DESARROLLO DE APLICACIÓN SOBRE NINTENDO DS PARA TRANSFERENCIA DE DATOS VÍA WI-FI

Alumna: Nuria Martínez Soriano (numarso@ei.upv.es)

Director del proyecto: Manuel Agustí Melchor

Departamento: Departamento de Informática de Sistemas y Computadores (DISCA)

Centro: Escuela Técnica Superior de Ingeniería Informática (ETSINF)

Titulación: Ingeniería Técnica en Informática de Sistemas

Universidad Politécnica de Valencia

Código P.F.C.: DISCA-158

Fecha P.F.C.: 2011

Índice de contenido

Índice de figuras

Nota: Las figuras 10 y 69 cambian la nomenclatura porque son un conjunto de figuras que representan pasos a seguir en una configuración (de Code::Blocks y configuración Wi-Fi vía WFC respectivamente).

1 Introducción

El proyecto se centra en dos objetivos, el primero consiste en el diseño e implementación de una aplicación para Nintendo DS, que se encarga de evaluar los trabajos realizados por los alumnos de la asignatura de Integración de medios digitales (IMD), de la intensificación de multimedia, a través de valoraciones y comentarios ;en el segundo objetivo se pretende estudiar la conexión y las funciones Wi-Fi que soporta la consola gracias a una librería llamada **dswifi**, añadiendo a la aplicación la posibilidad de realizar transferencia de datos.

Como trabajo previo se ha desarrollado una primera versión de esta aplicación con las siguientes características:

- Primero se muestra en la pantalla superior los títulos de los trabajos, obtenidos de un fichero de texto, y en la inferior un menú con el número de trabajos que existen, dando la posibilidad de elegir uno mediante el pad de la consola.
- Después de la elección de un trabajo, en un nuevo apartado podemos optar la manera de escribir un comentario de texto, o bien con un teclado virtual o mediante reconocimiento de caracteres.
- Seguidamente, aparece un nuevo apartado para dar valores del 1 al 3 a ciertos aspectos que hay que evaluar sobre el trabajo, por ejemplo, valoración global, navegación, programación, etc. (1 regular, 2 bien, 3 muy bien)
- Finalmente, nos dice el nombre del fichero de texto que hemos creado con la información que hemos escrito y vuelve al menú de los títulos. El nuevo fichero se encontrará en la tarjeta micro SD en el mismo directorio donde esté el ejecutable.

Éstas características se mantendrán en la nueva versión de la aplicación, pero con notables mejoras que se han realizado y se irán explicando más adelante una a una en el resto de apartados, siendo las siguientes:

- Inclusión de interfaz gráfica. Añadiendo botones para interactuar con ellos.
- Más utilidad a la pantalla táctil. Aprovechando al máximo la especialidad de la consola.
- Flexibilidad en la navegación de la aplicación.
- Creación de un cursor para el teclado y el reconocimiento de caracteres.
- Incorporación de mayúsculas y caracteres nuevo en el reconocimiento de caracteres. Caracteres como las vocales acentuadas, números, la “ñ”, símbolos...
- Código fuente mejor estructurado. Organizados en varios ficheros en C.

Todo ello dando a facilitar la usabilidad y simplicidad de cara al usuario hacia la aplicación, además del diseño.

También se incorpora una característica nueva muy importante en el desarrollo, la **transferencia de datos**. Se obtendrán los títulos de los trabajos a evaluar (en formato .txt) mediante una descarga al sitio web donde está almacenada y también se podrá enviar a dicha web el contenido creado en la aplicación, las evaluaciones, que para llevarlo a cabo se ha realizado un pequeño servidor en PHP que sólo se encarga de esta tarea.

2 Hardware

Son varios los dispositivos hardware necesario para hacer funcionar el “homebrew” realizado durante el proyecto, además trabajar con red inalámbrica: tener una Nintendo DS, un flashCard , router...

2.1 Especificaciones Técnicas

El proyecto se ha probado con dos versiones de Nintendo DS distintas, la más antigua de todas, y la siguiente, la DS Lite. A continuación se explican las características técnicas de las consolas, tanto sus diferencias y sus puntos comunes entre ellas.

Nintendo DS original

También conocida como “la Dsfat” ya que en comparación con los modelos más nuevos, es gruesa y pesada. Salió a la venta en el año 2004. Si bien tenía un tamaño razonable para un sistema portátil, es casi el doble de ancho que la Lite.

- **Dimensiones :** 148,7 x 84,7 x 28,9 (en milímetros, ancho * largo * alto)
- **Peso:** Aproximadamente, 275 gramos.
- **Pantallas:** Dos pantallas de 3 pulgadas, TFT LCD, 256x192 píxeles (260,000 colores diferentes), 0.24mm de distancia de punto. La inferior añade un panel táctil transparente.
- **Batería:** Batería de ión litio que proporciona entre 6 y 10 horas de juego con la carga al máximo (dependiendo de las aplicaciones que uses). Se carga en unas 4 horas aproximadamente. Adaptador AC (También compatible con GBA SP). Modo Sleep incorporado para ahorrar batería.

Figura 1: Nintendo DS original

Nintendo DS Lite

La Nintendo DS Lite es un 21% más ligera que el modelo original. En Europa salió a la venta el 23 de junio del 2006. La posición de los botones ha sido cambiada ligeramente, como el botón ON/OFF, los botones SELECT y START. Las luces de encendido y de carga han sido cambiadas a la parte superior derecha y el micrófono se sitúa en el centro. Es 100% compatible con la Nintendo DS original, tanto en modo LAN como en Wi-Fi.

- **Dimensiones NDSL:** 133 x 73,9 x 21,5 (en milímetros, ancho * largo * alto)
- **Peso:** Aproximadamente, 218g.
- **Pantallas:** Dos pantallas TFT/LCD reflectoras semitransparentes a color, de 3 pulgadas, con luz incorporada de intensidad variable, con una resolución de 256x192 píxeles y 0.24 mm de tamaño de punto. La inferior táctil.
- **Batería:** Batería de ión litio que proporciona entre 6 y 10 horas de juego (brillo al máximo)y entre 12 y 16 horas (brillo al mínimo), tras una carga de 4 horas; modo sleep para ahorro de energía; adaptador AC. (No es compatible con NDS original, ni GBA SP)

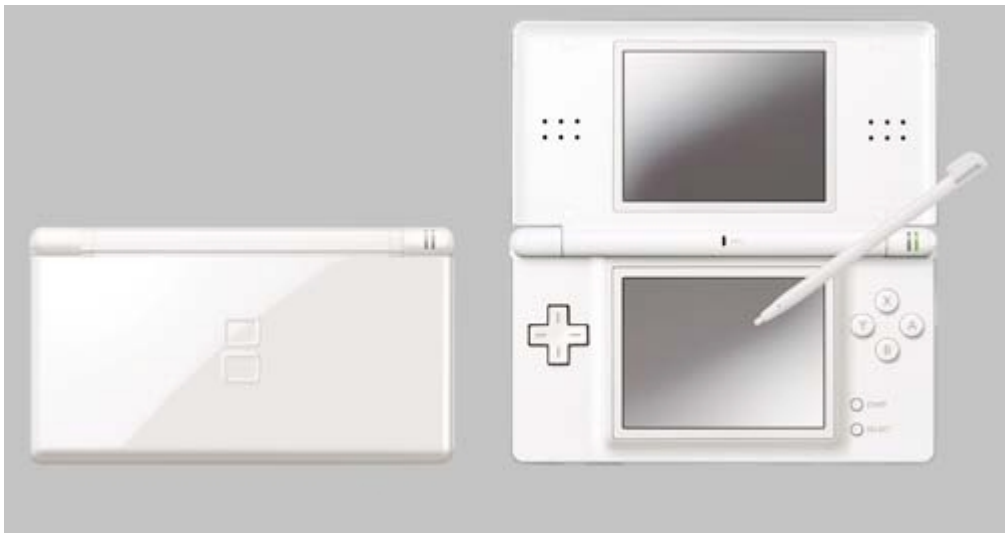

Figura 2: Nintendo DS Lite

Características comunes entre ambas:

- **Procesador CPU:** Dos procesadores, principal ARM9 a ~67Mhz y secundario ARM7TDMI a ~33MHz (modo GBA y control de I/O). El primero se encarga de llevar las riendas del programa: procesamiento de gráficos, vídeo y la lógica del juego. Recoge las instrucciones de memoria principal, y cuenta con una caché de 32KB para instrucciones y otra de 16KB para datos; el secundario se encarga de gestionar el audio, la red inalámbrica Wi-Fi, algunas teclas de la consola (como los botones X e Y) y la compatibilidad con los juegos de GBA. Tiene acceso a una pseudo-caché 'IWRAM' de 64KB para instrucciones y datos.

- **Dos procesadores gráficos 2D**: uno de ellos con capacidades 3D. Manejo de Sprites por hardware y el 3D 6144 vértices por frame.

- **Memoria principal:** 4 MB, Se encarga de almacenar los ejecutables y la mayoría de los datos de juego en curso.

- **Altavoces:** Altavoces internos estéreo que pueden proporcionar sonido "virtual surround" dependiendo del software. Además, dispone de 16 canales de sonido que son independientes.

- **I/O:** Puertos para cartuchos de juegos de DS o para juegos de Game Boy Advance y entrada para auriculares estéreo y micrófono.

- **Controles:** Botones direccionales (Arriba, Abajo, Derecha, Izquierda), Botones A B X Y, gatillos L y R, Start, Select, reconocimiento de voz a través del micrófono, pantalla táctil y botón POWER. Los controles de la NDS Lite son los mismos pero con algunos cambios de localización.

- **Comunicación inalámbrica:** IEEE 802.11b, siendo 802.11 un protocolo de intercambio de datos. Las designaciones 'a', 'b' y 'g' son las velocidades de red. Nintendo DS es compatible con los routers que aceptan 802.11b y 802.11g. Solo soporta encriptación WEP (*Wired Equivalent Privacy*) y un protocolo especial inalámbrico creado por Nintendo que usa cifrado RSA en la señalización (Ej. PictoChat usa este protocolo) .
  El rango de comunicación inalámbrica varía de 10 a 30 metros, dependiendo de las circunstancias. En muchos casos, varios jugadores pueden participar en la misma partida multijugador utilizando un solo cartucho.
  La señal de Wi-Fi recibida es más débil que la otros dispositivos , como por ejemplo un portátil.

- **Otras Características:** Software PictoChat incluido en la consola y que permite a un máximo de 16 usuarios chatear simultáneamente entre sí. Reloj a tiempo real incluido (RTC), con alarma (no disponible mientras se juega), fecha y hora. Idiomas: Español, inglés, japonés, francés, alemán, italiano.

2.2 FlashCard

Para poder realizar “homebrew” (aplicaciones y juegos caseros) en una Nintendo DS, es necesario disponer de una flashCard. Es un dispositivo hardware, del mismo tamaño de un cartucho original para DS, con el cual se puede almacenar copias de seguridad y ejecutar el “homebrew” realizado . Muchos son los modelos que existen hoy en día en el mercado, de los que se diferencian en dos grupos:

- **Flashcards Slot1:** Son los más abundantes en la actualidad. Corresponden a los cartuchos del mismo tamaño que los juegos para la consola y la gran parte de ellos necesitan una micro SD para almacenar la información ejecutable (juegos, videos, música, aplicaciones...).
  Para funcionar, necesita en la tarjeta su propio "kernel" o “firmware”, siendo éstos cartuchos actualizables mediante nuevos “kernel” que se ponen a disposición en las web de cada tarjeta. Debe estar almacenado en la micro SD. Lo único es que no permite la ejecución de roms de GBA.
  Es el modelo que yo recomiendo para trabajar con “homebrew” por el gran catalogo existente de modelos y fácil manejo. Es el cual he usado para mi aplicación.

- **Flashcards Slot2:** Algo obsoletos, estas flashcards se colocan en el Slot2 de la DS, que es donde van los juegos de GBA. La mayoría si permiten la ejecución de roms para GBA. Aunque hay que tener cuidado con ellas porque como el Slot2 no es del mismo tamaño en la NDS original y en la NDS Lite, la compatibilidad no es la misma para ambas consolas.

Figura 3: Muestras de flashCards de ambos tipos

En adelante me voy a centrar sólo en las de tipo Slot1, ya que como he nombrado anteriormente, es el tipo que he usado para mi trabajo. También es importante nombrar que la Nintendo DSi usa otro tipo especial de flashCard, ya que la arquitectura de esta consola es distinta de las otras NDS, además solo usa la del grupo Slot1, ya que la NDSi ya no dispone de la ranura para juegos de GBA.

Dentro de esta gama, las flashCards más habituales son: R4 DS Revolution, M3 DS Real, SuperCard DSONE, CycloDS Evolution, entre otras. Para el proyecto he usado la **R4 Revolution** con dos actualizaciones de "kernel" distintas, siendo más actual *Wood R4 v1.29* (abril 2011), *Wood R4 v1.13*, y la otra *Kernel R4DS v1.11* además de otra flasChard, **la M3 DS Real** con el "kernel" *Sakura versión 1.48* (M3 necesita un cartucho para el slot2).

Para averiguar la versión instalada de un "kernel" dentro de la tarjeta, ésta debería de verse en algún lugar de la pantalla desde el propio menú del "kernel" (como es el caso de R4DS v1.11) o en caso contrario, deberá existir un submenú con uno de los apartados llamado "help information" con toda la información sobre el "kernel".

Figura 4: Hardware necesario para el "homebrew"

Instalación de un "kernel"

Primero hay que obtener la versión de "kernel" más actual del momento desde la página oficial de la flashCard correspondiente. Después descomprimir los archivos del .zip o .rar, donde habrán carpetas y archivos. Todo esto se debe de copiar a la tarjeta micro SD directamente (no dentro de ninguna carpeta). Importante aclarar que según la versión del "kernel", el nombre de los archivos puede verse modificado.

En la **figura 5** se muestra el contenido de archivos que contiene el "kernel" de la tarjeta R4 versión Wood v1.29 que se debe meter en la micro SD tal cual aparece. La carpeta **homebrew** la he creado yo misma para incorporar los "homebrew" realizados durante el proyecto. No es necesario que se llame así, ni tampoco la creación de ninguna carpeta, cualquier ejecutable se pueden meter en el directorio raíz si se desea.

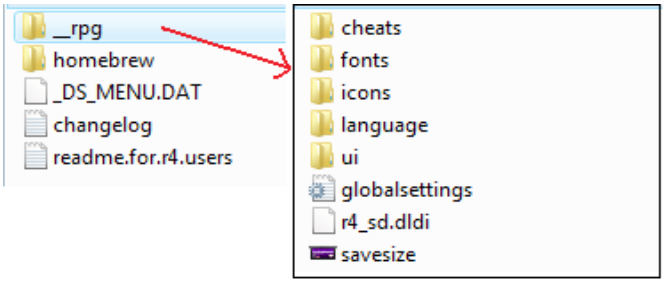

Figura 5: Archivos del "kernel" de Wood v1.29

2.3 WIFI Link

Si existiese el problema de tener un router incompatible con la Nintendo DS o porque el cifrado es WPA, la solución es una tarjeta inalámbrica portátil, de manera que permite conectar dispositivos Wi-Fi, como por ejemplo el Conector USB Wi-Fi de Nintendo, WIFI Link, WIFI Max, que son del tamaño de un pendrive, con puerto USB. Se ha trabajado con **WIFI Link** (IEEE 802.11 b/g), **figura 6**. Éste dispositivo permite dos modos de conexión, como estación para un PC a cualquier señal inalámbrica y como punto de acceso, para crear redes locales. Incluye un CD con un manual de usuario y los controladores para su instalación.

Figura 6: WIFI Link

Gracias a éste punto de acceso, se puede conectar la Nintendo DS a internet, entre otras consolas como PSP, Wii o PS3. Es capaz de soportar hasta 5 NDS al mismo tiempo y la señal llega hasta 200 metros aproximadamente. Además es compatible con Windows 98/SE/Me/2000/XP/Vista, Linux y MAC, y en el CD se encuentran los controladores de cada SO correspondiente dentro de la carpeta Drivers. Aunque parece que en Vista existen muchos problemas para la instalación de los controladores, llevándome a buscar otros controladores por la red.

Problema

Hasta ahora, he seguido tutoriales cuyos enlaces se encuentran en la bibliografía (**apartado 12**) de este PFC (Tutorial funcionamiento WIFI link) pero no he conseguido hacerlo funcionar ni en XP ni Vista, pues una vez instalando los controladores e intentar ejecutar el programa, apenas se abre un segundo y se cierra, cada vez que se conectase WIFI link, debería de aparecer un icono en la barra del escritorio (**figura 7**) y no aparece. En la página 14 del manual de usuario explica la parte de la configuración para crear un punto de acceso inalámbrico, donde además explica que para Windows XP/2000 automáticamente detectarían el USB de WIFI Link (no ocurriendo este caso cuando lo probé en XP).

Figura 7: Icono ZD

3 Herramientas de desarrollo

DevkitPro

Es un paquete de herramientas independientes (librerías, compiladores y utilidades) escogido para el desarrollo de la aplicación (versión *devkitpro 1.5.0.exe*). No sólo está disponible para Nintendo DS, sino que también lo está para GBA, GP32, Playstation Portable (PSP), GameCube y Wii. Además es multiplataforma, están disponibles para Windows, Linux y Mac OSX. El proyecto se ha realizado para Windows (XP y Vista) por razones de accesibilidad y comodidad.

Está compuesta de varios compiladores, devKitARM (NDS, GBA y GP32), devKitPPC (Gamecube, Wii) y devKitPSP (PSP). De estas tres sólo se necesita **devKitARM**, que permite la compilación de binarios compatibles con la arquitectura ARM. El compilador de devKitARM usa GCC (*GNU Compiler Collection*), que es un compilador integrado del proyecto GNU para C, C++, Objective C y Fortran. La implementación de la aplicación para Nintendo DS está programada en C.

Sobre la instalación de devKitPro, es muy sencilla, aunque para ahorrar memoria es recomendable desmarcar las opciones de devKitPPC y devKitPSP, ya que no se trabaja en ningún momento con ellas, ocupan bastante memoria y alargan la instalación. Simplemente es pulsar el botón “siguiente” como muestra la **figura 8** hasta que comience a instalarse.

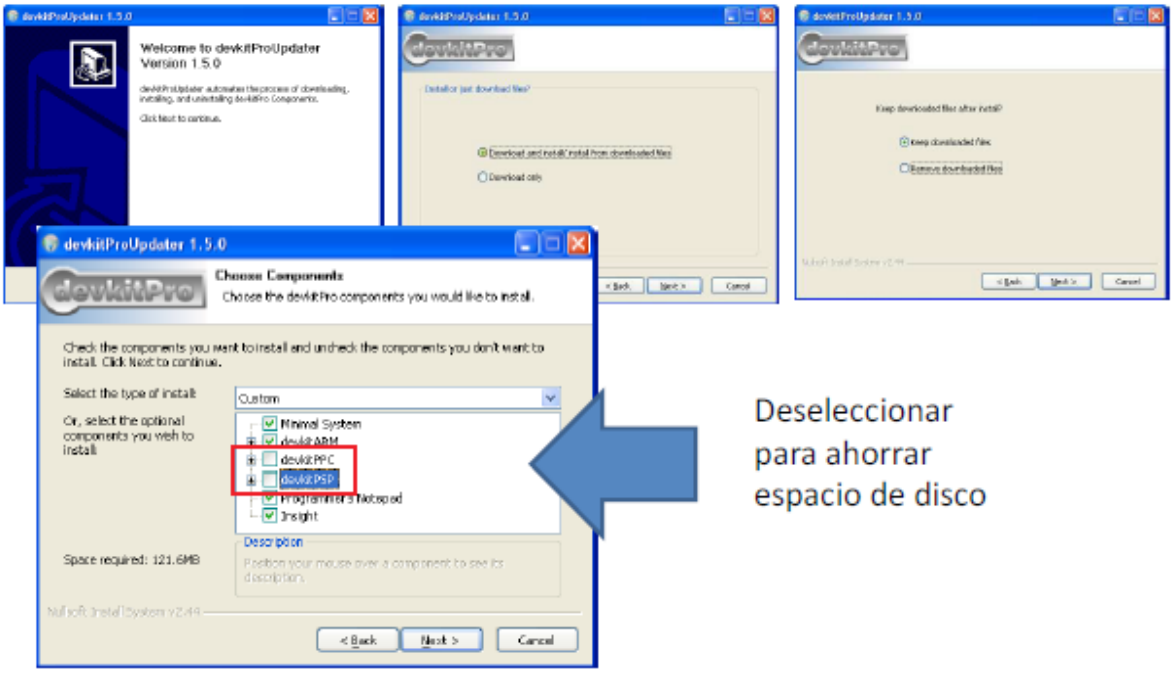

Figura 8: Guía para la instalación de devKitPro en Windows

DevKitPro viene con las librerías **libnds** que son las librerías que adaptan el código C/C++ que realiza el hardware específico de la NDS. Además de las libnds, también viene con las librerías **dswifi, libfat** y **libfilesystem**, muy importantes para el proyecto, ya que dswifi (versión *0.3.13*) se centra en las funciones Wi-Fi y las otras dos para tratar con ficheros . Otra librería menos importante para el proyecto que también viene incluida es **maxmod**, para el sonido.

Librerías no incluidas en DevKitPro

Aparte de éstas librerías se ha utilizado las **PAlib** (versión *PAlib 100707 "QuickFix")*, un conjunto de librerías basadas en libnds que facilitan su uso, además de aportar la herramienta de PAGfx para la conversión de los gráficos (sprites y background) y el emulador No&GBA. PAGfx ha sido usado para realizar la interfaz gráfica de la aplicación, que se explicará más detalladamente en el **apartado 5**.
Para la instalación de las PAlib, descomprimir en el directorio de devKitPro y recompilar PAlib en C:\devkitPro\PALib\source ejecutando clean.bat y seguidamente build.bat.

También existe otra librería llamada **Woopsi** que no ha sido usada en este proyecto porque se han realizado pruebas y no han salido como esperaba ni me han convencido, además de que el *Makefile* necesita ser modificado para poderla usar junto con PAlib. Es una librería que permite programar “homebrew” con el aspecto de la interfaz gráfica de ventanas de Amiga programadas en C++. Tienen unas características que la hacen única, por ejemplo, soporte para múltiples ventanas, modo debug**,** los elementos (ventanas) se pueden arrastrar, etc.

Y por último las **NightFoxLib**, que podrían ser una alternativa a PAlib y parecidas. Solo programadas para las últimas versiones de devkitPro, con funciones Wi-Fi (trabajan solo con el protocolo UDP), dibujo de fondos, rotar textos, cargar sprites, etc.

Code::Blocks

Es un IDE (Entorno de desarrollo integrado “open source”) que ayuda en lo que se refiere a programación en general, y es la herramienta que he usado para desarrollar la aplicación (versión *10.05*). Es extensible mediante plugins, totalmente configurable y además multiplataforma, al igual que devKitpro.

Puede enlazarse a una variedad de compiladores. Por defecto, Code::Blocks buscará una serie de compiladores y configurará los que encuentre. Algunos de los compiladores compatibles:

- Microsoft Visual Studio Toolkit (una extensión de compilador de C++ de Microsoft)
- GCC, en sus versiones para Microsoft y GNU/Linux.
- Borland C++ Compiler
- Digital Mars Compiler
- Intel C++ Compiler
- Open Watcom

Como para este proyecto, se necesita el compilador que viene con devKitPro, el devKitArm, con Code::Blocks es posible añadir compatibilidad con otros compiladores y configurarlo según las necesidades.

