := ARM7_MAXMOD_DSWIFI
```

1. MAXMOD

Esta librería es capaz de reproducir efectos de sonido en formato WAVE además de música en formato MOD.

La reproducción de música en módulos, la gestión automática de los canales de audio y la optimización de la librería (según su página web, la carga del procesador ARM7 en NDS con 16 canales de música, no supera el 12%) la convierten en la mejor opción para añadir sonido al homebrew creado para NDS.

1.1 Efectos de sonido

La reproducción de efectos de sonido con Maxmod es bastante similar a como se realizaba con ASlib. Ahora los archivos de audio que contienen los efectos de sonido, en lugar de ser RAWs a colocar en la carpeta Data del proyecto, serán WAVEs, que colocaremos en la carpeta Audio.

También será necesario incluir la librería de Maxmod y únicamente las librerías *soundbank_bin.h* y *soundbank.h* (que se generarán en tiempo de compilación), en lugar de una cabecera por cada uno de los sonidos que queramos agregar al proyecto.

```
#include <maxmod9.h>

#include "soundbank_bin.h"
#include "soundbank.h"
```

Después inicializamos Madmox:

```
mmInitDefaultMem((mm_addr)soundbank_bin);
```

El argumento es la dirección de memoria del banco de sonidos, y dado que con el *Makefile* de PAlib la librería se llama siempre soundbank_bin.h, esta función tendrá siempre el mismo argumento.

Lo siguiente será cargar en memoria los sonidos:

```
mmLoadEffect(SFX_SFXA);
mmLoadEffect(SFX_SFXB);
```

Los sonidos tienen el nombre del archivo en mayúsculas, sin extensión y con el prefijo 'SFX_'. En este ejemplo, SFX_SFXA hace referencia al archivo sfxa.wav y SFX_SFXB, a sfxb.wav.

Y por último, reproducimos los efectos de sonido:

```
if(Pad.Newpress.A) mmEffect(SFX_SFXA);
if(Pad.Newpress.B) mmEffect(SFX_SFXB);
```

Al igual que con ASlib, la función básica reproduce los sonidos con las opciones por defecto (volumen máximo, sonido centrado, etc).

Para cambiar el volumen de un efecto de sonido usaremos la función :

```
mmEffectVolume(handle, volume);
```

Con el volumen entre los valores 0 (silencio) y 255 (máximo).

Si queremos cambiar la orientación del sonido en los altavoces, la función a utilizar será:

```
mmEffectPanning(handle, panning);
```

En la que con 0 indicaremos que suene sólo por el altavoz izquierdo; con 255, el derecho y con 127, el sonido estará centrado.

Y si bien en nuestro ejemplo no hace falta, para proyectos más grandes tenemos la función:

```
mmUnloadEffect(SFX_SFXA);
```

Que liberla la memoria del efecto de sonido que recibe como argumento.

1.2 Música

Los módulos de música que Maxmod soporta son los siguientes formatos: MOD, S3M, XM e IT. La ventaja de este formato es que no se almacenan directamente las ondas, si no eventos relacionados a instrumentos, por lo que su tamaño suele ser muy pequeño.

Al igual que con los efectos de sonido, deben incluirse las librerías de Madmox y el banco de sonidos, pero el prefijo que denota los módulos es 'MOD_' siendo así, en nuestro ejemplo, el archivo *music.mod* referenciado con la variable *MOD_MUSIC*.

Una vez inicializado el sistema de audio, para cargar un módulo en memoria usaremos la siguiente función, que recibe como único argumento el módulo a cargar:

```
mmLoad(MOD_MUSIC);
```

Tras ello, para que el módulo empiece a reproducirse, debemos indicar con la función de comenzar:

```
mmStart(MOD_MUSIC, MM_PLAY_LOOP);
```

El segundo argumento de esta función nos permite decir si queremos que el módulo se repita en un bucle infinito (MM_PLAY_LOOP) o si por el contrario, queremos que se reproduzca una única vez y se detenga (MM_PLAY_ONCE).

Si quisieramos parar por completo la reproducción del módulo usaríamos mmStop(); sin argumentos, ya que sólo se reproduce un módulo, por lo que no es necesario indicar cuál hay que parar.

Si queremos liberar la memoria que ocupa el módulo, debemos liberarla del siguiente modo:

```
mmUnload(MOD_MUSIC);
```

Con ello dejamos libre la reproducción de módulos y podemos cargar otro en su lugar. Esto, en un videojuego por ejemplo, serviría para tener una melodía distinta en cada nivel del juego.

En el ejemplo se puede ver cómo pausar un módulo que se está reproduciendo:

```
if(Pad.Newpress.A && mmActive()) mmPause();
else if(Pad.Newpress.A && !mmActive()) mmResume();
```

La función *mmActive*() devuelve cero si el módulo está parado o pausado y otro valor en caso contrario, por lo que nos permite comprobar el estado para decidir si pausamos o si reanudamos el módulo.

Nota: El hardware de la NDS no permite pausar realmente, por lo que Madmox hace una pequeña trampa software para pausar la reproducción, que si bien suele funcionar bien, puede producir algunos problemas con ciertas muestras de sonido.

Durante la reproducción, podemos modificar ciertos parámetros del módulo, tales como el volumen, el tempo o la altura:

```
// Control del volumen.
volume += Pad.Held.Up - Pad.Held.Down;
if(volume > 1024) volume = 1024;
if(volume < 0) volume = 0;

mmSetModuleVolume(volume);

// Control del tempo.
tempo += Pad.Held.Right - Pad.Held.Left;
if(tempo > 2048) tempo = 2048;
if(tempo < 512) tempo = 512;

mmSetModuleTempo(tempo);

// Control de la altura.
pitch += Pad.Held.R - Pad.Held.L;
if(pitch > 2048) pitch = 2048;
if(pitch < 0) pitch = 0;

mmSetModulePitch(pitch);
```

En el propio ejemplo se pueden ver los valores máximos de los tres parámetros.

En el caso del tempo, 1024 representa la velocidad normal, 2048 sería el doble de velocidad y 512 la mitad. Incrementar el tempo incrementará la carga del procesador para reproducir el módulo.

Como con el tempo, 1024 para la altura es el valor por defecto.

Por defecto, al cerrar la consola, ésta se suspende, deteniendo por tanto la reproducción de música. Para evitar que se suspenda, debemos indicarlo con la siguiente función de PAlib:

```
PA_SetAutoCheckLid(0);
```

Si más adelante quisieramos que la consola volviese a suspenderse, llamaríamos a la misma función, pasándole como argumento un 1. Podemos saber el estado de la bisagra con *PA_LidClosed()* por si queremos dejar de dibujar mientras la consola esté cerrada o podamos volver a hacerlo tras abrirla.

Maxmod permite sólo la reproducción de un módulo de forma simultánea, pero si hay que reproducir otro módulo solapadamente, lo permite con los 'jingles', que, como su nombre indica, su intención original es para reproducir pequeñas melodías, posiblemente realcionadas con eventos.

Los jingles se cargan en memoria igual que los módulos, pero se reproducen con:

```
mmJingle(MOD_JINGLE);
```

A diferencia de la reproducción del módulo no tiene segundo argumento porque los jingles se reproducen siempre en modo MM_PLAY_ONCE, es decir, no es posible realizar bucle.

El único parámetro que se puede modificar de un jingle es el volumen:

```
mmSetJingleVolume(volume);
```

Funciona exactamente igual que su equivalente para módulos, siendo 1024 el máximo y valor por defecto y 0 el mínimo.

2. LIBFAT

Esta librería se basa en el hardware de la NDS, por lo que al probarla en emulador es bastante seguro que nos encontremos con errores inesperados. Es mejor probar lo que se realice con ella directamente en la consola.

El *Makefile* de PAlib ya viene preparado para enlazar Libfat en un proyecto, por lo que para crear proyectos que hagan uso de esta librería solo debemos incluir la cabecera:

```
#include <fat.h>
```

E inicializar la librería en el código antes de hacer uso de cualquiera de sus funciones:

```
fatInitDefault();
```

Y con libfat funcionando, se pueden usar ya las funciones de stdio relacionadas con ficheros, como se ve en los ejemplos *read* y *write*.

Para abrir un archivo se usa:

```
FILE* file = fopen ("text.txt", "rb");
```

Siendo el segundo argumento el modo a abrir el archivo, y siendo los valores siguientes los más habituales:

r/rb	Abrir para escribir
w/wb	Crear o truncar archivo existente.
a/ab	Crear o anexar a archivo existente.

Se puede leer en los archivos (abiertos del modo adecuado con *fopen*) usando la siguiente función:

```
fread(dest, 256, 1, source);
```

Donde *dest* es la cadena de carácteres destino, 256 indica el numero máximo de items a leer, 1 indica el tamaño del carácter en bytes y *source* es el fichero de origen.

Para escribir en un fichero usaremos *fwrite* como se ve a continuación:

```
fwrite("Cadena a escribir en text.txt", 30, 1, text);
```

Donde ahora el primer argumento es el origen de los datos, en este caso una cadena de carácteres directamente, 30 indica el tamaño máximo de la cadena, 1 el tamaño del carácter en bytes y *text* es el fichero destino.

Una vez terminadas las operaciones que queramos realizar con el fichero es muy importante que sea cerrado antes de apagar la consola para evitar pérdidas de datos.