Configuración del IDE

En este apartado, explicaré la configuración del compilador de devKitPro para que sea compatible en Code::Blocks. Antes de nada, devKitPro debe de estar ya instalado en el PC. La información se ha obtenido y modificado de un tutorial en el apartado Code::blocks (segundo enlace) de la bibliografía (**apartado 12**). Las **figuras 10.X** indican los pasos a seguir en la configuración.

Figura 9: Logotipos de Code::Blocks y devKitPro

En primer lugar, hay que agregar el compilador, para ello nos dirigimos a **Settings -> Compiler and Debugger...**

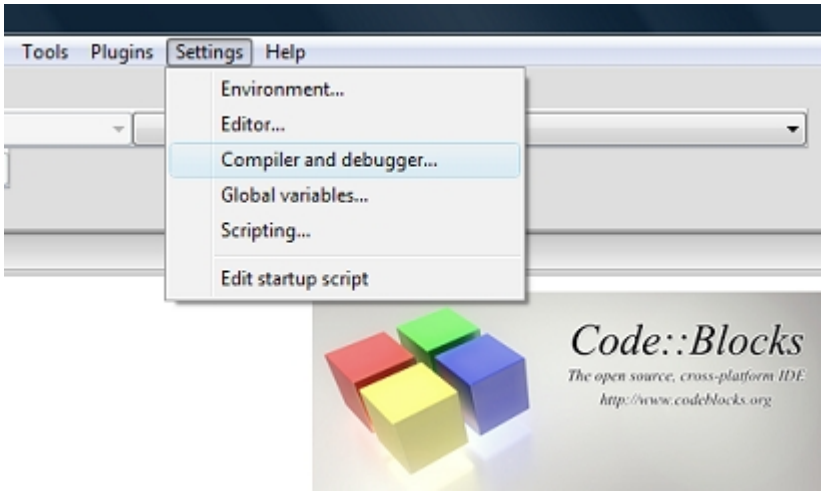

Figura 10.1: Configuración paso 1

Seguidamente tenemos que seleccionar un compilador de la lista, cualquiera valdría pero yo elegí **GNU ARM GCC Compiler** porque es el que más se parece al de ARM de DevKitPro.

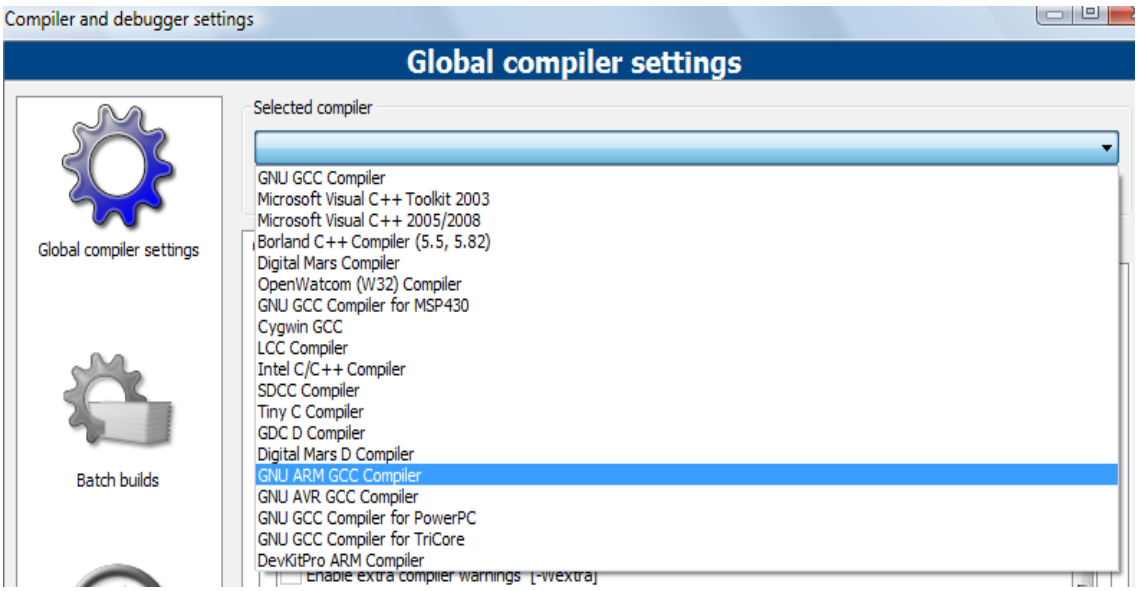

Figura 10.2: Configuración paso 2

Una vez seleccionado GNU ARM GCC Compiler u otro, hacemos clic en el botón **Copy**, esto duplicará el compilador y habrá que ponerle otro nombre para identificarlo después. yo le puse **DevKitPro ARM Compiler**.

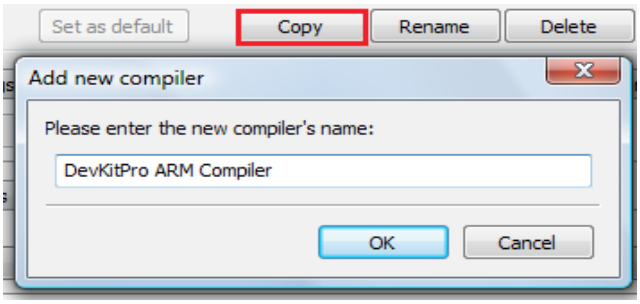

Figura 10.3: Configuración paso 3

Al hacer clic en **OK** nos aparecerá una ventana de aviso que nos recuerda que asignemos la ubicación de los programas de compilación, simplemente hacemos clic en **Aceptar**.

Nos vamos a la pestaña **Toolchain Executables** y completamos los valores con los programas de devKitPro, pueden ser los mismos que se muestran en la **figura 10.4**. Lo verdaderamente importante es la ubicación del *msys* y el último (la orden *make*, que es el que se ejecuta al momento de compilar, ya que usaremos el *Makefile* que siempre usamos. Los otros se pueden completar pero no son necesarios).

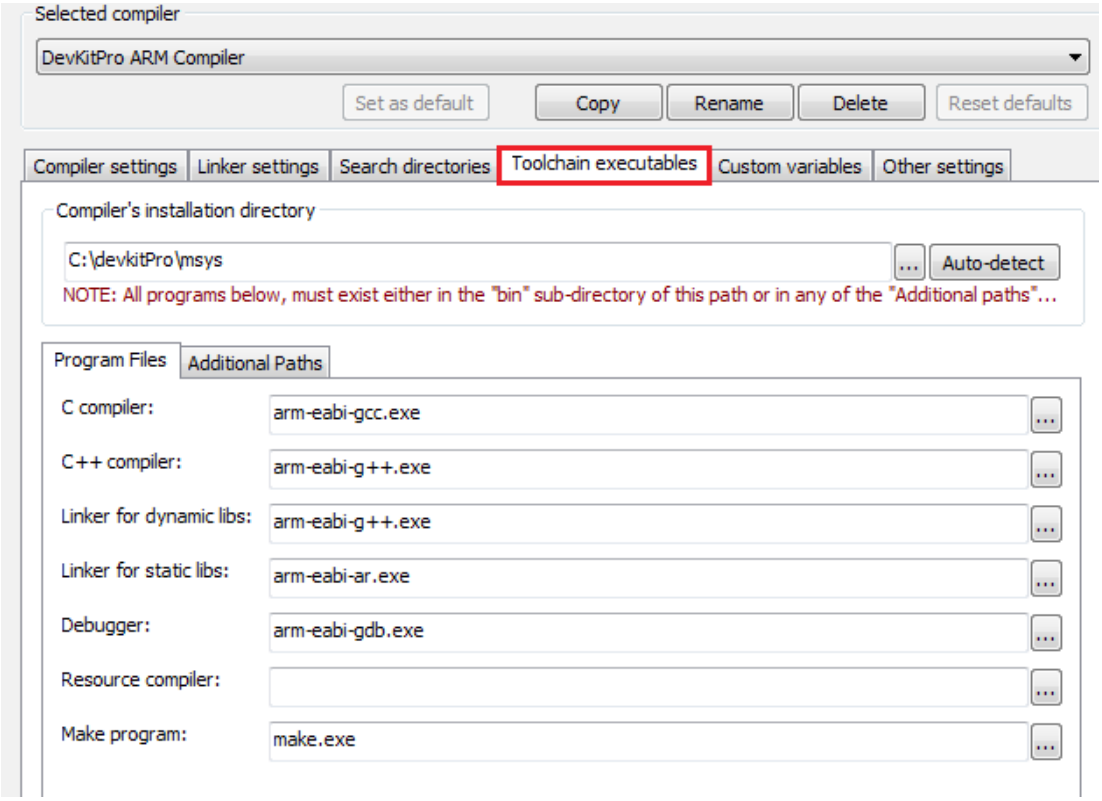

Figura 10.4: Configuración paso 4

En caso de querer asignar otras ordenes, tenemos que ir a la subpestaña **Additional Paths** y colocar la ubicación de los programas del d*evKitARM*. Al terminar, hacemos clic en **OK** y ya hemos terminado de instalar el compilador.

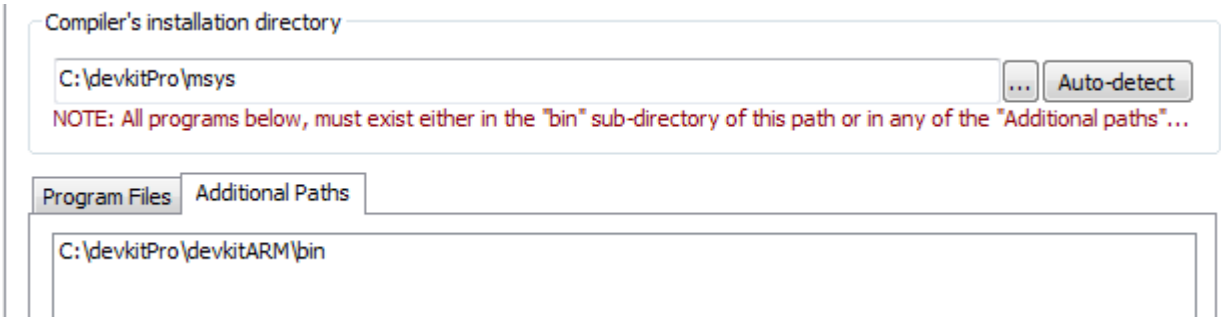

Figura 10.5: Configuración paso 5

Ahora solo hay que comprobar que lo que se ha hecho funciona correctamente. Abrimos un nuevo proyecto en Code::Blocks, **File -> New -> Project...** Seleccionamos **Empty Project** (proyecto vacío) y luego clic en **Go**.

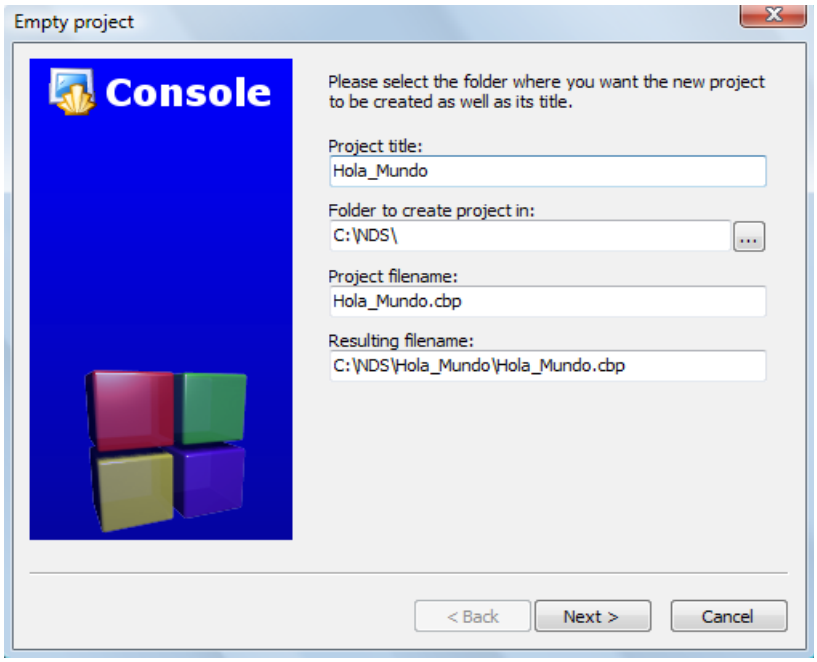

Figura 10.6: Configuración paso 6

A partir del "paso 6" indicado en la **figura 10.6** la configuración se debe hacer con cada uno de los nuevos proyectos o pruebas que se creen, ya que la configuración que se ha realizado del compilador si se guarda, pero la que vamos a explicar ahora es independiente para cada proyecto. Esta parte es importante para que funcione correctamente sin quejas del compilador. También es necesario que el nombre de la carpeta donde estén las directorios de los proyectos sea un nombre sin espacios (C:\NDS\ es como se llamará en mi caso).

Elegimos el compilador que hemos añadido en la casilla Compiler, **DevkitPro ARM Compiler** o con el nombre que se haya puesto. Desmarcamos la casilla *Create "Debug" configuration* si ésta estuviese marcada y dejamos la casilla *Create "Realease" configuration* marcada. Clic en **Finish.**

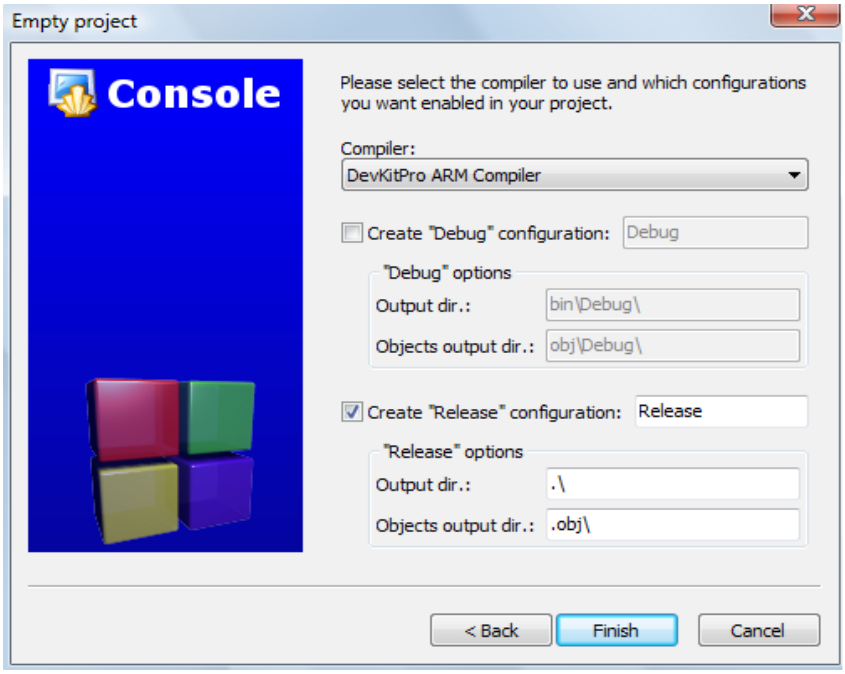

Figura 10.7: Configuración paso 7

Ahora, nos dirigimos a la dirección C:\devkitPro\examples\nds\hello_world, que es donde se encuentra el ejemplo de Hola Mundo de las librerías libnds donde dentro estarán y se deberá copiar la carpeta **source** y el ***Makefile.*** Seguidamente se han de añadir a la carpeta donde está el proyecto que hemos creado con Code::Blocks, es decir en C:\NDS\nombre_del_proyecto_creado. En general se puede utilizar cualquier ejemplo ya realizado, aunque si usa imágenes y/o sonidos, también se debe copiar otras carpetas, como por ejemplo **data**, o **include**.

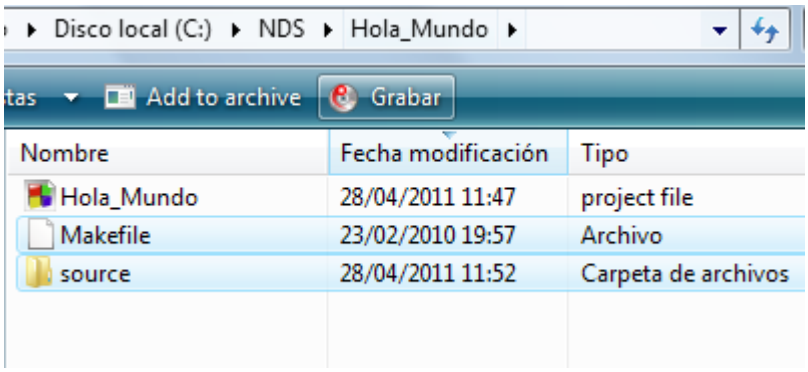

Figura 10.8: Configuración paso 8

En Code::Blocks hacemos clic con el botón derecho en el nombre del proyecto "Hola_Mundo" y seleccionamos **Add files...** para añadirle el código que se encuentra en la carpeta source. El ***Makefile*** también se puede incluir, así como otros elementos como las imágenes y los sonidos, pero no son necesarios. Así se vería el Code::Blocks generalmente:

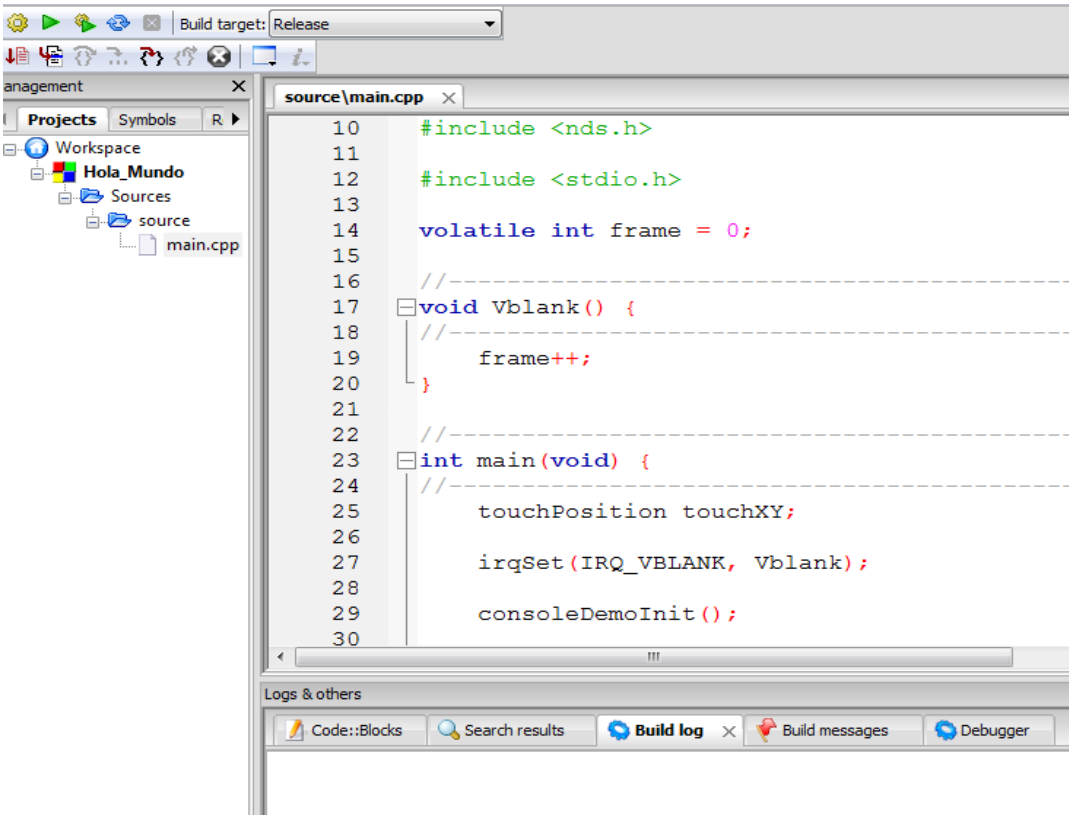

Figura 10.9: Vista del proyecto en Code::Blocks

Para poder compilar todavía falta que ajustar tres cosas muy importantes. Nos dirigimos a **Project -> Properties...** para configurar la forma de compilar y marcamos la opción **This is a custom Makefile**, para indicar que el Code::Blocks use el ***Makefile*** que copiamos en el directorio del proyecto como se marca en rojo en la **figura 10.10**.

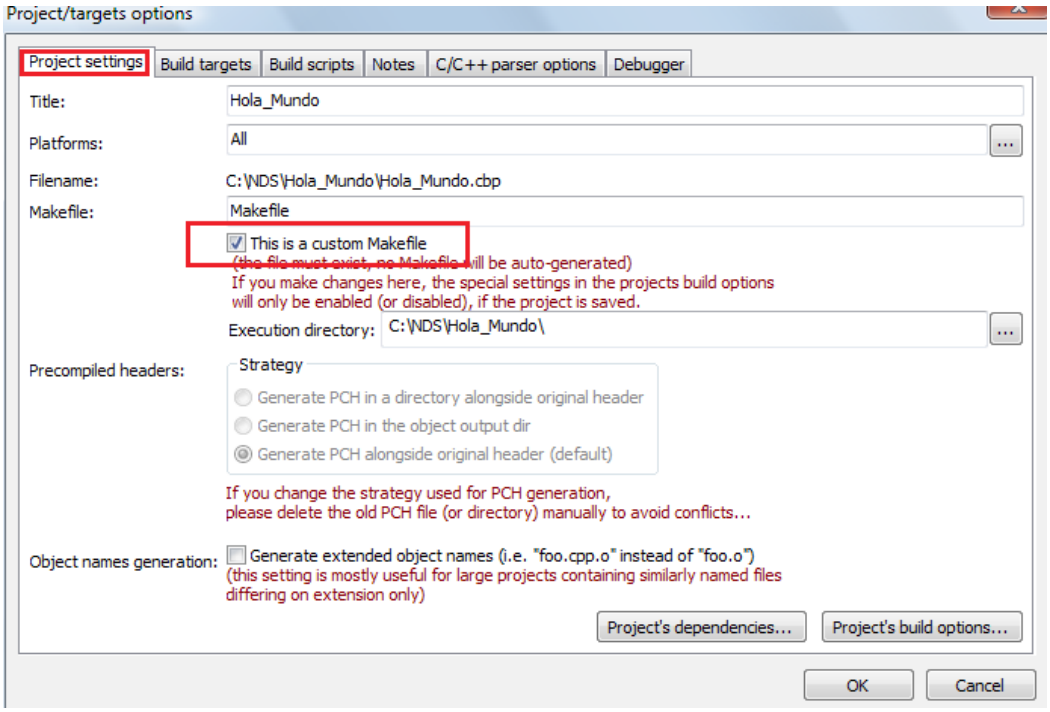

Figura 10.10: Configuración paso 9

Seguidamente en la pestaña **Build Targets** en **Output filename** debemos cambiar el .\Hola_Mundo.**exe** por .\Hola_Mundo**.nds**. Al terminar, damos clic en **OK**.

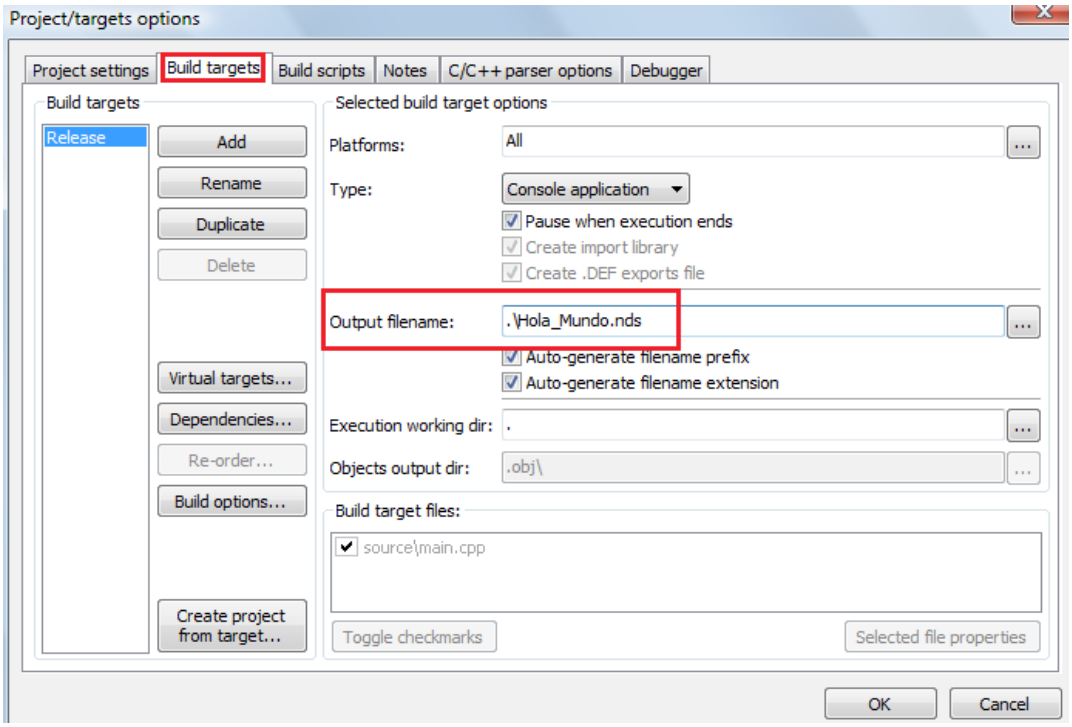

Figura 10.11: Configuración paso 10

Finalmente, para acabar la configuración vamos a **Project -> Build options...** para cambiar los parámetros del *Makefile*. En la pestaña **"Make" commands** colocamos las opciones que aparecen en la imagen “paso 11” de la **figura 10.12**. Al terminar clic en **OK**.

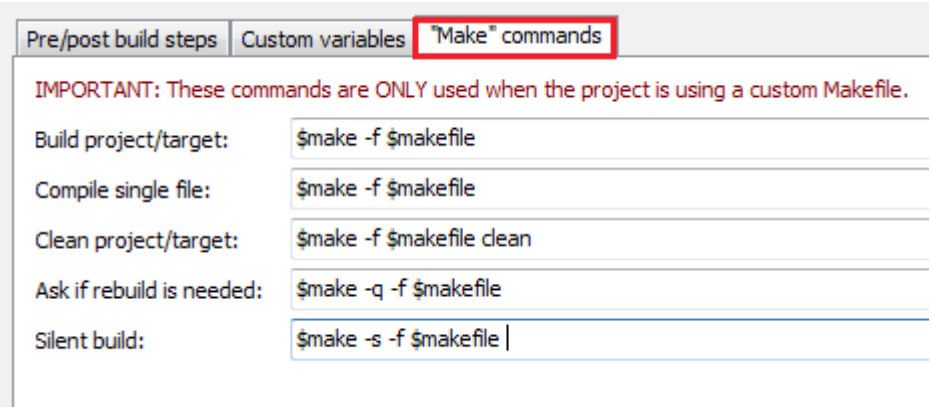

Figura 10.12: Configuración paso 11

Ya está todo listo para compilar el ejemplo pulsando el botón **Build.** Una vez creado el ejecutable .nds ya se puede probar en un emulador, por ejemplo el que viene con las librerías PAlib, No$Gba (**figura 11**).

Figura 11: Ejecución de Hola_Mundo

Emuladores

Es muy importante tener un software para emular todos los ejecutables de prueba que se van realizando durante el proyecto, exceptuando algunas funciones que no van en el emulador, como el tratamiento de ficheros. Existe una colección de emuladores que se encuentran en un programa llamado **WinDS Pro** (versión del año 2011), obtenido de la página oficial de éste. Contiene los siguiente emuladores: No$Gba, idea$, DeSmuME y VBA. El que utilizo es No$Gba, porque a la hora de ejecutar "homebrew" es el que menos problemas me ha dado, pues DeSmuME no ejecuta bien los "homebrews" que he realizado y probando, aunque si funcionan correctamente los juegos de Nintendo DS.

4 Desarrollo para DS con PAlib

4.1 Introducción a PAlib

La implementación de la aplicación se centra en las funciones que ofrece la librería PAlib y el lenguaje C, además de la librería dswifi . Para saber la estructura inicial en PAlib y el cómo organizar los ficheros de un proyecto en distintos directorios, hay plantillas que nos facilitan esta tarea en *C:\devkitpro\PAlib\template* (si es con C) o *C:\devkitpro\PAlib\cpptemplate* (si es con C++).

Una plantilla define los siguientes directorios (obligatorio solo es source):

- **source:** para los ficheros fuente (.c, .cpp y .s, también permite los .h).
- **include:** para los ficheros de cabecera (.h).
- **gfx:** contiene un conversor (PAGfx) y los gráficos del proyecto.
- **data:** para los ficheros de datos del proyecto (se incorporarán al fichero nds).
- **music:** ficheros de música (mod, s3m, xm o it) y efectos de sonido (wav).
- **efsroot:** ficheros accesibles desde EFS.

En mi proyecto, tengo los directorios source y gfx. Además de los directorios, también los siguientes ficheros (para C):

- **Makefile:** instrucciones de compilación y obligatorio.
- **NombreProyecto.layout:** necesario para Code::Blocks.
- **source/main.c** o los que contenga: contiene el programa.
- **logo.bmp:** logo de la aplicación, simplemente una forma más de personalizar la aplicación, pero no es obligatorio. Por defecto muestra el logo de PAlib.

Si fuera con C++:

- **source/main.cpp:** el punto de entrada al programa, generalmente no hace falta modificarlo.
- **source/MyApp.cpp:** el código fuente de la aplicación.
- **source/MyApp.h:** definición de la clase aplicación.

Por defecto en PAlib, el main.c requerido siempre será de ésta forma:

```
#include <PA9.h>

int main(){

     PA_Init(); // Iniciar PAlib

     PA_InitVBL();

     // bucle infinito para que el programa funcione
     while(true)
```

```
	{
		PA_WaitForVBL(); // Espera al siguiente frame (60 frames por segundo)
	}
}
```

Cuando se implementa cualquier código para Nintendo DS es muy importante la función *PA_WaitForVBL()* en PAlib o *swiWaitForVBlank()* en libnds, ya que si se olvida ponerla cuando se debe, puede que el resultado no sea el esperado. Sirve principalmente para sincronizar la consola a la hora de interactuar con ella (mediante el pad o el stylus) cuando se usen las funciones requeridas para lograr tal interacción.

4.2 Detalles C/C++ para la DS

Libnds (por lo tanto también PAlib) usa sinónimos para referirnos a los tipos primitivos de toda la vida en sus funciones.

	Unsigned (sin signo)	**Signed (con signo)**
char	u8, byte	s8
short	u16	s16
int, long	u32	s32
long long	u64	s64

Ejemplos aplicados a algunas funciones

- void PA_SetDSLBrightness(u8 level): regula la luminosidad de la consola en 4 niveles posibles (*level* 0-3), aunque esta función funciona con todos los niveles en la DS Lite. En la Nintendo DS original sólo se consigue apagar o encender la luz de la pantalla.
- void PA_SetScreenLight(u8 screen, u8 light): hace lo mismo que la función anterior pero solo es encendido o apagado (1-0) por medio del parámetro *light.* Pero con la ventaja de que con *screen* se le puede indicar a que pantalla se le quiere realizar la función de la luminosidad. Ésta función se ha usado en la aplicación, pues es buena para ahorrar batería si ya existe luz necesaria en el entorno.
- *void PA_SetVideoMode*(u8 screen, u8 mode): cambia el modo de video.
- *void_PA_OutputText*(u8 screen, u16 x, u16 y, char *text,...): salida de texto.

Trabajar en C++ tiene sus ventajas pero hay que tener cuidado al usar la librería estándar de C++ en la DS por los STL, las excepciones, los streams, las sobrecargas de memoria... porque incrementa mucho el uso de memoria.

4.3 Salida de texto

El texto en la aplicación es muy importante y útil para explicar al usuario lo que debe hacer en cada momento, claro que, cuanto menos texto haya, mejor.

Con PAlib, la salida de texto es tarea fácil y dispone de sencillas funciones. Para mostrar texto en cualquier pantalla, primero se debe cargar en el sistema mediante la función:

- void *PA_LoadDefaultText(u8 screen, u8 bg_select);*

screen: debe ser 0 o 1 (pantalla inferior o pantalla superior respectivamente).
bg_select: fondo donde irá el texto. 0..3, el texto va normalmente en el fondo 0.

Ahora ya se puede escribir texto en la pantalla que se le haya pasado como argumento "screen" mediante la siguiente función, que además devuelve el número de caracteres impresos que contiene la cadena del texto:

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

screen: 0,1
x, y: coordenadas, en tiles (0-31) y (0-23), respectivamente.
text: texto ASCIIZ.

Una de las ventajas que tiene PAlib es que se puede escribir el texto en las coordenadas que el programador quiera. Con esto hay que tener cuidado porque si se escriben dos salidas de texto distintas en las mismas coordenadas se sobrescribe una con la otra. En la **figura 12** siguiente se ve la distribución en tiles y en coordenadas en la pantalla:

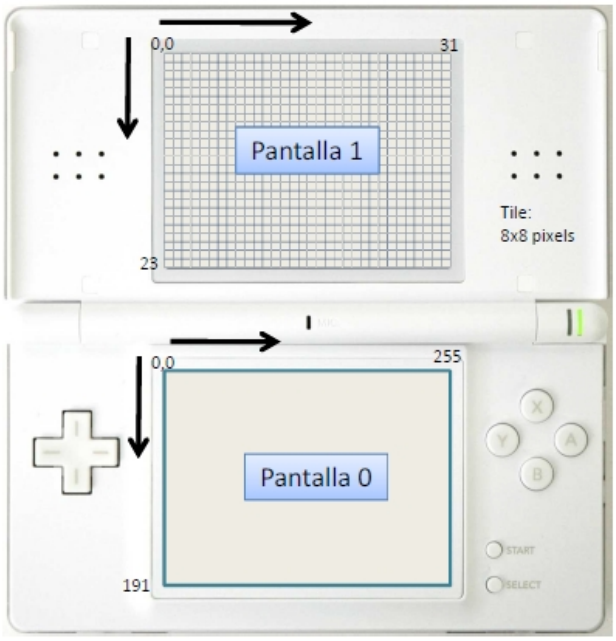

Figura 12: Pantalla superior (1), muestra tiles; Pantalla inferior (0), muestra coordenadas

En PAlib también existe su printf particular para mostrar otros tipos de datos, con un gran parecido a la función anterior, solo que no devuelve nada:

- void PA_OutputText(u8 screen, u16 x, u16 y, char *text, ...);

Para borrar todo el texto en una pantalla se utiliza *PA_ClearTextBg(u8 screen);*.

Ejemplo completo de texto

```
#include <PA9.h>

int main()
{
	PA_Init();
		PA_InitVBL();

	PA_LoadDefaultText(1, 0);	// Carga texto en la pantalla superior
	PA_LoadDefaultText(0, 0);		// y en la inferior de la consola

	int variable= 15;

	PA_OutputSimpleText(1, 0, 0, "Hola Mundo");
	PA_OutputText(0, 0, 7, "variable es igual a %d", variable);

	while (1)
	{
		PA_WaitForVBL();
	}

	return 0;
}
```

La **figura 13** muestra la ejecución de éste ejemplo implementado.

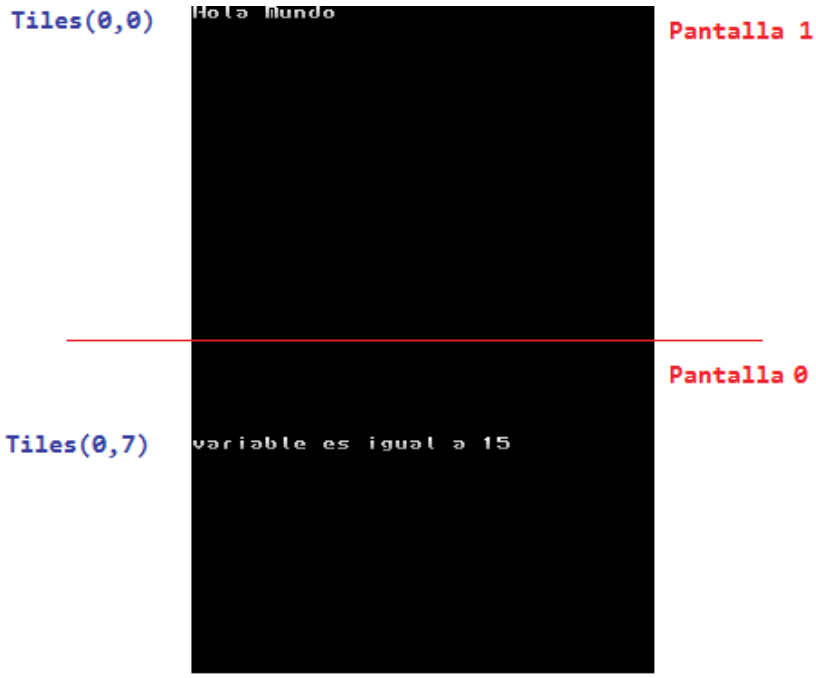

Figura 13: Ejecución del ejemplo de texto

Esto es lo mínimo que hay que saber sobre texto, pero todavía hay muchas más funciones para enriquecer el uso del texto, siendo una de ellas los colores. Nintendo DS tiene una profundidad de color de 262.144 colores (18 bits, 6 bits por canal) y es posible cambiar el color del texto. El cambio de color esta muy aprovechado en mi aplicación, porque gracias a ello he logrado jugar con los colores para remarcar el texto de otro color distinto cuando el usuario lo toca con el lápiz táctil (esto se utiliza para la elección de trabajos sobretodo (**figura 15**), entre otras cosas). La función de la que hablo es *PA_SetTextTileCol(u8 screen, u8 color);* donde color puede ser del 0..9 mostrados en la **figura 14**.

Figura 14: Colores disponibles

Nota: el orden de los colores es el orden de los números, empezando por el 0 (blanco) y terminando por el 9 (negro), éste último no se ve a causa de que el fondo ya es negro. TEXT_{WHITE, RED, GREEN, BLUE, MAGENTA, CYAN, YELLOW, LGRAY, DGRAY, BLACK},

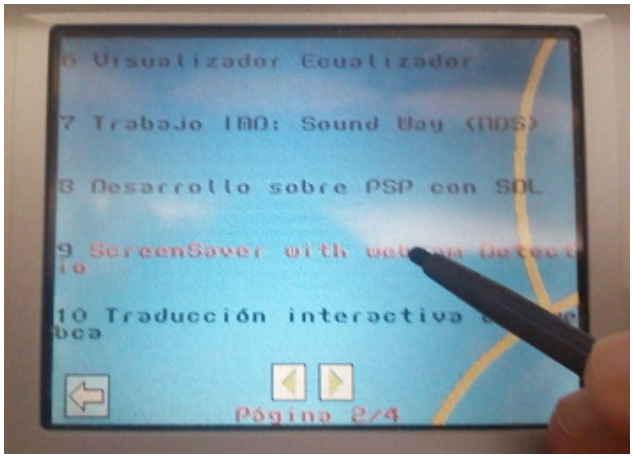

Figura 15: Elección del trabajo 9 de mi aplicación

PAlib también permite usar fuentes personalizadas. Habría que crear una fuente organizándola en una rejilla y guardándola en un bitmap. Cada carácter ocupa un tile (8x8 píxeles), y los 32 primeros son especiales. Yo he preferido usar la fuente que viene por defecto de PAlib porque la veo sencilla y legible de cara al usuario.

4.4 Dispositivos de entrada

Es muy importante saber manejar los dispositivos de entrada que ofrece la Nintendo DS para poder interactuar con ella, ya sea con los botones, la pantalla táctil, el micrófono o incluso otros dispositivos conectados a la ranura (Slot 2) como expansión, sin olvidar la cámara que contiene la Nintendo DSi.

Gracias a las libnds o las PAlib, son muchas las posibilidades que proporcionan para tratar estos dispositivos de entrada. Aunque, quizás, lo que peor está de tratar es el micrófono, ya que ambas librerías vienen cada una con un solo ejemplo de micrófono, en libnds se encuentra en *C:\devkitPro\examples\nds\audio\micrecorden* y en las PAlib en *C:\devkitPro\PALib\examples\Sound\Microphone.*

Más importante para la aplicación que he realizado son las utilidades de la botonera y sobretodo la pantalla táctil. La definición de sus parámetros se puede encontrar en *PA_Keys.h* de la librería PAlib. A continuación paso a explicar las estructuras de ambas utilidades donde se podrá comprobar ciertas similitudes a la hora de programar, falicitando ésta tarea.

Botonera

Para el acceso a la botonera de Nintendo DS en PAlib, se usa la estructura "**Pad**", que se actualiza en cada frame. La estructura completa sería: **Pad + Evento + Botón**

Figura 16: Combinaciones posibles para el pad

"Pad" es obligatorio para indicar que se va a usar la botonera. Luego, viene seguido de un evento, siendo por ejemplo el primero en explicar el llamado **Newpress,** cuya funcionalidad es esperar a que el botón sea simplemente pulsado. Con **Held** al pulsar el botón y mantenerlo presionado hará el evento que deba hacer hasta que sea soltado y finalmente con **Released** el evento será una vez soltemos el botón ya pulsado previamente. Después del evento se le asigna el botón a cumplir tal evento, mostrados en la **figura 16** todas las combinaciones existentes.
Newpress y Released solo están activos un frame, mientras que Held puede estar el tiempo que se requiera. Ejemplo básico:

```
#include <PA9.h>

int main(){

     PA_Init();
     PA_InitVBL();
           PA_LoadDefaultText(1, 0);  // se carga el texto

          while(true)
          {
                if(Pad.Newpress.A)
                   PA_OutputSimpleText(1, 0, 0, "Has presionado el
botón A");

                if(Pad.Held.Right)
                 PA_OutputSimpleText(1, 0, 2, "Estás aguantando el
cruceta hacia la derecha");
           PA_ClearTextBg(1); //Mientras no haya nada pulsado

           PA_WaitForVBL();
     }

          return 0;
}
```

En la aplicación, el mayor uso de la botonera es de una forma muy intuitiva y simple. Para avanzar hacia delante pulsando A, para volver o cancelar el botón B (Son estándares en los videojuegos de Nintendo) y la cruceta (más conocido como pad direccional) para cambiar de página o cambiar de aspecto a la hora de evaluarlos. Ejemplo: *Pad.Newpress.Down*

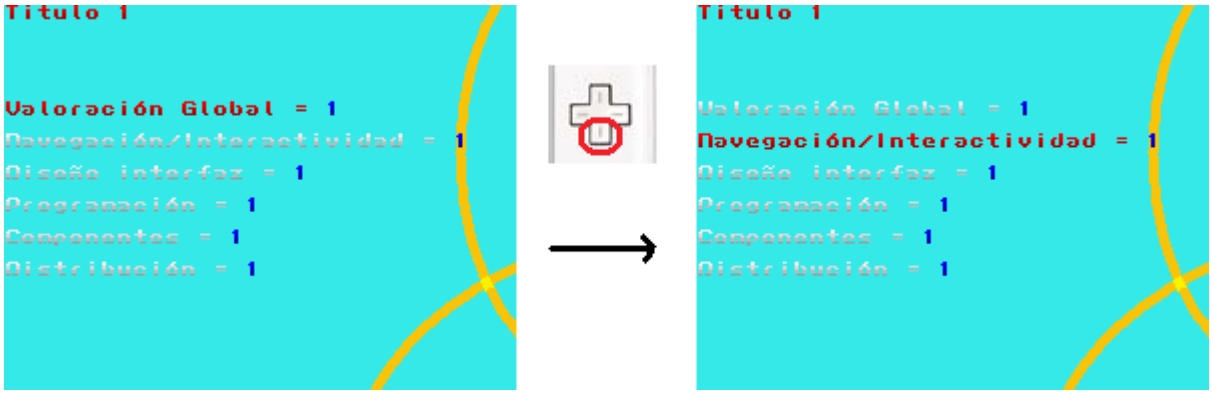

Figura 17: Ejemplo en la aplicación al pulsar la cruceta hacia abajo.

Este cambio también funciona presionando un sprite que está en la pantalla inferior pero esto se explicará en el **apartado 5.1**.

Stylus

Más cómodo e intuitivo para el usuario es usar el lápiz en la pantalla táctil, gracias a las instrucciones de texto y los botones (sprites). La estructura está compuesta por la palabra "**Stylus"** seguido de un evento (las mismas que con la botonera) y también se actualiza en cada frame: **Stylus + Evento (Newpress, Held, Released).**

Y por otra parte, también se puede saber mediante "Stylus", que coordenadas se estántocando o se deben de tocar para que algo pase. El origen es desde la esquina superior izquierda hasta la inferior derecha de la pantalla táctil (0-255, 0-191), y devuelve las coordenadas tocadas en ese momento. Esto sería : **Stylus + Coordenada (X ó Y).**

Un ejemplo podría ser parecido al que he escrito arriba sobre la botonera:

```
#include <PA9.h>

int main(){

     PA_Init();
     PA_InitVBL();
           PA_LoadDefaultText(0, 0);   // se carga el texto

     while(true)
          {
               if(Stylus.Held)
                   PA_Print(0, "Stylus Position : %d, %d    \n",
Stylus.X, Stylus.Y);
```

```
                PA_WaitForVBL();
        }

            return 0;
}
```

En la aplicación he hecho uso de estas dos cualidades. Se puede navegar por toda la aplicación solamente con el lápiz táctil, sin necesidad de usar la botonera, esto sería tocando los sprites, escribir el comentario con el teclado o el reconocimiento, y la elección de un trabajo (excepto en el uso del cursor del reconocimiento de caracteres, que se debe de mover hacia la izquierda o derecha con la cruceta para desplazarlo de forma obligada).

Para la elección de un trabajo se han usado las coordenadas X e Y, creando unas **áreas** (5 en total) que será donde estén escritos los títulos de los trabajos. Cuando un trabajo es tocado sin quitar el stylus (Held) cambia de color de negro a rojo, y cuando soltamos el stylus, la aplicación entenderá que será la elección del trabajo que desea el usuario. Eso si, éstas áreas están fijas, ya que los títulos de los trabajos son dinámicos y no se puede saber con anterioridad la longitud que van a ocupar en pantalla. En cambio para la elección de teclado o reconocimiento de caracteres (**figura 18**), entre otras, si que son áreas con la misma longitud que el texto.

Figura 18: Área para la elección del teclado de la aplicación

Muestro un trozo de código de como está implementada la primera área del título que esté en la posición más arriba del menú:

```
if(Stylus.X>1 && Stylus.X<254 && Stylus.Y>5 && Stylus.Y<14)
{
              PA_SetTextTileCol(0,1); //cambia el color del texto a
rojo en la pantalla inferior
                *opcion=0+posicion;         //calcula que número de
trabajo se ha elegido
              marcaTitulo(*opcion,2,1);
              if(!Stylus.Held) { menu=5; PA_WaitForVBL(); }

}
```

Nota: la función "*marcaTitulo(int opcion,int x, int y)*" se encarga de imprimir el título seleccionado de color rojo, en el área que ha sido tocada.