```
fclose(file);
```

Con esto, solo queda el soporte a directorios. Para ello, es necesario incluir al principio del código la librería <sys/dir.h> como se ve en el ejemplo *FATListDirectory* de los ejemplos de PAlib.

Para abrir un directorio, se usa:

```
DIR_ITER* dp = diropen ("/directory/path/");
```

Para iterar por el directorio:

```
dirnext (dp, filename, &filestat);
```

Donde dp es un directorio abierto con diropen, *filename* es una cadena de texto que se rellenará con el nombre del siguiente archivo o directorio en dp/ y *filestat* se rellenará con las estadísticas del archivo.

Para empezar desde el principio del directorio abierto, se usa:

```
dirreset (dp);
```

Y para cerrarlo:

```
dirclose (dp);
```

Las funciones *chdir* y *mkdir* funcionan como los estándares POSIX.

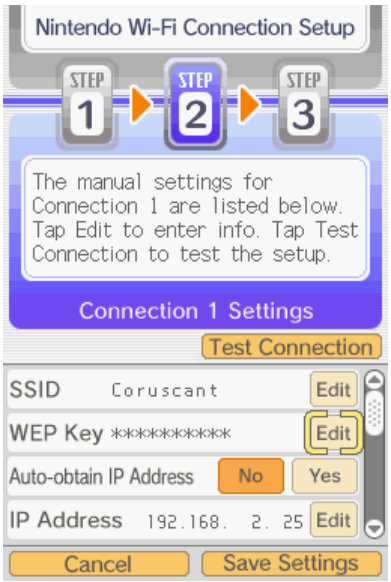
3. DSWIFI

3.1 Configurar la Conexión Wifi de Nintendo

El menú de configuración Wifi de la NDS no es accesible desde un primer momento. Para poder configurar la conexión de nuestra consola, necesitamos un juego que traiga el sello de "Nintendo Wifi Connection":

En los menús del juego habrá alguna opción para acceder a la configuración de la conexión, posiblemente usando el nombre de 'Nintendo WFC'.

Una vez encontremos el menú, configurar la Wifi es fácil, solo hay que seguir los menús rellenando los datos apropiados para el punto de acceso que estemos usando.



Cuando hayamos configurado la conexión, podremos trabajar con DsWifi y el interfaz que nos ofrece PAlib para esta librería.

Nota: En el botón de Options podemos encontrar información necesaria para configurar la conexión, como la dirección MAC de nuestra consola.

3.2 Conectarse al punto de acceso

En el ejemplo *connect*, que es una pequeña ampliación del ejemplo que viene con PAlib, podemos ver cómo conectarse al punto de acceso previamente configurado.

Lo primero es inicializar las funciones Wifi:

```
PA_InitWifi();
```

Y a continuación conectamos con el punto de acceso usando la siguiente función:

```
PA_ConnectWifiWFC();
```

En el ejemplo se realiza una prueba de conexión para asegurarse de que estamos conectados antes de continuar la ejecución del programa.

Con Wifi_GetIP() obtememos la IP asignada a la consola, una vez conectados, en un entero de 32 bits sin signo. Transformarlo a la notación estandar para IPs es cosa del programador.

```
u32 addr = Wifi_GetIP();
```

Esta última función tiene una diferencia significativa con las dos anteriores: el prefijo. El prefijo 'PA_' indica que la función pertenece al set de PAlib y es, por tanto, una abstacción de las funciones de DsWifi. Sin embargo, *Wifi_GetIP* es directamente una función de DsWifi.

Por último, si nuestra aplicación ha acabado todo lo que tiene que hacer en red podemos desconectar y desactivar los subsistemas de Wifi, liberando de carga al procesador.

```
Wifi_DisconnectAP();
Wifi_DisableWifi();
```

Si después se quisiera reconectar, habría que volver a inicializar con *PA_InitWifi()*.

Nota: Al igual que con Madmox, ejecutaremos los ejemplos de DsWifi en la propia consola en lugar de en emulador.

3.3 Sockets

Los sockets en DsWifi están programados simulando el API de los sockets Berkeley, por lo que para usarlos, se puede buscar documentación de estos últimos

A la hora de crear un socket, PAlib tiene una función para hacerlo fácilmente:

```
int sock;
PA_InitSocket(&sock, host, port, PA_NORMAL_TCP);
```

El servidor en la función puede ser tanto una IP como un nombre de dominio. El puerto será el puerto en el que se conecte el socket. El último argumento es el modo del socket: PA_NORMAL_TCP es un socket TCP normal, para crear un socket TCP no bloqueante usaremos PA_NONBLOCKING_TCP.

Usando sockets normales (como en los ejemplos incluidos), la ejecución del programa quedará bloqueada hasta que se cumpla la conexión o haya un *timeout*.

Si lo que queremos crear es un socket de escucha para un servidor:

```
int sock;
PA_InitServer(&sock,9999,PA_NORMAL_TCP,10);
```

Tras el puerto y el modo del socket, indicamos el número máximo de conexiones. En NDS, por limitaciones de hardware, el máximo ronda los 20.

Y para poner el socket a escuchar, basta con la funcion *accept* de sockets normales:

```
connectedclient = accept(sock,NULL,NULL);
```

Enviar y recibir mensajes también funciona del modo de los sockets de Berkeley:

```
send(sock,buffer,256,0);
```

```
recv(sock,buffer,256,0);
```

El tercer argumento indica el tamaño del buffer.

Fuentes y enlaces de interés para esta tercera parte

[1] Página oficial de Maxmod
http://www.maxmod.org/

Inlcuye la referencia con todas las funciones de la librería, así como los tipos y definiciones.

[2] Página oficial de Libfat
http://chishm.drunkencoders.com/libfat/

Además de página oficial, es una guía de referencia rápida para usar la librería.

[3] Página oficial de DsWifi
http://www.akkit.org/dswifi/

La documentación de esta librería explica a nivel de registro el funcionamiento de la librería. Además tiene un foro donde se pueden encontrar ejemplos y soluciones a problemas por parte de la comunidad.

[4] Antigua wiki de PAlib
http://www.palib.info/wiki/doku.php

Si bien está desactualizada y abandonada en favor de la documentación enlazada en la parte 2 de esta guía, incluye ejemplos de las librerías vistas en esta parte que todavía son válidos con las versiones actuales de las librerías.

CONCLUSIONES

El objetivo de este documento pretende dar un punto de apoyo incial para quien quiera programar en la consola Nintendo DS; primero, conociendo un poco la máquina sobre la que vamos a programar (parte 1); segundo, presentando las técnicas y herramientas fundamentales para desarrollar una aplicación que nos ofrece PAlib (parte 2); y tercero, presentando otras librerías que nos permiten ampliar las capacidades de nuestras aplicaciones (parte 3).

La primera parte cumple, sin ser una guía demasiado estricta, pues podría ampliar la información hablando de la arquitectura de la consola, de los registros de su memoria y más material que ayudaría sobre todo a quien quisiera programar en bajo nivel, pero dado que en el resto de la guía me centro en PAlib, he considerado innecesario profundizar tanto en el hardware, mostrando únicamente lo que siempre hay que tener presente: que se está programando para una máquina pequeña en lugar de para un pc de sobremesa donde tenemos memoria de sobra y un sistema operativo gestionándolo todo detrás.

Con la segunda parte de la guía, la más extensa, es con la que más satisfecho estoy. Con ejemplos desarrollados por mí y el texto aclaratorio generado a partir de esos pequeños ejemplos, presenta herramientas con las cuáles ya se pueden desarrollar aplicaciones y juegos completos para Nintendo DS.

Por otro lado, me hubiese gustando ahondar más en la librería *libNDS*, pero a partir del subapartado de gráficos la gestión del hardware de la consola se complicaba hasta tal punto de que de haber seguido por ahí, la gestión de registros, memoria gráfica y demás habría requerido un apartado propio en la guía.

También se echan en falta funciones 3D básicas, pero PAlib eliminó sus funciones a tal propósito y la parte de 3D habría requerido hacer uso de las herramientas que nos ofrece devkitPro, que si bien intentan simular a OpenGL (siendo las funciones bastante similares) volvía a requerir, como he dicho antes, una excesiva gestión del hardware por parte del programador.

Por último, sobre la tercera parte de la guía, la intención es simplemente presentar otras librerías que se pueden utilizar conjuntamente a PAlib. El subapartado de Maxmod creo que es una guía bastante completa de la librería; sin embargo, los de las otras dos librerías simplemente las dan a conocer, ya que tanto en libfat como en dswifi se puede profundizar mucho más. Para conseguir ejemplos más 'vistosos' de libfat sería necesario presentar otras librerías, que permitiesen por ejemplo streaming de audio directamente desde un archivo o que permitiesen cargar imágenes sin convertirlas previamente a vectores.

También me hubiese gustado comentar alguna otra librería, como por ejemplo el motor 3D Nitro Engine o la librería Thmp3 compatible con streaming de mp3, pero la guía debía tener un punto final, y dado que con lo que tiene, permite realizar ya aplicaciones homebrew funcionales, decidí ponerlo ya.

Iván López Rodríguez
10 de Julio de 2011