En la **figura 19** he señalado cómo son de larga y anchas las áreas, y dónde están. De ancha es toda la pantalla (siempre: *Stylus.X>1 && Stylus.X<254*) y de larga un poco más de lo que ocupa un carácter (sobre 9 píxeles: *Stylus.Y>5 && Stylus.Y<14*) siendo lo que varía en cada área.

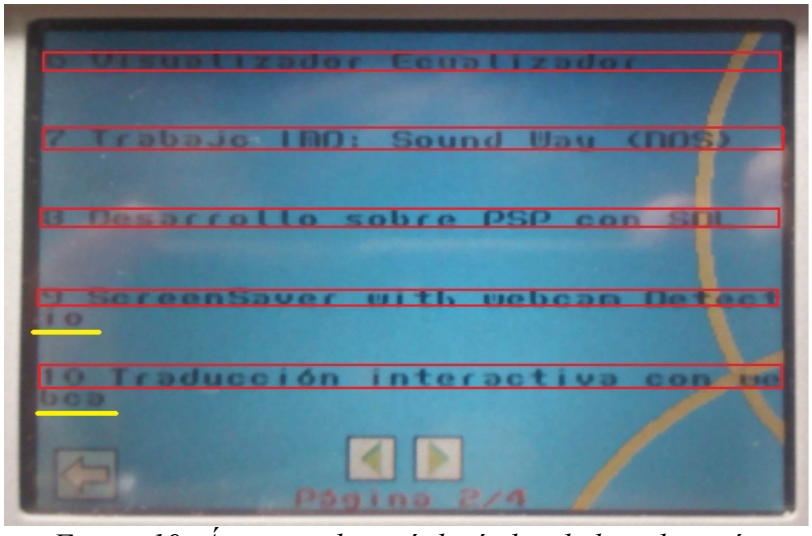

Figura 19: Áreas en el menú de títulos de la aplicación

El área marcada en zona superior de la pantalla (donde se sitúa el trabajo 6 "Visualizador Ecualizador"), pertenece al trozo de código del ejemplo anterior, y también he subrayado en amarillo los títulos más largos que el área definido.

Existen otras posibilidades para jugar con el stylus: gracias a un booleano, se puede saber si se ha realizado "doble clic" (dos golpes seguidos con el lápiz) mediante **DblClick**; con **Downtime** se sabe el tiempo (en VBLs) que tiene contacto el puntero con la pantalla; por el contrario con **Uptime**, que es el tiempo que no hay contacto; por último está **Pressure** para jugar con la presión con que se toca la pantalla (0 sin contacto, alrededor de 10 cuando hay). Todo esto siguiendo la estructura "Stylus".

Teclado virtual

Otro dispositivo de entrada que incorpora libnds y PAlib es un teclado para escribir texto (**figura 20**). Es muy básico, no implementa eco, ni entrada con buffer, ni fin de entrada. Ejemplo de PAlib en *C:\devkitPro\PALib\examples\Input\Keyboard\KeyboardCustom*. El teclado se carga mediante una función:

- *void PA_LoadDefaultKeyboard(u8 bg_number);*

bg_number: número del fondo donde cargar el teclado.

Figura 20: Teclado virtual PAlib

Y para devolver la tecla que pulsa el usuario en el teclado es mediante:

- *char PA_CheckKeyboard(void)*;

que además, distingue entre minúsculas y mayúsculas.
Para sacar por pantalla el teclado y poder usarlo como tal, hace falta la función:

- *void PA_KeyboardIn(s16 x, s16 y)*;

x,y: coordenadas para desplazar el teclado de abajo hasta la posición indicada.

Para mover el teclado por la pantalla sería con *void PA_ScrollKeyboardXY(s16 x, s16 y), que es parecido a la anterior solo que el teclado ya debe estar en pantalla. Y finalmente para quitar el teclado de la pantalla mediante un barrido hacia abajo es con la función PA_KeyboardOut().*

Nota: la imagen del teclado llamado **keyboardcustom.png** se debe incorporar en el directorio gfx, junto a todos los sprites.

El teclado que uso en mi aplicación es el mismo que el del ejemplo de PAlib (también era el que usaba en la versión anterior), pero con una novedad en la implementación, ya que era uno de los objetivos a mejorar. Se ha añadido un **cursor** para darle al usuario la oportunidad y total flexibilidad de rectificar en cualquier momento el texto sin tener que borrar todo el comentario ya escrito. El cursor es una “línea vertical roja” situada justo debajo del del carácter a escribir y se mueve, o bien mediante la cruceta (izquierda/derecha) o bien tocando los sprites correspondientes (**figura 21**).

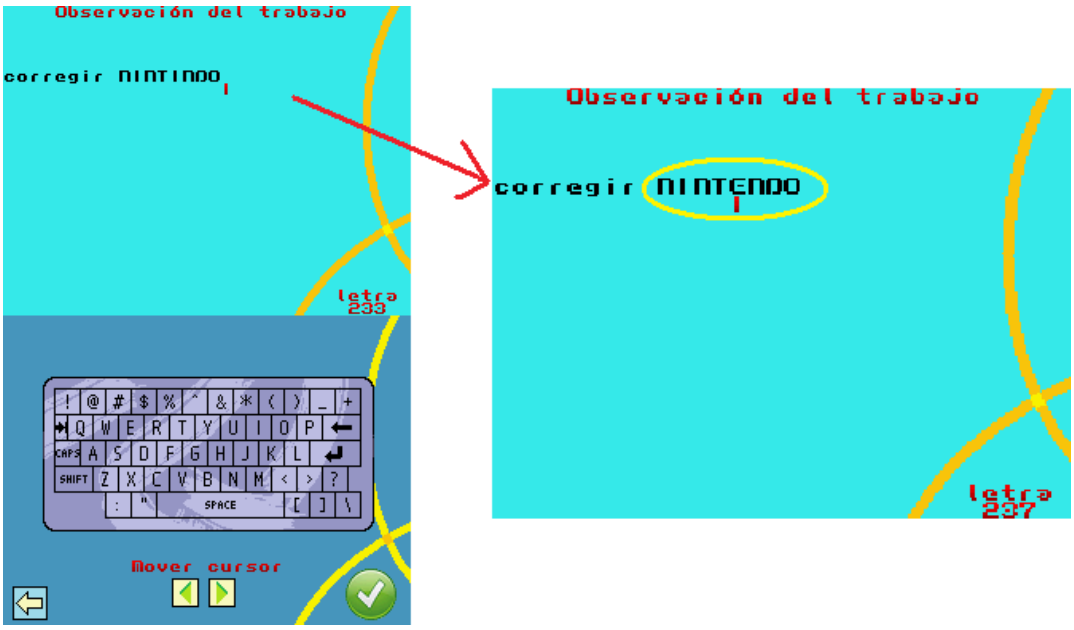

Figura 21: Movimiento del cursor hacia la izquierda para corregir NINTINDO a NINTENDO

Se puede observar en la pantalla superior, en la esquina inferior derecha, que se indica el **número de caracteres** que hay disponible para realizar la observación del trabajo (coincide con la capacidad del array "text[250]" donde se guardan los caracteres, cuando 250 llega a 0 salta al principio del texto). Y también se ha quitado la opción de hacer un salto de línea (tecla "enter"), por el motivo de que el desplazamiento del cursor se dificulta dando errores moviéndose por donde no debe.

Reconocimiento de caracteres

La otra opción que se ofrece para escribir el comentario de texto en la aplicación es mediante reconocimiento de caracteres usando la pantalla táctil. PAlib viene con un ejemplo ya implementado *C:\devkitPro\PALib\examples\Input\Stylus\RecoGraffiti.*
Es similar al sistema de entrada de **Palm** ("Graffiti"), que es el software de reconocimiento de escritura manuscrita utilizado en el sistema operativo Palm OS para PDAs. Se basa en una neografía (desarrollo de la escritura) en mayúsculas de los caracteres que se pueden trazar con el stylus sobre la pantalla, mostrando en blanco (sobre fondo negro) el trazado realizado. Son 4 las letras que han sido simplificadas por la dificultad que conllevan realizar su trazado sin tener que despegar el stylus de la pantalla: **A F K** y **T.**

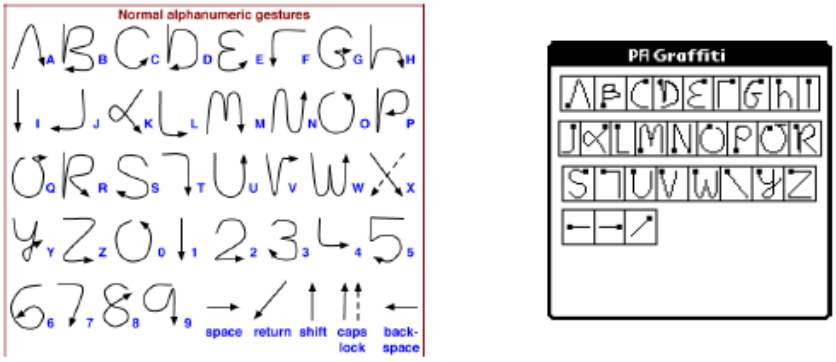

Figura 22: A la izquierda Palm Graffiti, a la derecha PA Graffiti

La implementación que viene con PAlib tiene varios inconvenientes en comparación al teclado, pues lo primero que se observa es que ésta versión no reconoce números ni ningún símbolo, únicamente letras en minúsculas, poner punto final, espacio en blanco, borrar y salto de línea. También se ha implementado un cursor para el reconocimiento (aunque solo funciona con la cruceta, ya que con sprites daba problemas al necesitar la pantalla inferior para escribir las letras, explicado un poco más detallado más adelante) y por otra parte que funcione sin que ocurra otro problema se le ha eliminado la opción de realizar un salto de línea en el código fuente, pues el cursor no se coloca donde toca al saltar de línea y también al borrar caracteres tampoco estaba donde debería, siendo eliminado el siguiente trozo de código:

```
if (letter == '\n')
{     // Enter pressed
      text[nletter] = letter;
      nletter++;
}
```

Como comentaba en el párrafo anterior, el reconocimiento de caracteres de PAlib viene muy corto de caracteres predefinidos, pero gracias al uso de un par de funciones en el código de la aplicación todo esto pudo resolverse notablemente.

- *void PA_UsePAGraffiti(u8 use);* cuya labor es activar (mediante use=1) o desactivar (con use=0) las formas predefinidas por el “graffiti”.
- *void PA_RecoAddShape(char letter, char* shape);* sirve para añadir los caracteres propios que se deseen crear realizando el trazo de manera personalizada para después ser reconocido en la escritura.
- *void PA_RecoShape(void);* espera a que se escriba en pantalla un carácter creado con la función anterior.

En PAlib, se puede encontrar dos ejemplos que use estas funciones en el directorio *C:\devkitPro\PALib\examples\Input\Stylus\RecoAddShape* y en *C:\devkitPro\PALib\examples\Input\Stylus\RecoShapeInfo.* Gracias a cualquier de estos ejemplos, al dibujar un trazo cualquiera con el stylus, muestra por pantalla una serie de caracteres (entre letras, números y símbolos) que representa aproximadamente la

longitud y la forma del trazo, ejemplo mostrado de la creación del carácter de la "**ñ**" en la **figura 23**. En la pantalla inferior se ha dibujado con el stylus el nuevo carácter (se ha intentado hacer lo más parecido a una "**ñ**" sabiendo que no se puede despegar el stylus de la pantalla al hacer el trazo) y en la superior marcado dentro del rectángulo amarillo podemos ver la secuencia generada del trazo, en este caso "**HHHF;;<=EHHHL00**" y de su longitud en píxeles, "**421**" en este ejemplo.

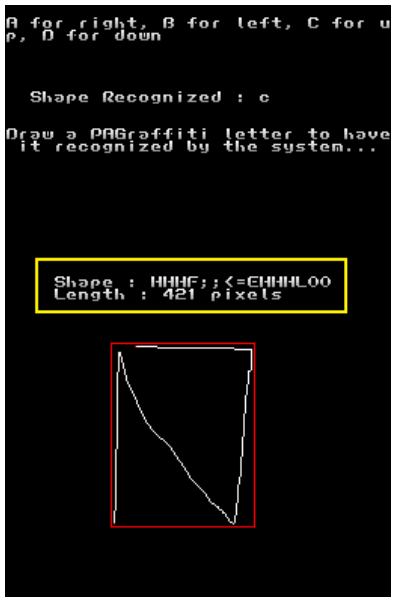

Figura 23: Trazo y secuencia del carácter a crear

Entonces, para crear tal carácter se haría de la siguiente forma:

- *PA_RecoAddShape('ñ', "*HHHF;;<=EHHHL00*");*

En conclusión, se han añadido al reconocimiento de caracteres los números, las vocales acentuadas, la "ñ", la coma y el signo de interrogación. Todo esto se puede encontrar en una función llamada *void Rcnuevos()* con sus respectivas secuencias de trazos en **valoracion.c.** Durante la observación usando el reconocimiento de caracteres se le muestra al usuario el cómo se hacen todos los trazos predefinidos y los nuevos también (**figura 24**), pero como todos no caben en la pantalla de la consola, al pulsar el botón "select" se cambia la imagen para mostrarlos**.** Las vocales acentuadas se escriben como si se escribiesen las vocales en minúsculas.

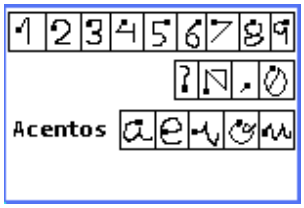

Figura 24: Información de los trazos nuevos

Por supuesto no podrían faltar las mayúsculas, pero aquí se ha hecho con otra idea a la anterior y es que hay un inconveniente en crear tantos caracteres. Si se crean muchos caracteres nuevos será más complicado para que el programa entienda a la primera qué carácter se ha escrito, por lo que es importante hacer trazos lo más diferentes posibles entre ellos, además de fáciles e intuitivos para el usuario. Entonces, si hay 26 letras (sin contar la "ñ") ¿sería buena idea tener ya 26 trazos para las minúsculas (que son las predefinidas) y realizar otros 26 trazos nuevos para las mismas letras pero para escribirlas en mayúsculas, sumando un total de 52 trazos? (todo esto sin haber contado los nuevos caracteres creados con la función *PA_RecoAddShape*). - No, ya que dificultaría mucho la usabilidad del reconocimiento de caracteres y perderíamos mucho tiempo en escribir una simple frase. Evitar esto sería importante y facilitaría mucho la escritura. Por ejemplo, da la casualidad de que la letra "t" y los números "1" y "7", se dibujan con trazos parecidos causando el problema principal de la pregunta que he realizado, es decir, el programa puede no escribir el carácter deseado a la primera por semejanza a otros.

Con la función *PA_UsePAGraffiti(u8 use);* se puede hacer de una manera muy sencilla y ahorrarse esos 26 trazos de más. En la implementación de la aplicación se ha creado una función llamada *reconocimientoMayusculas()* en **valoracion.c** donde con use=0 se desactiva el "graffiti" y se definen las letras en mayúsculas con el mismo trazo que las minúsculas, llevando a una reutilización y ahorro importante de caracteres. Para activar o desactivar las mayúsculas, sólo hay que pulsar el botón "start" de la botonera, puas al activar de nuevo el "graffiti" los trazos de las minúsculas mantienen su prioridad.

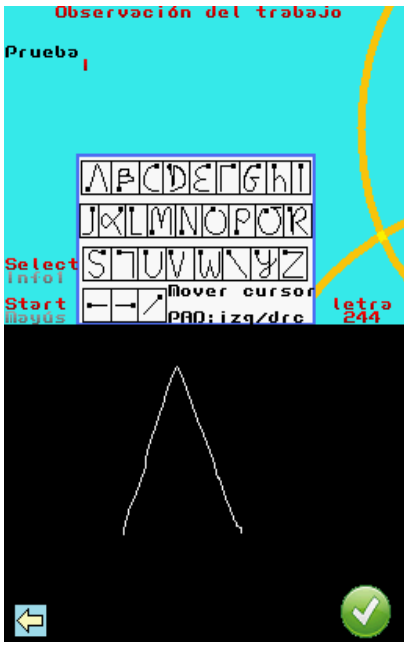

Figura 25: Reconocimiento de caracteres dentro de la aplicación

En el ejemplo de la **figura 25** se ve como la última letra es una 'a' de la palabra prueba, y en la pantalla inferior está dibujada esta última letra, correspondiendo al Graffiti.

Si no se han colocado los sprites para el cursor, ni para otra tarea como la del cambio de minúsculas a mayúsculas, es por el hecho de que cuando se toca el sprite para mover el cursor o cambiar de minúsculas a mayúsculas el reconocimiento entiende que se dibuja el carácter de un punto "." y lo imprime.

En la aplicación, no hay ningún problema si el usuario quiere cambiar de repente de teclado a reconocimiento o viceversa, ya que el comentario que haya escrito no se eliminará al volver hacia atrás y el cursor apuntará donde debe también (esto se debe a que son variables globales).

4.5 Logo

Todo juego y aplicación tiene un logo representativo, que se observa en el cuadro remarcado de rojo de la **figura 26** cuando vamos a elegir el juego en le menú o en el menú de la flashcard que se use.

Figura 26: Ubicación del logo

Cuando se realiza "homebrew", este logo se puede diseñar. Por ejemplo, en PAlib, si no diseñamos un logo para una aplicación, el *Makefile* obtendrá el logo de PAlib por defecto, que se encuentra en *C:\devkitPro\PALib\lib,* llamado **defaultlogo.bpm** (**figura 27**).

Si se piensa crear un logo para la aplicación, éste debe de estar en la carpeta del proyecto junto al *Makefile*. Debe de tener unas características, que son las siguientes:

- Formato BMP 32x32 píxeles.
- Indexado de 256 colores, pero con hasta 16 colores.
- El primer color de la paleta define el color transparente.
- Copiarlo como logo.bmp junto al Makefile

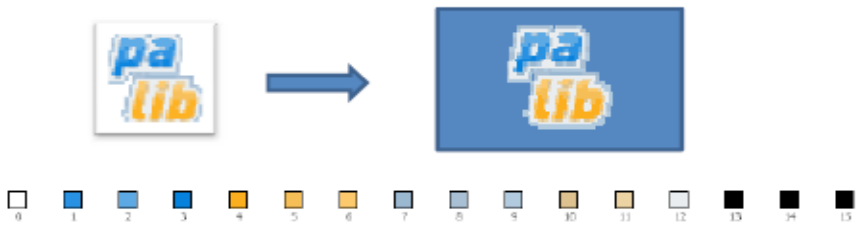

Figura 27: Paleta del logo de PAlib

Como el nombre para la aplicación se llama **Evaluación DS**, el logo se ha hecho con las tres iniciales EDS haciéndole referencia de forma sencilla (**figura 28**). Cuatro son los colores que contiene la paleta de este icono, más el transparente que sería el blanco. Con cualquier programa de edición de imagen se puede diseñar.

Figura 28: Logo de la aplicación

5 Recursos gráficos

Una de las notables mejoras realizada en la aplicación ha sido la incorporación de interfaz gráfica, añadiendo botones y fondos para facilitar al usuario, sobretodo, la usabilidad. Gracias a PAlib, y la herramienta que incorpora de **PAGfx** (**figura 29**), la creación de interfaz gráfica se convierte en una tarea llevadera. Además, permite diseñar los gráficos de manera personalizada y con un toque original sin tener que recurrir a fuentes externas.
También es muy importante disponer de herramientas de edición de imágenes como paint, gimp, photoshop... para el diseño de sprites y fondos o para su modificación. Yo personalmente me he apoyado usando **paint** y **photoshop** para crear mis propias imágenes.

PAGfx

Aquí explicaré cómo funciona ésta aplicación para preparar los **sprites** y los **fondos** antes de incorporarlos a la aplicación.

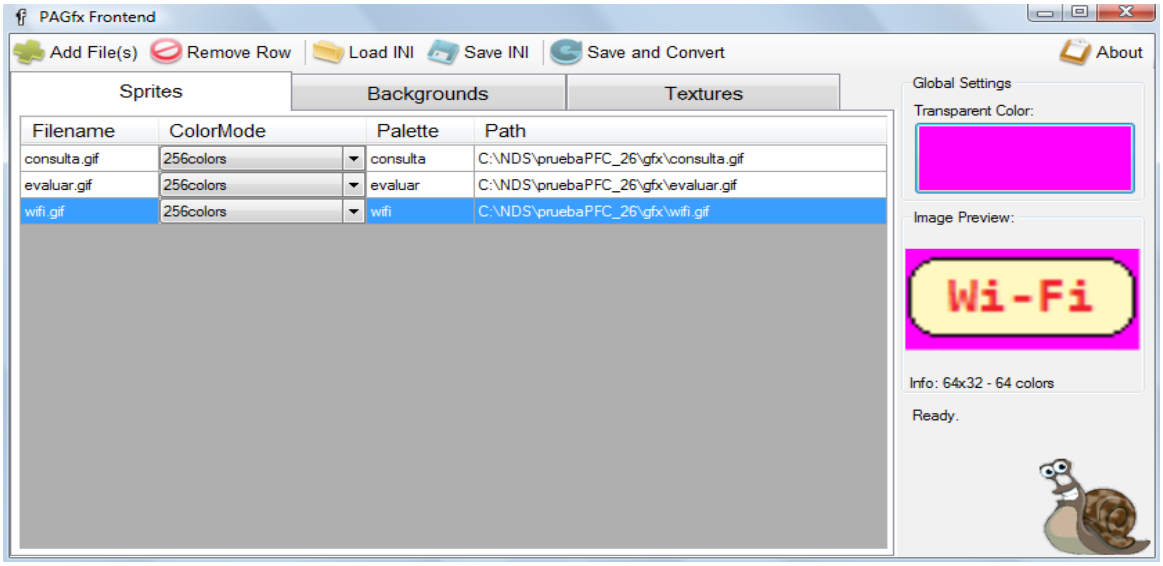

Figura 29: Vista de PAGfx

PAGfx genera para cada tipo:

- **Sprite:** gráficos (*_Sprite.bin) y paleta (*_Pal.bin)
- **Fondos:** mapas (*_Map.bin), tiles (*_Tiles.bin), paletas (*_Pal.bin) e información (*.c)
- **Texturas** (aunque no trabajo con ellas): textura (*_Texture.bin) y paleta (*_Pal.bin)

Para todos genera *all_gfx.h (*que se debe incluir en el programa), un documento de texto llamado *PAGfx* y un archivo de opciones de configuración también llamado *PAGfx*. Se puede establecer el color transparente que se requiera (por defecto, será el magenta, como vemos en la imagen de arriba). El magenta está formado por la mezcla de Rojo=255, Verde=0, Azul= 255.

Para cada recurso es importante elegir el tipo de conversión a realizar.

- Para los sprites, su tamaño debe ser múltiplo de 8, normalmente son de un máximo de 256 colores. Para asegurar que sea de 8 bits, el formato que he usado para mis sprites y mis fondos es de tipo GIF, ya que así me aseguro un máximo de 256 sin preocuparme de pasarme de colores.
  Además de todo esto, se puede establecer el nombre de la paleta a utilizar (que se puede compartir con otros sprites).
- Para fondos, su dimensión horizontal debe ser múltiplo de 256 y la vertical de 192, que son las medidas máximas para ambas pantallas de la Nintendo DS. Tipos disponibles: 8 bits, 16 bits, TileBg, LargeMap, InfiniteMap, RobBg y EasyBg.
- Para texturas, sus tipos solo pueden ser de 8 y 16 bits.

Entonces, una vez realizada la conversión de los sprites y los fondos, el directorio se vería parecido a la imagen que muestra la **figura 30**:

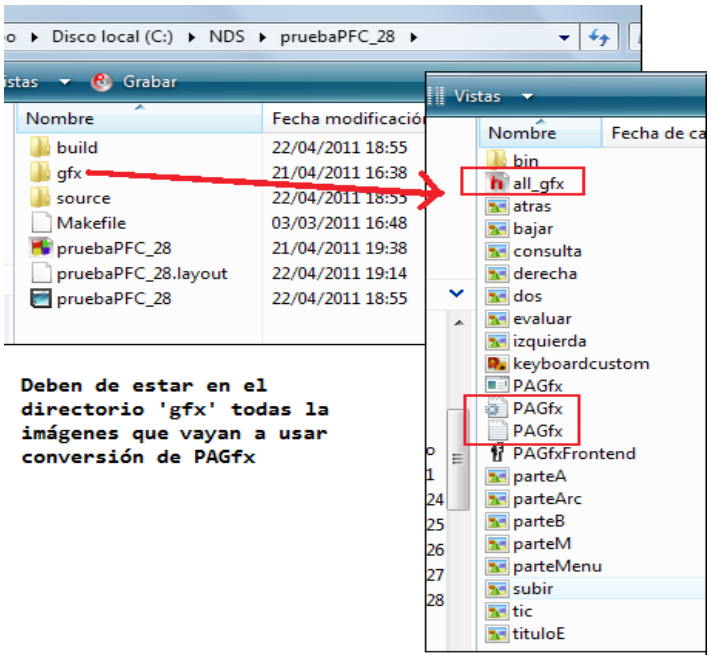

Figura 30: Directorio de gfx

5.1 Sprites

Los botones gráficos incorporados son **sprites**, ya que con PAlib existen funciones que permiten la interacción al tocarlos en la pantalla táctil, lo que lo convierten en algo muy útil. Un sprite es un tipo de mapa de bits dibujado en la pantalla especializado sin cálculos adicionales de la CPU. Suele ser pequeño y puede asumir otras formas distintas a las de un rectángulo asignando un solo color como transparente (canal alfa).Generalmente son usados en video juegos para crear los gráficos de los personajes y sus animaciones.

Las características de los sprites en Nintendo DS son las siguientes:

- 128 sprites por pantallas.
- Se pueden mover por la pantalla sin dificultad.
- Se pueden animar (cambiando la imagen mostradas).
- Se puede hacer parcialmente transparente.
- Se pueden rotar y escalar.
- Sus posiciones permitidas son desde las coordenadas 0..511, 0..255 (a partir de ahí, aparece por la izquierda/arriba).
- Dispone de 3 modos de color: Paleta de 16 colores (en GBA) y 32 bytes por tile (8*8*4 bits= 256 bits=32 bytes); paleta de 256, aunque necesitan el doble de memoria que el anterior; 16 bits (no usan paleta), y usan mucha memoria (8*8*2 = 128 bytes por tile). Se suele usar la paleta de 256 bytes.
- No se puede ver un sprite debajo de otro transparente.

En cuanto al tamaño del sprite, el hardware de la consola lo limita a los siguientes combinaciones de tamaños posibles:

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

Funciones para cargar y tratar un sprite

Para cargar un sprite se realiza mediante dos funciones, la primera:

- *PA_LoadSpritePal(u8 screen, u8 palette_id, void* palette);*

carga la paleta del sprite en la pantalla seleccionada y en la paleta indicada:
screen: 0,1
palette_id: 0..15 (siendo el más prioritario el 0, y el menor el 15)
palette: es un puntero a la paleta.

La segunda función crea y coloca el sprite en pantalla:

- *PA_CreateSprite(u8 creen, u8 obj_number, void* obj_data, u8 obj_shape, u8 obj_size, u8 color_mode, u8 palette, s16 x, s16 y);*

Screen: 0,1
obj_number: número de sprite (0..127).
obj_data: imagen del sprite
obj_shape, obj_size: tamaño del sprite. Usar siempre la constante OBJ_SIZE_<W>X<H>, H=8,16,32, 64
color_mode: 0..16 colores, 1:256 colores.
palette: id de la paleta a usar.
x, y: coordenadas de posición en pantalla, en píxeles.

Una vez en la aplicación se han creado todos los sprites, para poder interactuar con ellos habría que escribir la función *PA_InitAllSpriteDraw();* y finalmente para este ejemplo en concreto, el identificador de la paleta es el número 11, por lo que la función para interactuar con éste sprite cuando lo tocas únicamente sería:

- void PA_SpriteTouchedPix(u8 obj_number);

Ejemplo:

```
PA_LoadSpritePal(0, 11, (void*)nombreSprite_Pal);
PA_CreateSprite(0, 11,(void*)nombreSprite_Sprite, OBJ_SIZE_32X32,1, 11, 165,10);
PA_SpriteTouchedPix(11);
```

Existen otras funciones con relación a los sprites como *PA_MoveSprite(u8 id);* que sirve como su nombre indica, para mover el sprite de posición, entre otras funciones. Hay varios ejemplos más en las librerías PAlib, C:\devkitPro\PALib\examples\Sprites.

La **figura 31** muestra cómo se verían algunos de los sprites fuera y dentro de la aplicación:

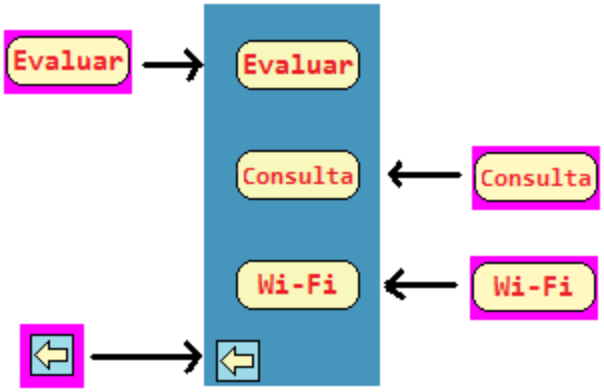

Figura 31: Sprites del menú principal de la aplicación

A veces el ir creando sprites sin parar, sobrecarga la memoria y esto dará un error, enseñando la pantalla negra y bloqueando todo (**figura 32**). Para que esto no ocurra, no basta solo con eliminar el sprite mediante la función *void PA_DeleteSprite(u8 Screen, u8 palette_id);* ya que esta función no libera la memoria, solo elimina el sprite de la pantalla. La función sería:

- void PA_ResetSpriteSys();

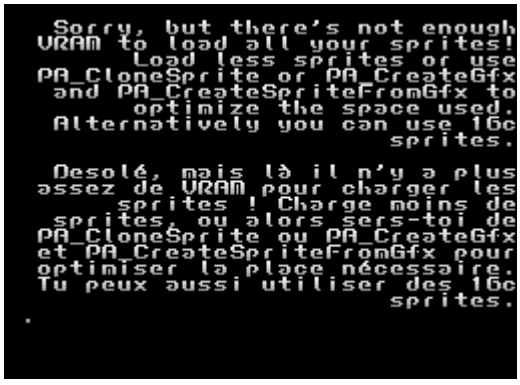

Figura 32: Error por sobrecarga de sprites

5.2 Fondos (Background)

Nintendo DS tiene, para cada pantalla, 4 fondos. Cada uno con su propia paleta. La prioridad es igual que en los sprites, el fondo 0 más prioritario y el 4 el que menos. Hay varios tipos de fondo:

- **Tiled:** compuesto de un tileset, una paleta y un tilemap, creados por PAGfx.
- **Bitmap:** de 8 a 16 bits, es el que se puede dibujar (aunque sólo puede haber uno por pantalla).
- **Rotables/escalables:** máximo dos por pantalla.

Ejemplo de fondo tiled en la **figura 33**:

Figura 33: Ejemplo de fondo tiled

El código (nombreFondo.c) que crea PAGxf cuando convierte un fondo tiled:

```
#include <PA_BgStruct.h>

extern const char nombreFondo_Tiles[];
extern const char nombreFondo_Map[];
extern const char nombreFondo_Pal[];

const PA_BgStruct nombreFondo = {
      PA_BgNormal,
      256, 192,

      nombreFondo_Tiles,
      nombreFondo_Map,
      {nombreFondo_Pal},

      3136,
      {1536}
};
```

Aunque para la versión antigua de la aplicación se han usado fondos de tipo bitmap (JPG), en la del proyecto he optado por los fondos **tiled.** La función principal que realizan los fondos en la aplicación solo es para adornar y para que el usuario se sienta cómodo. Sobre los fondos de tipo bitmap:

- **Ventajas:** Permiten dibujar directamente a la pantalla (con el puntero, texto de calidad, etc). Se puede cargar imágenes "normales" como JPG, GIF, BMP y RAW (en la versión antigua de la aplicación se trabajó con fondos en JPG). Los de 16 bits no necesitan paleta (color verdadero).
- **Desventajas:** Consumen mucha memoria (3/8 y 2/3 de la memoria disponible para fondos de una pantalla), lo que obliga continuamente a estar "vaciando" memoria con una función (como con los sprites, aunque éstos no consumen tanta memoria). Se acceden píxel a píxel, por lo que son lentos para dibujar en ellos. Sólo disponibles en el fondo 3.

Funciones para cargar fondos

Con una sola función se logra cargar y mostrar por pantalla el fondo con tiles:

- void *PA_LoadBackground(u8 Screen , u8 bg , const PA_BgStruct* bg_name);*

Screen: 0,1
bg: fondo 0..3
bg_name: puntero a la estructura que almacena el fondo, generada por PAGfx.

Ejemplo: *PA_LoadBackground(1,3,&fondo);*

Para eliminarlo también se realiza con una sola instrucción al igual que en los sprites, *void PA_DeleteBg(u8 Screen, u8 bg_select);* siendo **bg_select** el número de fondo donde se encuentra cargado.

Al igual que con los sprites, esta instrucción solo elimina el fondo de la pantalla, dejando intacta la memoria ocupada anteriormente. Para ello existe otra función muy importante para que evite el pantallazo negro similar al de los sprites (**figura 34**) y libere esa memoria de la VRAM:

- *void PA_ResetBgSysScreen(u8 screen);*

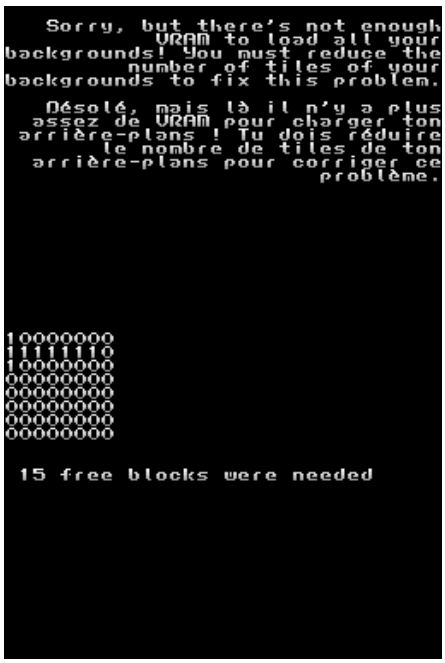

Figura 34: Error por sobrecarga de fondos

Para cargar fondos de tipo JPG, GIF... PAlib dispone de funciones especiales para cada tipo, y muchas otras funciones para realizar cosas como rotaciones, scroll, etc. Pero en la aplicación no se ha necesitado nada de todo esto.

En la **figura 35** muestro dos de los varios fondos en GIF que he diseñado personalmente para la aplicación.

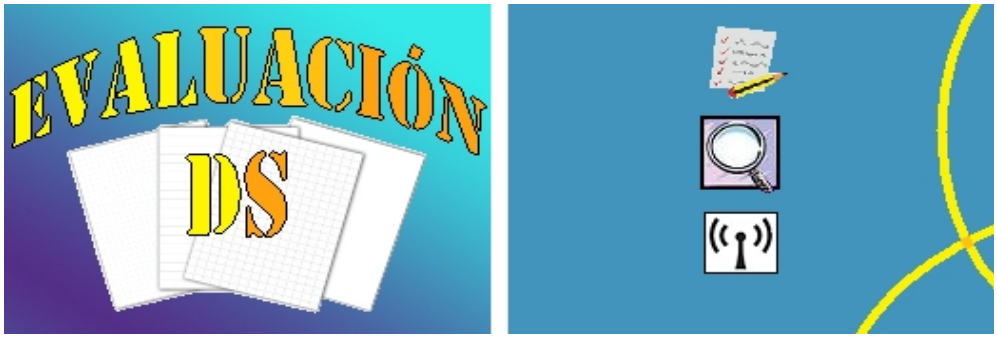

Figura 35: Fondos de la portada de la aplicación y del menú principal

6 Sistemas de ficheros en la DS

Una de las principales limitaciones de la Nintendo DS es que el programa debe ocupar, como máximo 4MB. Para resolver el problema existe la posibilidad de guardar recursos en sistemas de ficheros separados del programa para acceder a ellos bajo demanda. Todo método para hacer uso del sistema de fichero en la DS hace uso de las funciones de entrada y salida estándar de C: **fopen, fclose, fread, fwirte**... Voy a hablar de 3 formas de acceso.

6.1 Filesystem

Un ejemplo en PAlib se encuentra en C:\devkitPro\PALib\examples\Filesystem\ Filesystem, dentro del directorio “filesystem” hay que almacenar los archivos o directorios que se requiera. De ésta forma se pueden acceder a la información sin tener que almacenarla en la memoria SD ya que se encuentra dentro del archivo .nds incrustado al final de la ROM, y en el emulador (No$Gba) mostraría el contenido del fichero. He hecho la prueba incorporando al directorio “filesystem” el fichero .txt (**figura 37**) con la información de los títulos y luego ejecutar el “homebrew” desde el emulador, mostrando los títulos como sale en la **figura 36**.

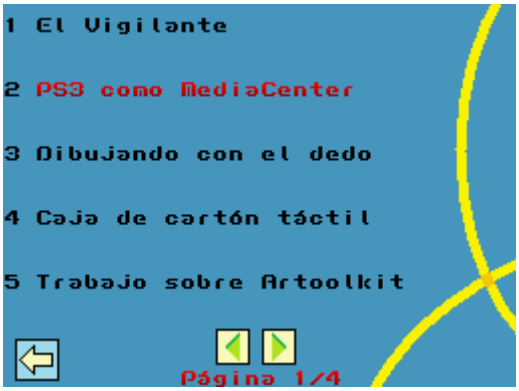

Figura 36: Menú de trabajos desde el emulador gracias a Filesystem

Esta forma de leer el fichero sólo la he usado para trabajar mediante el emulador y no perder demasiado tiempo probando la aplicación en la consola. Tiene el problema de que no se permite crear ficheros nuevos (necesario para almacenar las evaluaciones en nuevos ficheros) ni tampoco lee de la memoria SD (para las consultas de los ficheros creados).

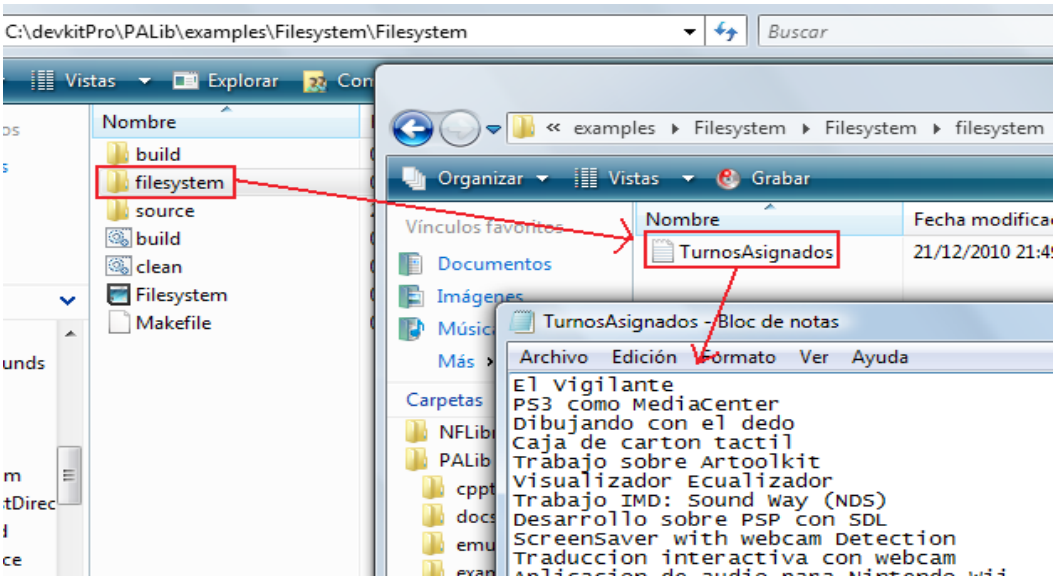

Figura 37: Directorio del ejemplo Filesystem

A la hora de usar Filesystem en la implementación basta con importar las librerías *#include <sys/stat.h> e inicializarlas* mediante la función *nitroFSInit().*

Ejemplo de código:

```
#include <PA9.h>
#include <sys/stat.h> // incluir la librería

int main(int argc, char* argv[]) {

    PA_Init();
    PA_InitVBL();
    PA_LoadDefaultText(1, 0);

    if(nitroFSInit()) // inicializa las librerías filesystem
    {
         PA_OutputSimpleText(1, 0, 0, "Filesystem init ok");

         FILE*  file  =  fopen("/TurnosAsignados.txt",  "rb");
         // abre el fichero .txt y lee su contenido

          if(file != NULL) { [...] }

    }else PA_OutputSimpleText(1, 0, 0, "Filesystem init error!");

    while(1)
    {
        PA_WaitForVBL();
    }

    return 0;
}
```

6.2 EFSLib

Su función es similar al de Filesystem pero todavía es más completa y a su vez compleja. EFS son las siglas de embedded file system library y para usar éstas librerías hay que seguir los siguientes pasos:

- Copiar los ficheros **efs_lib.c** y **efs_lib.h** al directorio source del proyecto (**figura 38**).
- Copiar el fichero **efs..exe** al directorio raíz del proyecto (**figura 38**).
- Editar el Makefile del proyecto y descomentar la línea **#USE_EFS = YES** (quitar #)
- Incluir el .h en main.c/cpp : **#include "efs_lib.h".**
- El directorio raíz del sistema de ficheros está en el directorio **efsroot** del proyecto. Copiar los recursos en el directorio.

También puede almacenar sprites y fondos para acceder a ellos cuando se desee.
Mi versión de PAlib no tiene ningún ejemplo con EFSLib así que, habría antes de nada que obtenerlas.

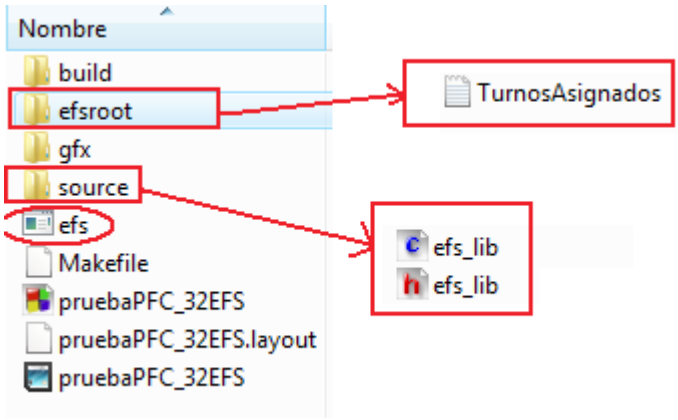

Figura 38: Directorio de un proyecto con EFS lib

Ejemplo de código para la iniciar las librerías EFS:

```
if(!EFS_Init(EFS_AND_FAT | EFS_DEFAULT_DEVICE, NULL));
   {
        PA_OutputSimpleText(1, 0, 1, "Error en EFS...");
        while(true) PA_WaitForVBL();
   }
```

Se ha intentado hacerlas funcionar pero en la ejecución no funciona como se espera, por lo que no se ha trabajado con ellas.

6.3 FAT

Es el sistema de ficheros FAT de la SD y es más sencillo que usar EFSLib. Son las elegidas en la aplicación del proyecto, pero se han detectado problemas de compatibilidad con varios de los “kernel” que he probado, funcionando sólo con la tarjeta R4 con el “kernel” de las versiones de Wood (v1.13 y v1.29).

Dos son los ejemplos que se encuentras en PAlib en
C:\devkitPro\PALib\examples\Filesystem\FATListDirectory y en
C:\devkitPro\PALib\examples\Filesystem\FATReadWrite.

Para poder usar el sistema FAT, sólo hay que incluir *fat.h* e inicializarla con la función *bool fatInitDefault().* Seguidamente, ya se puede usar cualquier instrucción de entrada y salida estándar.

El error que ocurre si se intenta acceder a alguna de estas funciones sin haber inicializado correctamente el sistema FAT será el que apunta la **figura 39**.

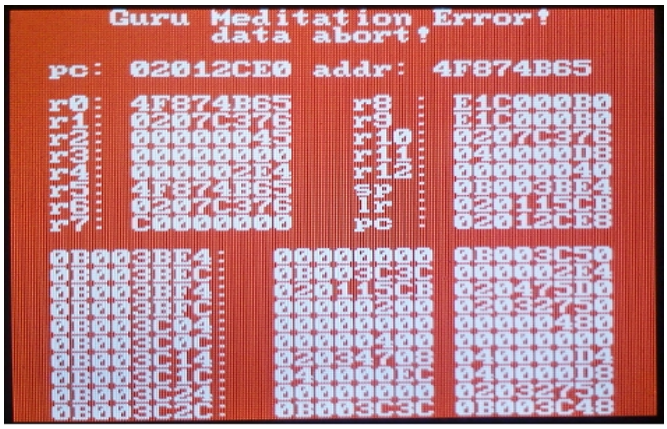

Figura 39: Error al acceder al sistema FAT

Para los juegos este es el sistema que se usa para crear y guardar la información de los datos conseguidos en cada partida en el archivo .sav.

Aunque para mi aplicación también se ha estudiado la posibilidad de usar librerías exclusivas para el tratamiento de ficheros XML, como libxml2 y mxml, ambas en C. Pero ha dado muchos problemas para su compilación e instalación en devkitPro.

7 Introducción Wi-Fi

El Wi-Fi es un punto clave en el proyecto, pues se estudia las funciones y las posibilidades que puede ofrecer en la consola. Como se ha comentado en el apartado de hardware, la implementación Wi-Fi en Nintendo DS soporta el subconjunto de IEEE 802.11b (a 2.4 Hz), de 1 a 2 Mbps, compatible sólo con seguridad WEP y que además no incorpora pila TCP ni protocolos IP. Una de las desventajas es que no funciona en los emuladores, por lo cual siempre habrá que probarlo en la propia consola.

PAlib no ofrece casi soporte propio para tratar el Wi-Fi pero existe la librería **dswifi** que es la encargada de ofrecer lo máximo en Wi-Fi para el "homebrew" y la que se va a usar en la aplicación. Ésta librería también implementa una pila TCP/IP que usa las funciones estándar de acceso a la red mediante sockets: **socket, bind, connect, send, recv,** etc.

Es importante decir que para su funcionamiento con las PAlib es necesario modificar ligeramente el *Makefile* comentando con # la primera línea y descomentando la segunda de la siguiente forma:

```
#ARM7_SELECTED := ARM7_MP3
ARM7_SELECTED := ARM7_MP3_DSWIFI
#ARM7_SELECTED := ARM7_MAXMOD_DSWIFI
```

Donde el "ifeq" que se ejecuta es la que marco en negrita a continuación, correspondiendo a la línea descomentada del "ARM7_SELECTED". Se puede observar todas las librerías que permite usar ésta línea.

```
ifeq ($(strip $(ARM7_SELECTED)), ARM7_MP3)
     LIBS := -lfilesystem -lfat -lnds9
endif
ifeq ($(strip $(ARM7_SELECTED)), ARM7_MP3_DSWIFI)
     LIBS := -lfilesystem -lfat -lnds9 -ldswifi9
endif
ifeq ($(strip $(ARM7_SELECTED)), ARM7_MAXMOD_DSWIFI)
     LIBS := -lfilesystem -lfat -lnds9 -ldswifi9 -lmm9
endif
```

La primera línea de las tres opciones del "ARM7_SELECTED" es para usar las funciones normales de PAlib, mientras que la segunda es para activar la librería dswifi (es la que se usa en la aplicación) y finalmente la última además de la librería dswifi incorpora **libmm9** para trabajar con el sonido (#include <maxmod9.h>).

7.1 Ejemplos Wi-Fi de libnds y PAlib

Tanto en las librerías libnds y PAlib se encuentran algunos ejemplos básicos muy útiles para saber como trabaja el Wi-Fi en la nds y son muy buenos para dar comienzo.

Los ejemplos de **libnds** se localizan en C:\devkitPro\examples\nds\dswifi, que está compuesto de tres ejemplos llamados **autoconnect, ap_search** y **httpget.**
El primero, simplemente realiza una conexión vía WFC; ap_search muestra por pantalla los puntos de accesos cercanos y permite conectarse a uno de ellos; y finalmente, httpget envía una simple petición HTTP a www.akkit.org e imprime sus resultados por pantalla.
.
Los ejemplos de **PAlib** se encuentra en C:\devkitPro\PALib\examples\Wifi\dswifi, también lo componen tres ejemplos llamados **Connect, DownloadFile** y **LeaderBoard.** En este caso el primero realiza la misma función que el ejemplo de autocconect, solo que con funciones de PAlib (mismo resultado); el segundo carga una imagen de una url; y LeaderBoard te permite seleccionar una puntuación (como la conseguida en un juego) y enviarlas a http://leaderboard.palib.info , respondiendo con un número, siendo este la clasificación conseguida gracias a la puntuación.

Quizás la forma más cómoda y eficaz de conectar la consola a Internet vía Wi-Fi es mediante la conexión **WFC de Nintendo**, aunque esto significa que antes ha debido ser configurada (**Anexo I**) gracias a un juego o aplicación que lo permita, como Mario Kart, Animal Crossing, DSOrganize etc... Una vez configurada, la información se almacena en la memoria del firmware de la consola, lo que la hace accesible para cualquier otra aplicación o juego que necesite Wi-Fi. Se pueden configurar hasta tres puntos de acceso diferentes.
Entonces cada vez que las funciones de la librería dswifi use la conexión vía WFC, buscará tal información en la memoria sin ningún problema. Una vez la consola active el Wi-Fi, la luz de encendido que se ve en la **figura 40** (verde o roja si queda poca batería) empezará a parpadear para indicar que está en modo de conexión.

Figura 40: Parpadeo con el uso del Wi-Fi

Ejemplo básico de conexión con libnds

```
#include <nds.h>
#include <dswifi9.h>
#include <netinet/in.h>

#include <stdio.h>

int main(){

     consoleDemoInit();  //para poder imprimir por pantalla

     iprintf("Simple conexión de Wifi \n\n");
     iprintf("Conectando vía WFC ...\n");

     //inicializa e intenta conectar vía WFC
     if(!Wifi_InitDefault(WFC_CONNECT))
     {
          iprintf("error al conectar!");

     } else{

               iprintf("Conectado\n\n");

           }

     while(1)
     {
          swiWaitForVBlank();
     }

     return 0;
}
```

Éste ejemplo que he realizado hace la misma función que el ejemplo de **autoconnect,** solo que como se muestra en la **figura 41** proporciona información de la configuración WFC.

Figura 41: autoconnect

Ejemplo básico de conexión con PAlib

```
#include <PA9.h>

int main()
{
	PA_Init();

	PA_LoadDefaultText(0,0);

		PA_InitWifi();	//inicializa el Wi-Fi

		if (!PA_ConnectWifiWFC())	//intenta conectar vía WFC
	{
		PA_OutputText(0, 1, 4, "error al conectarse!");
		return 1;
	}
		PA_OutputText(0, 1, 4, "Conectado!");

	while (1)
	{
		PA_WaitForVBL();
	}

	return 0;
}
```

En este caso, el ejemplo realiza lo mismo que el de Connect.

ap_search

No todos los ejemplos han funcionado como se esperaba. El ejemplo de ap_search, cuya idea resultaba ser muy útil para el proyecto, donde permite conectarse a un punto de acceso cercano, no ha dado resultado. Han sido muchas las pruebas realizadas para ver qué y cómo hacía el código fuente y ver dónde estaba el posible problema.

Figura 42: ap_search

En la **figura 42** corresponde a la pantalla inferior, donde muestra el número de puntos de accesos detectados y una lista con los nombres, eligiéndolo con el asterisco que se ve en la parte izquierda. El código venía con un pequeño defecto que se corrigió sin problemas, resultó que en el primer bucle while del main, había un " ; " que hacía de éste un bucle infinito.
Una vez corregido, se estudió paso a paso lo que hacía el ejemplo. La explicación de las funciones se encuentran en el siguiente **apartado 7.2.**

- Wifi_InitDefault(false); Primero simplemente inicia conexión, pero sin conectarse a ningún punto de acceso gracias al parámetro "false".
- Wifi_AccessPoint* ap = findAP(); crea un Wifi_AccessPoint y llama a la función *findAP()* que devuelve tal PA seleccionado. En la función *findAP()* se encarga de buscar mediante *Wifi_ScanMode()* los PA.
- Wifi_SetIP(0,0,0,0,0); es la función clave con la que se han realizado muchas pruebas, pero no funciona como debería. Así de esta forma, el router del PA proporciona toda la información mediante el DHCP.
- Wifi_ConnectAP(ap, WEPMODE_NONE, 0, 0); esta función conecta la NDS al PA seleccionado.
- Finalmente se espera la respuesta de conexión o de error esperando a que "status=ASSOCSTATUS_ASSOCIATED ó status=ASSOCSTATUS_CANNOTCONNECT".

Llegados a este punto, el ejemplo se queda en un bucle infinito sin dar respuesta alguna, mostrando por pantalla ".", como si el PA no le respondiera. Código de este bucle.

```
while(status != ASSOCSTATUS_ASSOCIATED && status != 
ASSOCSTATUS_CANNOTCONNECT) {
     int oldStatus = status;

     status = Wifi_AssocStatus();

     iprintf("%s", oldStatus != status ? 
ASSOCSTATUS_STRINGS[status] : ".");

     scanKeys();

     if(keysDown() & KEY_B) break;

     swiWaitForVBlank();
}
```

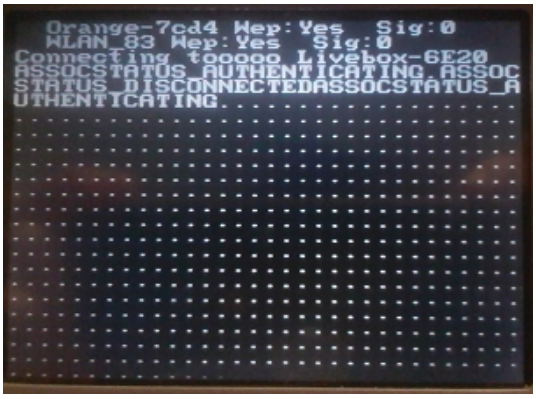

Figura 43: Esperando respuesta en ap_search

Lo más curioso es que, mientras está esperando respuesta de la forma que aparece en la **figura 43**, en la configuración del router se ve como éste le asigna una IP nueva, con un tiempo limitado de 60 segundos y cuya dirección física pertenece a la consola NDS con la que estoy trabajando. Una vez pasado el tiempo, la IP desaparece (**figura 44**).

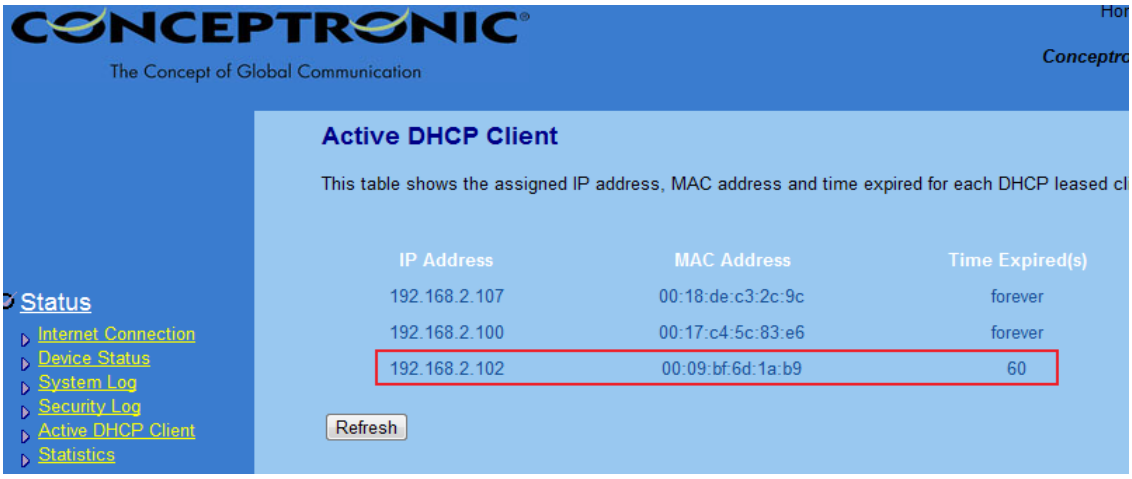

Figura 44: Active DHCP Client

Pero si ahora a la función Wifi_SetIP(0,0,0,0,0), le cambiamos los parámetros para que el router proporcione la configuración deseada por el usuario, por ejemplo:

- Wifi_SetIP(0xC0A80269,0xC0A80201,0xFFFFFF00,0x3E2A3F34,0x3E2AE618);

Los parámetros escritos en orden son la dirección IP, la puerta de enlace, la máscara, el DNS primario y el secundario, que deben ser escritos sólo en hexadecimal, ya que en decimal el compilador se queja. Sólo permite poner en decimal el 0 para indicar que tal parámetros se asignen vía DHCP.

Entonces, al elegir el PA deseado, ya no se queda en el bucle de forma infinita, pasando a mostrar la información de la **figura 45** por la pantalla. Como en la **figura 45** muestra por pantalla líneas repetidas, he marcado en rojo los parámetros a observar. El primer parámetro sería “conectado IP (): LAMDAC” que corresponde al nombre del router al que me he conectado, y siguen los parámetros puestos en la función *Wifi_SetIP,* que serían la dirección IP, la puerta de enlace, la máscara, el DSN primario y el secundario. Otro punto

a observar, es que muestra por pantalla los datos de forma inversa, por ejemplo la dirección IP 192.168.2.105 aparece así 105.2.168.192. Comprobé si esto podría causar algún problema, pero no pasa nada, está todo correcto.

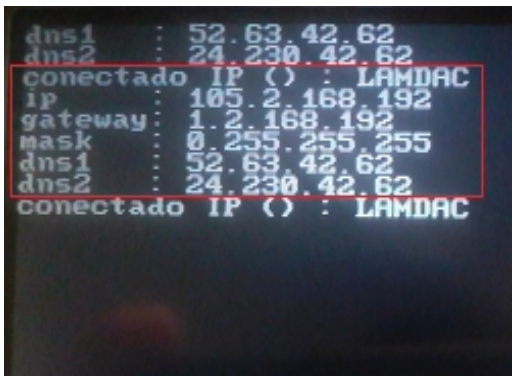

Figura 45: Información en ap_search

De esta forma, si se mira en la configuración del router, éste no proporciona ninguna IP a la consola, que debería de ser la 192.168.2.105 (corresponde a 0xC0A80269 en hexadecimal). Al rato mostraba por pantalla la frase “Could not resolve”. Además de haber insistido en esta configuración, se ha intentado descargar un fichero mediante la utilidad del ejemplo DownloadFile, y sin éxito.

Nota: se ha probado tanto con cifrado WEP y sin cifrado alguno.

httpget

Éste ejemplo funciona a la perfección y similar al ejemplo de PAlib **DownloadFile**. Realiza una petición HTTP y escribe el resultado por pantalla (**figura 46**).

"GET /dswifi/example1.php HTTP/1.1"
"Host: www.akkit.org"
"User-Agent: Nintendo DS";

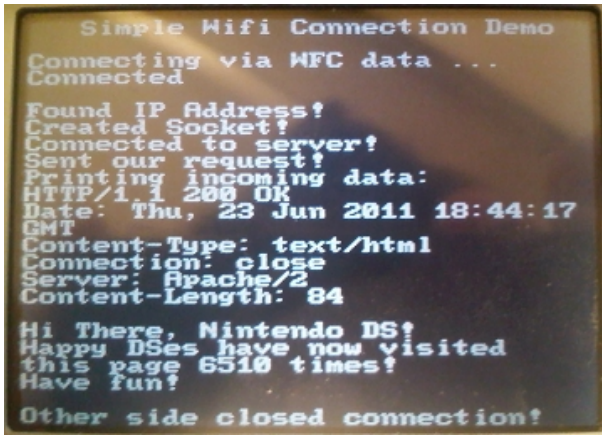

Figura 46: httpget

Para ello se necesita crear un socket y que se conecte al puerto 80. Como resultado se obtiene el texto “Hi There, Nintendo DS! Happy DSes have now visited this page X times! Have fun!”

DownloadFile

Gracias a este ejemplo, haciendo unas modificaciones para ajustarlo a los requisitos de mi aplicación, se ha conseguido descargar desde la web los títulos de los trabajos (en formato .txt, **figura 48**) al directorio de la consola para trabajar con el. En la **figura 47** muestro el contenido obtenido en la descarga.

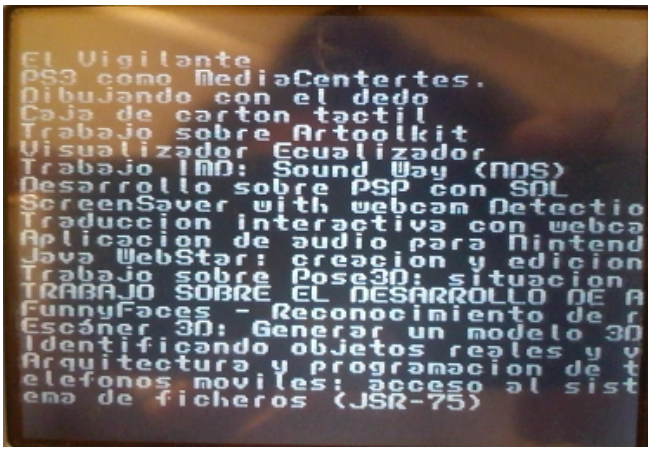

Figura 47: Descarga de los títulos mediante DownloadFile

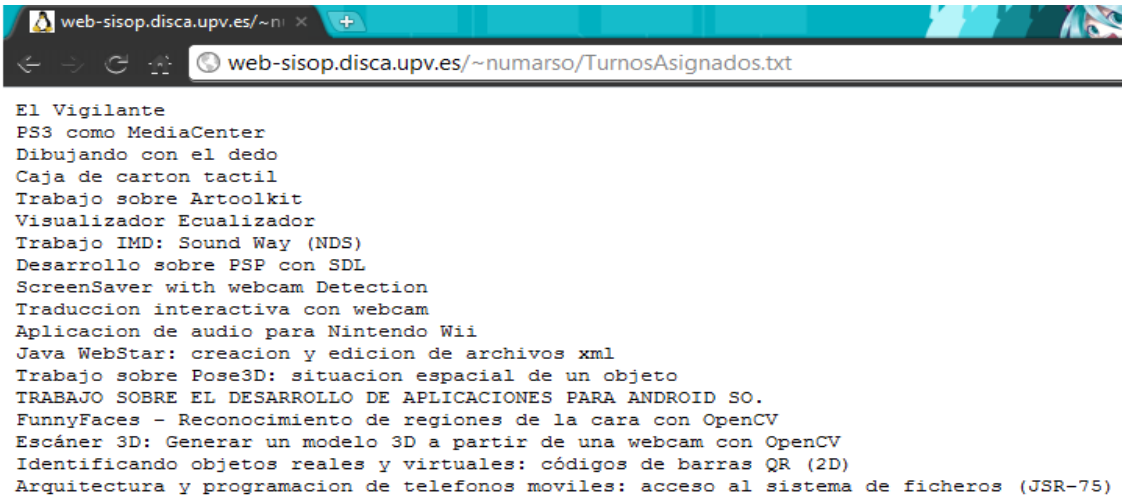

Figura 48: Vista del .txt desde la web

El ejemplo que viene en PAlib realiza una descarga de una imagen, pero no funciona mostrando errores (explicado en los siguientes párrafos). Es posible que tal imagen ya no exista en la red o por otras causas. Por ello se ha hecho pruebas con 3 imágenes de distinto tamaño y diferentes. Los tamaños son como de la misma pantalla de la DS 256x192, de 256x256 y más pequeño de 96x96. Con todas estas tres imágenes el programa mostraba por pantalla el texto “Done<0> Recv X bytes” donde X es el número de bytes recibidos de la descarga (ej. de bytes recibidos en la imagen más pequeña sería de 11953 bytes). No ha mostrado ninguna imagen de las tres por pantalla, pero si un error muy parecido a la figura ya vista anteriormente en el **apartado 6.3**, la **figura 39.**

Hablando más sobre este ejemplo es que escribe por pantalla errores si fuera el caso, mostrando una serie de números, solo que no explica cual es la causa del error. Para verlos habría que mirar directamente al código fuente. A continuación muestro un trozo del código modificado por mi para la obtención de los títulos, en la parte donde mostraría el error o en el caso contrario el número de bytes recibidos en la descarga.

```
err = ky_GetUrl(url, &bfr);

if (err < 0)
{

PA_OutputText(1, 1, 5, "Error: %d, %d, %d | %d: %s", err,
ky_errno, ky_h_errno, errno, strerror(errno));

} else {

     PA_OutputText(0, 1, 3, "%s<%d> Recv %d bytes.", (err == 0) ?
"Done" : "Fail", err, bfr.length);

            if (bfr.length > 0)
            {

            fprintf (abrirFichero, "%s", bfr.buffer);
            PA_OutputText(0, 0, 2, "%s", bfr.buffer);
            fclose(abrirFichero);

             }
        }
```

El error principal se devuelve en la variable *err*, con dos tipos de errores posibles donde devuelve la suma de -30 más otro error o -20 más otro error. Más común el error -20.

- Si err=-30 el error proviene de *ky_InitLink()* (que a su vez devolverá otro número para sumárselo a -30), este error significa que no se ha podido resolver el enlace para realizar la consulta HTTP.
- Si err=-20 el error entonces viene de más adelante, de *ky_Connect()* que se encarga de hacer la conexión de la url mediante sockets. Puede devolver desde -1 hasta -5. Un error típico suele ser la suma de -24, donde el -4 significa que la resolución del hostname no se pudo resolver o podría ser una IP mal formada.
- Si err=0 no hay error.

Éste ejemplo también se ha utilizado para subir un fichero desde la DS hasta una web gracias a un servidor PHP (**apartado 9**), pero sólo permite subir uno, al siguiente intento saltan errores (-24, 0, 0 | 22) y no he conseguido encontrar la solución al error.

Importante decir que trabajar con el ejemplo de DownloadFile para obtener el fichero de los títulos de los trabajos desde la web y usarlo en la aplicación con la librería FAT, sólo ha sido posible en la versión de R4 de **Wood v1.29** ha sido compatible con las dos sin mostrar problemas. Probando con otros “kernel” o que me funcionaba Wi-Fi correctamente y FAT no, o viceversa.

LeaderBoard

Éste ejemplo es una demo en estado beta que sirve para aportar una clasificación en línea proporcionando una ubicación central para registrar las puntuaciones de juegos caseros. En el enlace http://leaderboard.palib.info/games.php se puede observar puntuaciones subidas por usuarios.

El código siguiente es la parte importante del ejemplo, cuya función estrella sería:

- bool PA_SubmitScore(char *response, char *gamekey, char *playername, int score, char *level, u8 responsetype);

donde en *response* obtiene la respuesta del servidor, *gamekey* es una clave que hay que enviar para que la resolución sea aceptada, *playername* el nombre del jugador (mediante **PA_UserInfo.Name** se envía el nombre del usuario escrito en la configuración principal de la consola) , *score* la puntuación realizada, *level* el nivel donde se ha obtenido tal puntuación y finalmente *responsetype* donde con un 0 se recibe las posiciones superior al 10 y la propia posición personal y con 1 sólo el número de posición personal. Sólo que cuando se elige *responsetype*=0 no llega ninguna respuesta (**figura 49),** si *responsetype*=1, funciona correctamente.

```
if (Pad.Newpress.Start)
{
     sprintf(levelname, "Level_%d", level);

     submitted      =      PA_SubmitScore(highscore,      gamekey,
(char*)PA_UserInfo.Name, score, levelname,  rt );

     PA_ClearTextBg(1);

     if (submitted)
     {
           if (sprintf(readhighscore, "%s", highscore))
          {
                PA_OutputText(1,2,2, highscore);
           }
           else{
                          PA_BoxText(1,2,2,31,22,  "Your  score  was
sent to the online leaderboard but there was a problem with the
response.\n\nCheck  out  your  score  online:  \nleaderboard.\n
palib.com", 1000);
                }
     }
     else
     {
           PA_BoxText(1,2,2,31,22, "Your score was not transmitted
succesfully.  Press Start to try again",100);
     }
}
```

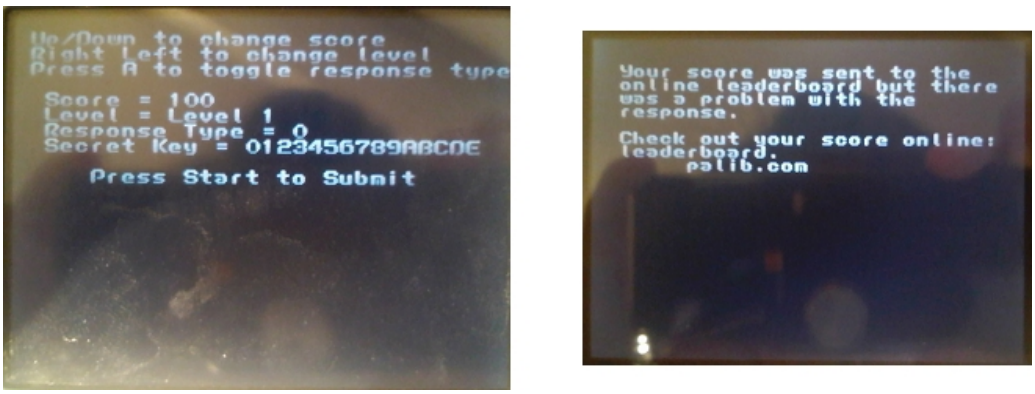

Figura 49: LeaderBoard

7.2 Funciones Wi-Fi

Voy a explicar el API de dswifi y algunas funciones que se encuentran en la librería PAlib, aunque éstas estén basadas en dswifi. La librería dswifi fue implementada por Stephen Stair, aunque su página oficial dedicada a ésta librería no ha sido actualizada desde finales del 2008.

La programación del sistema Wi-Fi en Nintendo DS requiere una comunicación entre los procesadores ARM9 y ARM7, lo que implica ser implementada en las dos, podemos encontrar funciones en **dswifi9.h** y **dswifi7.h**. Algunas funciones que nombraré de PAlib se encuentran en **PA_Wifi.h**.

Las funciones Wi-Fi del ARM7, son las que se encargan de acceder a los registros de la memoria del firmware de la consola para obtener los datos de conexión WFC, entre otros.

Función	Descripción	Ubicación

bool **Wifi_InitDefault**(bool useFirmwareSettings);	Inicia el Wi-Fi. Si *useFirmwareSettings* es true se conectará vía WFC, sino simplemente inicia sin conexión.
	
	`#define WFC_CONNECT true`
	`#define INIT_ONLY  false`
		dswifi9

unsigned long **Wifi_Init**(int initflags);	Inicializa la biblioteca Wi-Fi (ARM9) y la biblioteca sgIP. El parámetro *initflags* sirve para configurar algunas cosas opcionales, como por ejemplo cuando el LED de la consola parpadea.	dswifi9
int **Wifi_CheckInit**();	Se encarga de verificar que el ARM7 se ha inicializado correctamente. Devuelve 1 si el ARM7 está listo para el Wi-Fi, 0 en caso contrario	dswifi9
void **Wifi_DisableWifi**();	Encarga al ARM7 desactivar el Wi-Fi y dejar de recibir.	dswifi9

void **Wifi_EnableWifi**();	Ordena al ARM7 a entrar en un modo "activo" del Wi-Fi pero sin llegar a realizar conexión con ningún punto de acceso.	dswifi9
void **Wifi_ScanMode**();	Encarga al ARM7 para buscar periódicamente por los canales para recoger y registrar la información de los puntos de acceso cercanos de forma interna.	dswifi9
int **Wifi_GetNumAP**();	Devuelve el número de puntos de accesos encontrados por la zona.	dswifi9
Int **Wifi_GetAPData**(int apnum, Wifi_AccessPoint * apdata);	Recupera la información del punto de acceso seleccionado mediante un número (*apnum*) almacenándolo en *apdata*.	dswifi9
int **Wifi_FindMatchingAP**(intnumaps, Wifi_AccessPoint *apdata, Wifi_AccessPoint * match_dest);	Determina si existen varios puntos de acceso en el área local. Proporciona una lista de PA, retornando al índice de la lista del primer PA.	dswifi9
int **Wifi_ConnectAP**(Wifi_AccessPoint * apdata, int wepmode, int wepkeyid, unsigned char * wepkey);	Sirve para conectarse al punto de acceso del argumento *apdata*, *wepmode* indica si se utiliza WEP y de qué tipo. WEPMODE_NONE = 0 WEPMODE_40BIT = 1 WEPMODE_128BIT = 2 *wepkeyid* es un ID y *wepkey* la clave WEP. Devuelve 0 si todo es correcto, -1 si existe error.	dswifi9
void **Wifi_SetPromiscuousMode**(int enable);	La DS entra o sale de un modo "promiscuo", que sirve para capturar todo el tráfico que circula por ella. Con *enable* a 0 se desactiva. `WifiData->reqReqFlags |= WFLAG_REQ_PROMISC;`	dswifi9
void **Wifi_AutoConnect**();	Se conecta al punto de acceso especificado en la WFC de la consola.	dswifi9
int **Wifi_AssocStatus**();	Devuelve la información del estado de la conexión al punto de acceso que se esté conectado en ese momento.	dswifi9

	ASSOCSTATUS_DISCONNECTED ASSOCSTATUS_SEARCHING ASSOCSTATUS_AUTHENTICATING ASSOCSTATUS_ASSOCIATING ASSOCSTATUS_ACQUIRINGDHCP ASSOCSTATUS_ASSOCIATED ASSOCSTATUS_CANNOTCONNECT	
int **Wifi_DisconnectAP**();	Desconectarse del punto de acceso.	dswifi9
unsigned long **Wifi_GetIP**();	Devuelve la dirección IP actual de la DS codificada mediante un entero sin signo de 32 bits. Los octetos están ordenados al revés (el primero es el byte menos significativo): - Primer octeto: ip & 0xFF - Segundo octeto: (ip >> 8) & 0xFF - Tercer octeto: (ip >> 16) & 0xFF - Cuarto octeto: (ip >> 24) & 0xFF No puede ser válida sin haber conectado a un punto de acceso anteriormente, o poner la IP manualmente.	dswifi9
struct in_addr **Wifi_GetIPInfo**(struct in_addr * pGateway,struct in_addr * pSnmask,struct in_addr * pDns1,struct in_addr * pDns2);	Recibe la configuración del PA: IP, puerta de enlace, máscara, dns primario y dns secundario.	dswifi9
void **Wifi_Timer**(int num_ms);	Esta función se debe de llamar para realizar una interrupción periódica. Sirve como base para la actualización de la biblioteca SGIP, retransmisiones, tiempos de espera, etc, durante el número *num_ms* enviado. (milisegundos)	dswifi9
void **Wifi_SetIP**(unsigned long IPaddr, unsigned long gateway, unsigned long subnetmask, unsigned long dns1, unsigned long dns2);	Modifica la configuración Wi-Fi, IP, puerta de enlace, máscara y dns. Se deben de escribir en hexadecimal. Ejemplos: 192.168.1.1 es 0xC0A80101 255.255.255.0 es 0xFFFFFF00	dswifi9
int **Wifi_GetData**(int datatype, int bufferlen, unsigned char * buffer);	Recupera datos desde el hardware Wi-Fi. La dirección física (*MacAddr*) o bien	dswifi9

	el número de PA que la consola tenga configurados (de entre los 3 configurados en WFC). En *datatype* habrá que pasarle la información a recuperar:
	
	`WIFIGETDATA_MACADDRESS`
	`WIFIGETDATA_NUMWFCAPS`
	
	bufferlen será la longitud del buffer y *buffer* para copiar los datos. Devuelve -1 en caso de error.	

void **Wifi_Sync**();	Sirve para sincronizar ARM9 con ARM7.	dswifi9 dswifi7
void **Wifi_SetSyncHandler**(WifiSyncHandler sh);	Se encarga de decir a la librería Wi-Fi que notifique al ARM7 mediante el manejador gracias a un mensaje (*sh*) enviado por FIFO, y viceversa con el ARM9	dswifi9 dswifi7
u32 **Wifi_GetStats**(int statnum);	Recupera un elemento de las estadísticas del Wi-Fi que se han ido reuniendo de paquetes o bytes recibidos. El parámetro *statnum* es el número que debe devolver de WIFI_STATS, o 0 en caso de fallo. Se clasifican en estadísticas software y hardware. `WifiData->stats[statnum];`	dswifi9
int **Wifi_RawTxFrame**(unsigned short datalen, unsigned short rate, unsigned short * data);	Envía un frame a una tasa especificada. Parámetros: *datalen* es la longitud en bytes del frame a enviar, *rate* es la tasa (especificado como mbits/10, 1 Mbit = 0x000A, 2 Mbit = 0x0014) y *data* puntero a los datos a enviar (datos sin formato). El paquete incluye la cabecera hasta el final, pero sin incluir CRC. Siempre devuelve 0, pero sin ningún valor.	dswifi9
void **Wifi_RawSetPacketHandler**(WifiPacketHandler wphfunc);	Establece una estructura para procesar los paquetes que vayan llegando.	dswifi9
int **Wifi_RxRawReadPacket**(long packetID, long readlength, unsigned short * data);	Usa el manejador WifiPacketHandler para leer los paquetes que hayan llegado. Parámetros: *packetID* es un	dswifi9

	identificador único que localiza el paquete especificado en el buffer interno, *readlength* número de bytes a leer(número+1) y *data*, la ubicación de los datos a leer. Devuelve el número de datos leídos.	
void **Wifi_Update**();	Actualiza el ARM7. Debe de ser llamado periódicamente, como el *PA_WaitForVBL()* de PAlib o el *swiWaitForVBlank()* de libnds, para asegurar que los datos siguen fluyendo entre las dos CPU sin problemas. También se ejecuta con la llamada de otras funciones, como Wifi_Sync() o Wifi_Timer().	dswifi9 dswifi7
void **Read_Flash**(int address, char * destination, int length);	Lee una cantidad arbitraria de datos desde el chip del firmware. Parámetros: *address* es el offset para empezar a leer desde el chip; *destination* es un buffer para almacenar los datos del chip que se vayan leyendo; *length* el número de bytes a leer	dswifi7
void **Wifi_Interrupt**();	Manejador de la interrupción Wi-Fi del ARM7. No debería de ser llamada si hubieran otras habilitadas.	dswifi7
void **Wifi_Init**(unsigned long WifiData);	Requiere los datos devueltos *WifiData* de la función Wifi_Init de ARM9, entonces esta función del ARM7 habilita el Wi-Fi, lo que reduce de manera considerable la batería de la consola. (llama internamente a *GetWfcSettings()* para cargar los datos de WFC)	dswifi7
void **Wifi_Deinit**();	En el caso de que fuera necesario, esta función corta el sistema del Wi-Fi. Si se quisiese utilizar de nuevo, es obligatorio usar Wifi_Init.	dswifi7
void **installWifiFIFO**();	crea la conexión FIFO entre los procesadores.	dswifi7
bool **PA_ConnectWifiWFC**();	Conecta vía WFC.	PAlib
long **PA_chartoip**(char *message);	Adquiere la IP mediante el parámetro nombre *message,* que es el nombre del host.	PAlib
Int **PA_InitSocket**(int* socket, char* host, int port, int mode);	Para crear un socket de forma sencilla, donde *host* puede ser tanto una ip como	PAlib

	una dns, *port* el el puerto del socket y *mode* es el modo del socket : `PA_NORMAL_TCP` (para un socket TCP) `PA_NONBLOCKING_TCP` (para un socket TCP no bloqueado) Devuelve 1 si funciona, en caso de error 0.	

int **PA_InitServer**(int* sock, int port, int mode, int num_connect);	*sock* es el socket que escuchará los mensajes, *port* es el puerto por donde se escuchará, *mode* el el modo del socket:
	
	`PA_NORMAL_TCP`
	`PA_NONBLOCKING_TCP`
	
	y *num_connect* es el número máximo de conexiones siendo su limitación hardware alrededor de 20.
		PAlib

bool **PA_SubmitScore**(char *response, char *gamekey, char *playername, int score, char *level, u8 responsetype);	Envía mediante parámetros a un servidor PHP los argumentos, para almacenarlos en la web: http://leaderboard.palib.info/submitscore.php?gamekey=%s&playername=%s&levelname=%s&score=%d&responsetype=%d&end	PAlib

7.3 Estudio de aplicaciones que usan Wi-Fi

Existen varios “homebrew” que usan la librería dswifi y que cuyo funcionamiento es muy interesante a la hora de estudiar esta librería y verificar el uso de algunas de sus funciones. Ejemplo de algunos de estos título son: AirScan, DS2Key, DSFTP, DSOrganize, wifi+lib+test, Win2DS, wifiChat, entre otros.

Se han intentado ejecutar ciertos ejemplos para verificar su funcionamiento respecto a su implementación de dswifi, por ejemplo, el cómo hacen la conexión, que funciones Wi-Fi usan, si hay errores, etc... En algunos casos se necesitan programas o servidores fuera de la aplicación para lograr el funcionamiento de éstos (normalmente dentro de un PC), como es el caso de DS2Key, Win2DS, DSFTP, entre otros.

Muchos de estos ejemplos, están implementados de una forma que tienen tanto el control de ARM9 como el de ARM7, para obtener tal control se debe usar el template de libnds que los combina, ubicado en *C:\devkitPro\examples\nds\templates\combined.* Mediante un sistema FIFO, ambos ARM se comunican entre ellos enviándose los datos el uno al otro y viceversa. Por eso, muchas funciones de dswifi, son para contactar con ARM7. Ejemplos en este caso, Win2DS, wifi+lib+test, DSOrganize, DSFTP, wifiChat...

AirScan

Su funcionamiento tiene un cierto parecido al ejemplo de ap_search, busca y muestra los puntos de acceso cercanos de la zona, con su correspondiente cifrado (WEP, WPA o OPN que es open) y el número de la señal que le llega a la consola (**figura 50**). Es una aplicación más ordenada que el ejemplo ap_search, ya que contiene un desplazamiento más fácil por los puntos de acceso.

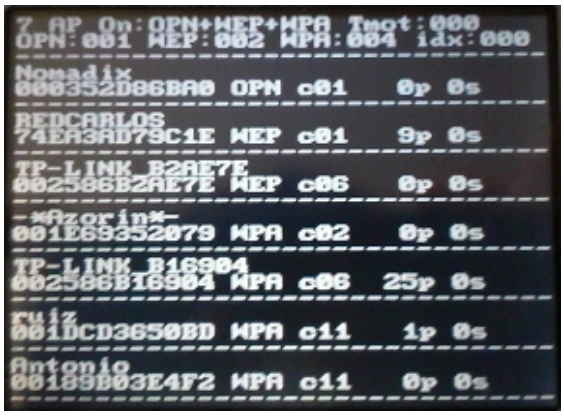

Figura 50: AirScan

Tiene la posibilidad de filtrar solo los PA del cifrado que se quiera. Aunque la aplicación sólo te permite conectarte a PA de tipo OPN, no soporta ni WEP ni WPA ("WEP/WPA AP not supported").

Intenta conectarse a un PA (*Wifi_ConnectAP()*) y conseguir una IP vía DHCP (*Wifi_SetIP()*). Código donde intenta la conexión:

```
int connect_ap(Wifi_AccessPoint * ap)
{
     int ret;
     int status = ASSOCSTATUS_DISCONNECTED;

     clear_main();

     /* Ask for DHCP */
     Wifi_SetIP(0, 0, 0, 0, 0);

     ret = Wifi_ConnectAP(ap, WEPMODE_NONE, 0, NULL);
     if (ret) {
          print_to_debug("error connecting");
          return ASSOCSTATUS_CANNOTCONNECT;
     }

     while (status != ASSOCSTATUS_ASSOCIATED &&
            status != ASSOCSTATUS_CANNOTCONNECT) {
          int oldStatus = status;

          status = Wifi_AssocStatus();
          if (oldStatus != status)
               printf_to_main("\n%s",
                              (char *)ASSOCSTATUS_STRINGS[status]);
          else
               printf_to_main(".");

          scanKeys();
          if (keysDown() & KEY_B)
               break;
          swiWaitForVBlank();
     }

     return status;
}
```

Se puede apreciar en el código un gran parecido al ejemplo ap_search. Según el código, si logra una conexión, intenta hacer una petición a la página principal de google. Todas las funciones que aparecen en el código fuente pertenecen a la librería dswifi. Pero al igual que el ejemplo ap_search, esta aplicación no funciona como debería, nunca logra conexión a los puntos de acceso OPN, imprimiendo por pantalla sin parar “.” mostrado en la **figura 51**.

Figura 51: Probando AirScan

DS2Key v0.2

Esta aplicación sirve para hacer de la DS un pad inalámbrico para emuladores de PC, que no superen los 6 botones más 4 para la cruceta. Emuladores como Megadrive, Supernintendo, Gameboy Advance, nes... Para ello hay que instalarse previamente en el PC el **Parallel Port Joysticks** (PPJoy) y configurarlo. Se ha hecho uso de varios tutoriales, pero éste me llegaba a pedir un ejecutable que no he conseguido encontrar, el "Profile Editor.exe". Por lo tanto esta aplicación no he sabido hacerla funcionar.

También había que poner en marcha un servidor que espera en el puerto 9501. Una vez todo en marcha, era encender la aplicación para nds.

Lo que destaco de esta aplicación es que, al igual que yo, usa las librerías PAlib y las funciones Wi-Fi de PAlib. Además tiene programada funciones para leer de ficheros XML. Debe leer los llamados DSPad.xml y DSPad_server.xml pero al ejecutar el programa siempre da error (**figura 52**). El código se encuentra en el fichero llamado **CORE_xml.c** y se puede ver cómo hace uso de sus funciones en **CORE_fileio.c**, en **CORE_power.c** y en **CORE_turbo_key.c**.

- char* xml_load(char* location): Lee un fichero y devuelve el contenido en una cadena de caracteres.
- int xml_read(void* pt, int(*TreatTag)(void*, char*, char*), char* mark): Lee la cadena. Devuelve 0 si hubo error y 1 si todo es correcto.

Figura 52: Error al leer DSPad.xml

En cuanto a sus funciones Wi-Fi, se pueden encontrar en **CORE_network.c** , donde la aplicación se conecta vía WFC y trabaja generalmente con sockets. Contiene una función para crear un socket (create_socket) y para enviar datagramas (sendto_rs), para que la aplicación envíe al PC las pulsaciones del pad que reciba la consola.

```
/*
    Create a socket
    Return:
            socket id
            -1 if error
*/
static int create_socket(unsigned short *port, int type){
    int sock,  ok;
    struct sockaddr_in  add;
    int l = sizeof(struct sockaddr_in);

    sock = socket(PF_INET, type, 0);
        if(sock == -1){
#ifdef DEBUG
            PA_Print(1, "socket() error\n");
#endif
            return -1;
        }

        if(type != SOCK_DGRAM){
        memset(&add, 0, l) ;
        add.sin_family      = AF_INET ;
        add.sin_port        = htons(*port);
        add.sin_addr.s_addr = htonl(INADDR_ANY);

        ok = bind(sock, (struct sockaddr*)&add, l);
        if(ok == -1){
#ifdef DEBUG
            PA_Print(1, "bind() error\n");
#endif
            return -1;
        }

        ok = getsockname(sock, (struct sockaddr*)&add, &l);
        if(ok == -1){
#ifdef DEBUG
            PA_Print(1, "getsockname() error\n");
#endif
            return -1;
        }
    }

    return sock;
}

/*
```

```
    Send a *complete* datagram in one shot
    Return size of sent datas
*/
static int sendto_rs(int sockfd, const void *msg, size_t count,
int flags, const struct sockaddr *to, int tolen){
    int returnValue = 0;

    do{
            returnValue = sendto(sockfd, msg, count, flags, to,
tolen);
    }while( (returnValue == -1) && (errno == EINTR) );

    return returnValue;
}
```

DSFTP v2.6

Es un programa que ejecuta un servidor FTP en la DS para transferir cómodamente los .nds o cualquier otro fichero sin la molestia de sacar la micro SD. Para que funcione correctamente, DSFTP debe de leer la configuración desde un archivo en **/data/settings/ftp.conf**, por lo que hay que crearse en la SD esta ruta e incorporar ftp.conf. Aquí un ejemplo del contenido de ftp.conf:

```
motd /ftp/motd.txt
logfile /data/logs/ftp.log
loglevel 4
timeout 60
listen 21
portrangestart 9000
portrangeend 9999

screensaver 30
wakeonlog false

user nuria
pass nds
root /
home /
write true
boot true
end user

user anonymous
root /ftp/anonymous
write false
end user
```

También hay que crearse **ftp/motd.txt**, donde motd.txt será un “saludo” para cuando se logre la conexión, **ftp/anonymous** por si la gestión se realiza de manera anónima, y finalmente **/data/logs/ftp.log**.

Según el manual del programa, se ha probado con varios clientes y deberían de funcionar:

- de línea de órdenes FTP en Mac / DOS
- RBrowser Lite en Mac
- MacOS X FTP virtual del sistema de archivos
- Firefox
- FileZilla (con el cual lo he probado yo)
- FireFTP (plugin de Firefox)
- WS_FTP

Cuando se arranca el cliente, se le debe de especificar la dirección IP de la consola DS, y el usuario con la contraseña que se le ha puesto en la configuración o bien como anónimo como indica la **figura 53** (“default” y contraseña generado de forma aleatoria por el propio DSFTP).

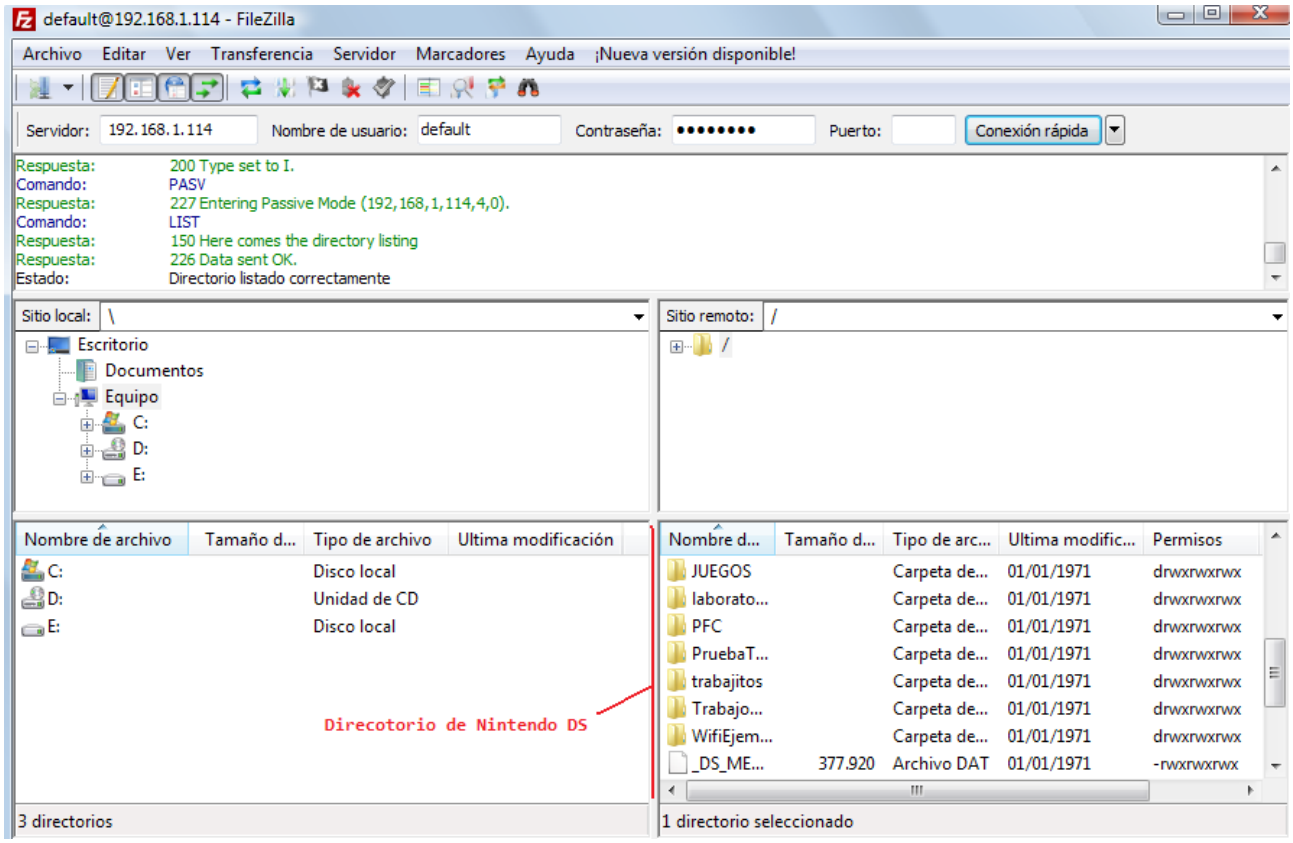

Figura 53: Conexión FTP con FileZilla

Sobre su implementación Wi-Fi solo se puede conectar al punto de acceso especificado en la WFC de la consola, luego comprueba y devuelve el estado de conexión. Como es un programa que debe de estar actualizándose continuamente y puede estar usando el Wi-Fi un gran periodo de tiempo, hace uso de las funciones de aviso y sincronización con el ARM7, *Wifi_CheckInit()*, *Wifi_SetSyncHandler()*, *Wifi_Update()* (*#define WIFI_UPDATE_FREQ 50*), *Wifi_Timer()*... En el fichero **BController.cpp** es donde se encuentra la implementación del funcionamiento Wi-Fi de DSFTP, entre otras más.

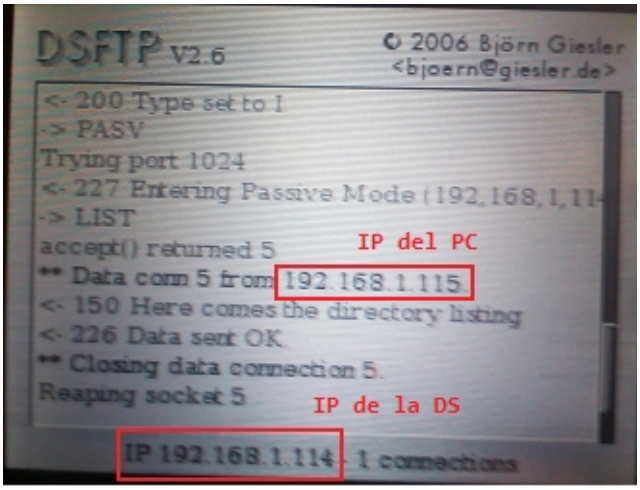

Figura 54: DSFTP

Una vez consigue conectarse la DS vía WFC, entonces llama a la función *initFTP()*, que pone en marcha el servidor FTP, se encarga de leer la configuración de **ftp.conf** y realizar la conexión con el cliente (**figura 54**).

Win2DS v0.82

Esta aplicación consigue conectar la DS con un PC (Windows) mediante un servidor UDP al puerto 8888, y su principal uso es poder manejar el puntero del ratón desde la consola, además de mostrar la pantalla del PC en ésta. Es tener tu PC en la propia consola. Quizás va un poco lento, pero funciona. Para realizar la conexión basta con escribir la dirección IP del PC donde se encuentre en servidor UDP funcionando dando como resultado lo mostrado en la **figura 55**.

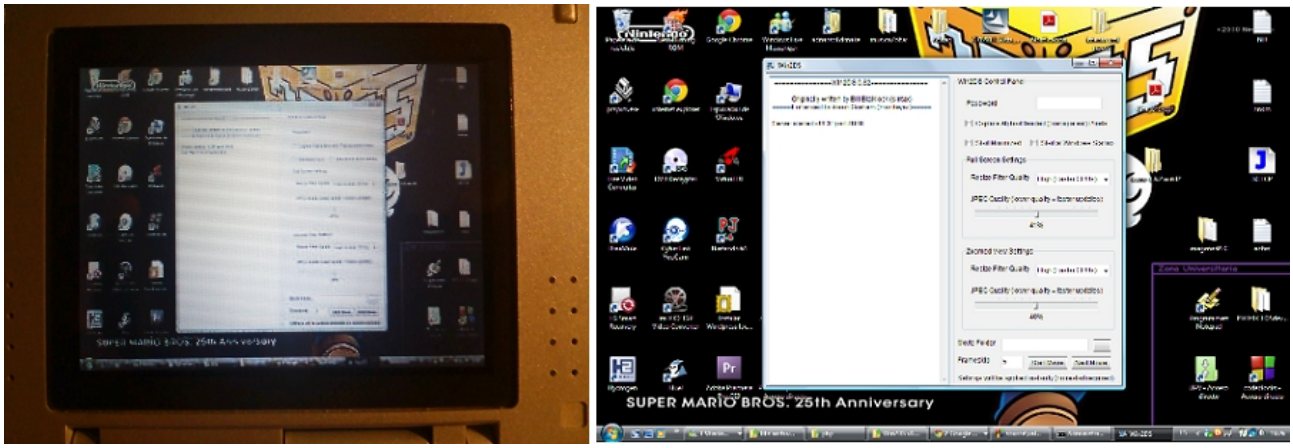

Figura 55: Conexión Win2DS

Sobre la conexión Wi-Fi da dos opciones, o bien vía WFC o con una conexión manual al estilo del ejemplo ap_search. Toda la información se encuentra en el fichero **arm9.cpp**. Pongo el un trozo de código del main donde se da la posibilidad de conexión que nombraba anteriormente (el sel=3 lleva simplemente a información relacionada con el autor), pero el importante es la función *startConnect()*.

```
          if(autoConnSel->getState()==SEL_ACTIVATED)
               sel = 1;
          else if(manualConnSel->getState()==SEL_ACTIVATED)
               sel = 2;
          else if(aboutSel->getState()==SEL_ACTIVATED)
               sel = 3;

          …

          switch                          (sel)
                    {
               case 1:
                    startConnect();
               break;
               case 2:
                    startConnect(true);
               break;
               case 3:
                    …
               break;
          }

          …

}

void startConnect(bool manual=false) {

     …

     if (manual) {
          if (connectionMenu() && selectAP()){
               Wifi_DisconnectAP();
//Wifi_SetStaticIP(global_ipaddr,global_gateway,global_snmask,glob
al_dns1,global_dns2);


Wifi_ConnectAP(&global_connectAP,global_wepmode,global_wepkeyid,gl
obal_wepkeys[0]);
          }
          else {
               Wifi_DisconnectAP();
               Wifi_DisableWifi();
               return;
          }
     } else
          Wifi_AutoConnect();

                                                                    …

}
```

En el caso de que *manual* sea true, entonces *connectionMenu()* permite al usuario escribir la configuración de la red (IP, puerta de enlace, máscara y los dos dns) y después la de *selectAP()*, permite seleccionar un punto de acceso de la zona, de este modo casi nunca suele conectar, hay que tener en cuenta que al PA donde se quiere conectar no tenga ningún cifrado, pues en la configuración no se dice nada de añadir la contraseña y aún siendo WEP no conecta. La **figura 56** es la captura de dicha configuración.

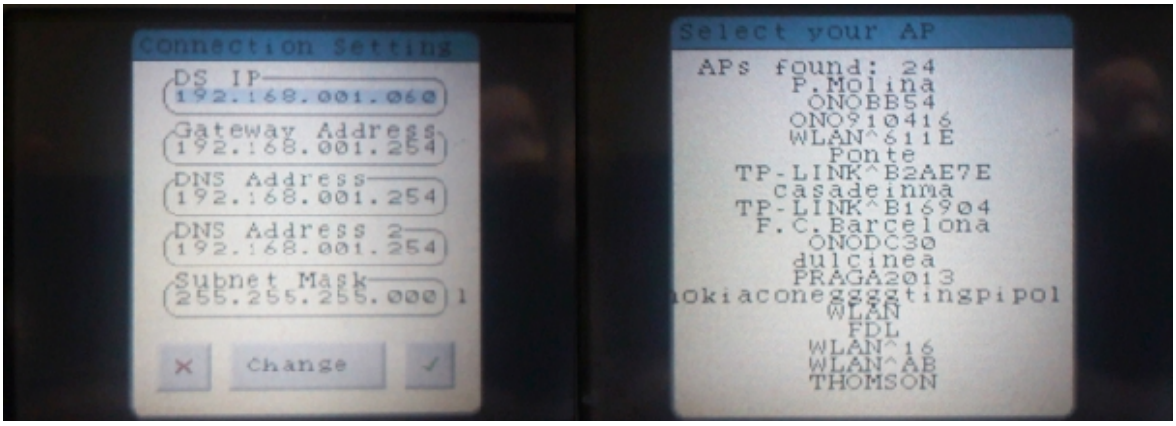

Figura 56: Configuración manual en Win2DS

Una vez conectado, ya sea de una forma u otra, se añade la IP del PC, la función *sendTouchUDP()* se encarga de realizar la conexión con el PC y muestra un nuevo menú de posibilidades: Virtual Desktop, GamePad Mode, Custom Keys, Disconnect y Reconnect.

DSOrganize v.32

Se considera uno de los “homebrew” más completo e interesante realizados hasta el momento ya que incluye: calendario con posibilidad de ir adjuntando citas, libreta de direcciones, lista de tareas, dibujo de notas a mano alzada, explorador de archivos, calculadora, cliente IRC, navegador web, repositorio de “homebrew”. Para que funcione correctamente, hay que incluir en la tarjeta SD un directorio nuevo, llamado **DSOrganize**, con todas las configuraciones por defecto que necesita el programa. Todo lo necesario se encuentra en una carpeta del mismo nombre dentro del .zip, con el ejecutable.

Además, tiene la posibilidad de ver cuales son los tres puntos de accesos ya configurados dentro de la consola y se pueden modificar (**figura 57**), e incluyendo a su vez otros tres espacios más de configuración dentro de la aplicación (en total 6) y lo más importante, es que funcionan todos.

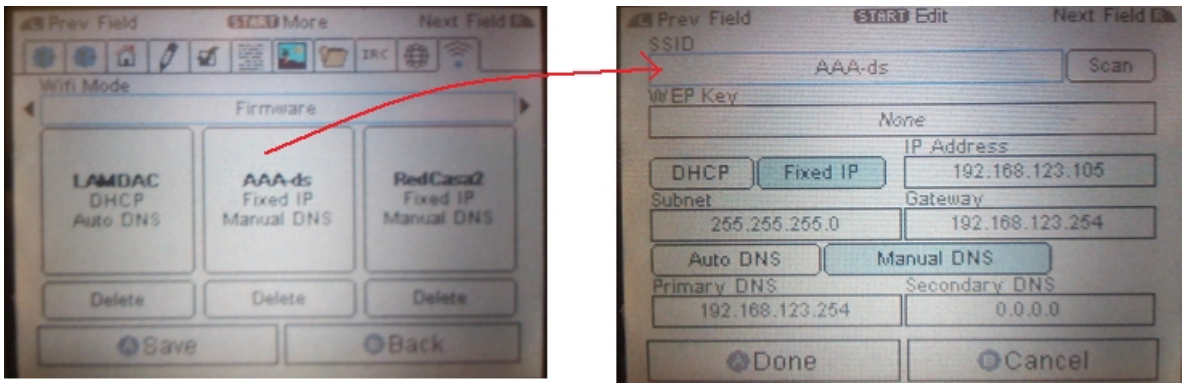

Figura 57: Configuración en DSOrganize

```
if(getSetting("General", "WifiMethod", sStr))
	{
		strlwr(sStr);

		if(strcmp(sStr, "firmware") == 0)
			wifiMode = WIFI_FIRMWARE;
		if(strcmp(sStr, "dsorganize") == 0)
			wifiMode = WIFI_DSORGANIZE;
	}
```

En el código de DSOrganize, dentro del source de arm9, y finalmente en el directorio drawing se encuentra el fichero **configuration.cpp,** es donde se encarga de obtener la configuración Wi-Fi. La función *initWifiInfo()*, avisa al ARM7, que es el que llevará esta tarea mediante *GetWfcSettings()* (**Anexo II**) y accede a los registros de la memoria donde se encuentran todos y cada uno de los datos, y finalmente dentro de initWifiInfo() se llama a *loadWifiInfo()*. Además el método *saveWifiInfo(WIFI_PROFILE *tProfile)* se encarga de guardar la nueva configuración Wi-Fi *tProfile,* que se pasa por parámetro.

Aquí un ejemplo de cómo accede a los datos de la conexión WFC.

```
void loadWifiInfo()
{
	// populate firmware settings
	for(int x=0;x<3;x++)
	{
		if(WifiData->wfc_enable[x] != 0)
		{
			// set up the enabled and wep flags
			wifiInfo[x].enabled = true;
			wifiInfo[x].wepMode  =  (WifiData->wfc_enable[x]  &
0xF);
			// grab the ssid
			strncpy(wifiInfo[x].ssid,  (const  char  *)WifiData-
>wfc_ap[x].ssid, WifiData->wfc_ap[x].ssid_len);

			// grab the wepkey
			memcpy(wifiInfo[x].wepKey,  (const  void  *)WifiData-
```

```
>wfc_wepkey[x], 16);

                // set up the fixed ip settings
                wifiInfo[x].ip = WifiData->wfc_config[x][0];
                wifiInfo[x].gateway = WifiData->wfc_config[x][1];
                wifiInfo[x].subnet = WifiData->wfc_config[x][2];

                // set up dns settings
                wifiInfo[x].primarydns   =   WifiData->wfc_config[x]
[3];
                wifiInfo[x].secondarydns  =  WifiData->wfc_config[x]
[4];

                // now figure out if they have dhcp
                u32 bitsSet = wifiInfo[x].ip | wifiInfo[x].gateway;
                if(bitsSet != 0)
                {
                     wifiInfo[x].dhcp = false;
                }
                else
                {
                     wifiInfo[x].dhcp = true;
                }

                bitsSet       =       wifiInfo[x].primarydns       |
wifiInfo[x].secondarydns;
                if(bitsSet != 0)
                {
                     wifiInfo[x].autodns = false;
                }
                else
                {
                     wifiInfo[x].autodns = true;
                }
           }
           else
           {
                // ensure we don't use this profile
                wifiInfo[x].enabled = false;
                wifiInfo[x].dhcp = true;
                wifiInfo[x].autodns = true;
           }
     }

     // populate normal settings
     for(int x=0;x<3;x++)
     {
          memcpy(wifiInfo    +    3    +    x,    readWifi()    +    x,
sizeof(WIFI_PROFILE));
     }
}
```

En el fichero **wifi_shared.h**, se encuentran las estructuras que manejan el Wi-Fi (**Anexo II**). Es el mismo fichero, (a excepción de una línea incluida) que se encuentra en el código de la librería dswifi. Por eso, del código anterior hace uso de tal estructura:

- extern volatile Wifi_MainStruct * WifiData;

La manera de configurar la conexión trabaja de la misma manera que lo haría un juego original de Nintendo (Mario Kart, Animal Crossing), DSOrganize solo configura el PA dentro del firmware de la consola, para luego usarlo vía WFC de Nintendo. Esto se puede ver, por ejemplo, si se usa el navegador en DSOrganize, que está en **webbrowser.cpp** y para iniciar el Wi-Fi llama a la función connectWifi(), que está en **wifi.h** ejecutando la función *Wifi_AutoConnect()* explicada en el API:

```
void connectWifi()
{
     if(wifiConnected)
          return;

     wifiConnected = true;

     Wifi_AutoConnect(); // WFC de Nintendo
}
```

wifi lib test

Permite hacer "wardriving" y capturar paquetes (*Do_Play_Wardriving()* y *Do_Play_PacketCapture()* en template.c del ARM9), además de conectarte vía WFC o manualmente (**figura 58**). También muestra la dirección física de la consola y los número de los puntos de accesos ya configurados (1,2 o 3), esto se conseguía gracias a la función Wifi_GetData() ya descrita en el **apartado 7.2.**

Nota: "wardriving" se encuentra explicada en Wikipedia:
http://es.wikipedia.org/wiki/Wardriving

```
printbtm(0,0,"DS Info:");
Wifi_GetData(WIFIGETDATA_MACADDRESS,6,(unsigned char *)info);
sprintf(temp,"MAC Addr: %02X:%02X:%02X:%02X:%02X:
%02X",info[0],info[1],info[2],info[3],info[4],info[5]);
printbtm(0,2,temp);

sprintf(temp,"Number of WFC Aps:
%i",Wifi_GetData(WIFIGETDATA_NUMWFCAPS,0,0));
printbtm(0,3,temp);
```

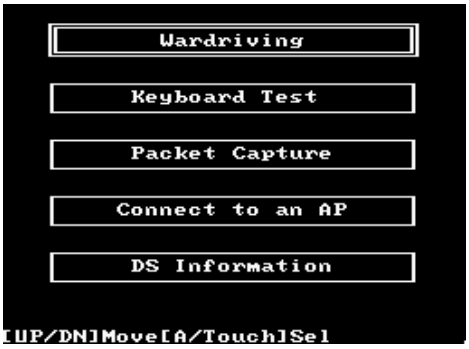

Figura 58: wifi_lib_test

La opción “wardriving” no tiene misterio, realiza una búsqueda de redes inalámbricas cercanas a la zona como hacen los ejemplos anteriores de AirScan o ap_search.
Desde el menú “Connect to an AP” que es como sale en la **figura 59**, podemos configurar los parámetros de la red y dejar que conecte, o elegir un PA cercano y configurar a partir de éste. La configuración se realiza de forma similar a la que proporciona el ejemplo de Win3DS.

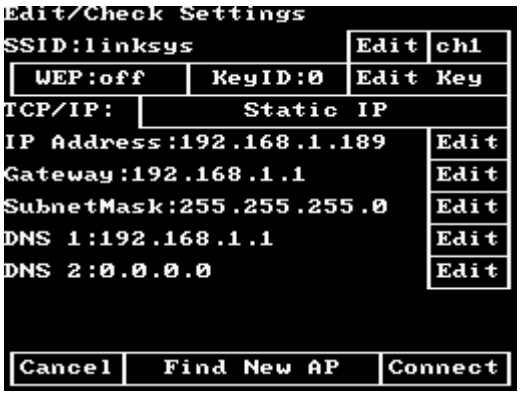

Figura 59: Configuración Wi-Fi

Los parámetros que manejan esto en el código, son variables globales que por defecto se inician a los valores que aparecen exactamente en la **figura 59**:

- Wifi_AccessPoint global_connectAP;
- unsigned char global_wepkeys[4][32];
- int global_wepkeyid, global_wepmode;
- int global_dhcp; // 0=none, 1=get IP&dns, 2=get IP&not dns
- unsigned long global_ipaddr, global_snmask, global_gateway, global_dns1,global_dns2;

Se da la opción de intentar conectar a un punto de acceso mediante DHCP, pero nunca he logrado conexión, ni si quiera con mi router. Luego también se da la posibilidad de editar solo los dos DNS y el resto mediante la obtención de DHCP, pero con esta opción tampoco ha habido suerte (**figura 60**). Finalmente, se puede editar todo lo que sale en la **figura 59,** y así si que he logrado conexión (a mi mismo punto de acceso).

```
int Do_ConnectAP(int easy) {

     …

     Wifi_DisconnectAP();
     // set DHCP/Static IP settings...
     if(!easy) {
          switch(global_dhcp) {
          case 0: // none


Wifi_SetIP(global_ipaddr,global_gateway,global_snmask,global_dns1,
global_dns2);
                    break;
          case 1: // dhcp both
                    Wifi_SetIP(0,0,0,0,0);
                    break;
          case 2: // dhcp IP/not dns
                    Wifi_SetIP(0,0,0,global_dns1,global_dns2);
                    break;
          }
     }

     …
```

Figura 60: Fallo de conexión vía DHCP

Un fallo muy usual como muestra la **figura 60**, es que a la hora de intentar conectar a un PA con configuración automática vía DHCP (*Wifi_SetIP(0,0,0,0,0)* o *Wifi_SetIP(0,0,0,global_dns1,global_dns2)*), tal conexión no se realice con éxito y ocurra un "timeout". Línea de error "sgIP DHCP error timeout!" se localiza en la línea 308 del fichero **sgIP_DHCP.c** en la fuente de la librería dswifi.

Para verificar que realmente se había conectado, hice una prueba mediante la ayuda de un servidor java y un terminal telnet (**SCTCP2.java, Anexo III**). Primero me conecté vía WFC e intenté lanzar wifi_lib_test hacia el servidor, mediante la opción de "TCP connect Test" (*Do_Play_TCPConnect()*) escribiendo la dirección IP de mi ordenador (es donde

está ejecutándose el servidor) al puerto 7077 (que es donde escucha). Como resultado final, el servidor envía un saludo a la consola dando a demostrar la conectividad (**figura 61**):

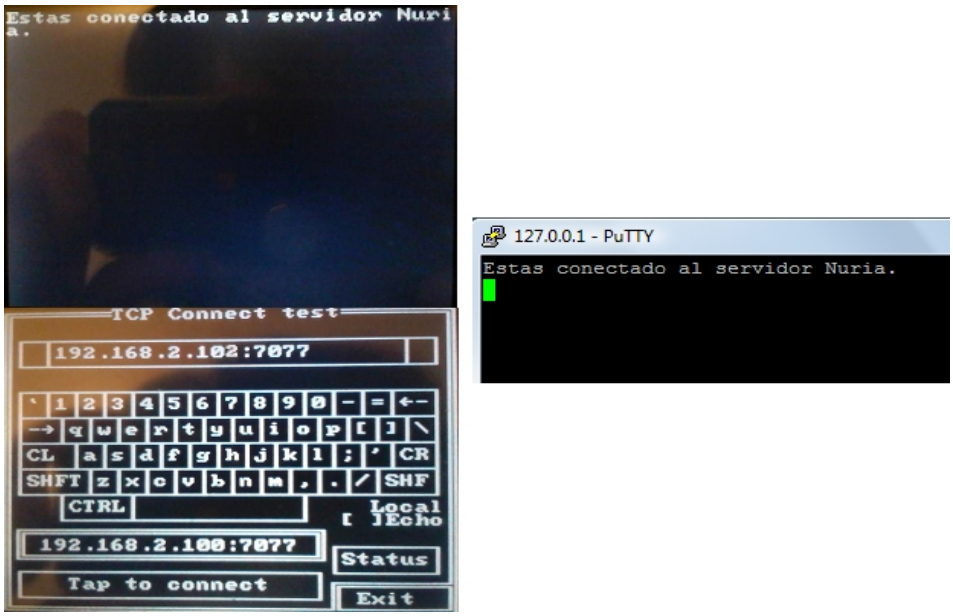

Figura 61: Conectividad con el servidor java

Luego, intenté de misma tarea de conexión al servidor pero conectándome a mi PA de forma manual, y no funcionó, la pantalla queda en negro totalmente.

7.4 Conclusión de conectividad Wi-Fi

Una vez estudiado varios ejemplos e implementado una aplicación con funciones Wi-Fi, como conclusión final de la conectividad de Nintendo DS decir que la más encontrada en estos ejemplos y sencilla de programar (ya que con una función se conecta) sería vía WFC. Esta forma de conexión inalámbrica pienso que es la más útil en el caso de que si la aplicación realizada se utilizase siempre o casi siempre para conectar con el mismo punto de acceso, pues una vez configurada la consola a tal punto de acceso (siempre con un juego o aplicación de forma externa que lo permita) la consola siempre se conectará a dicho punto con solo darle a un botón. Un claro ejemplo es DSFTP, pues está destinada para realizar transferencia de datos con el propio PC del usuario.

Por el lado contrario, si se realizase una aplicación para usarla cada vez con puntos de acceso diferentes la cosa cambia bastante, pues tener implementado una configuración manual en dicha aplicación, como el ejemplo de DSOrganize, sería lo más cómodo a la hora de conectar aunque también más complejo de desarrollar. Hay que tener en cuenta de que al punto que se quiera conectar sea un router con seguridad compatible para la consola. De todos los ejemplos estudiados en el **apartado 7.3** el único con configuración manual que me ha dado confianza por haberme funcionado siempre es DSOrganize, pues es el único ejemplo estudiado que permite guardar la configuración Wi-Fi realizada en la memoria de la propia consola.

De cara a mi aplicación implementar una configuración manual sería una gran mejora, siendo uno de los puntos de trabajos futuros a realizar.

8 Estructura de la aplicación

La aplicación está estructurada por ficheros programados en C y los clasifico en dos grupos de la siguiente manera:

- Los que se encargan de la interfaz gráfica y de interactuar con ella:

  - main.c
  - valoracion.c/.h
  - consultar.c/.h
  - wifi.c/h
  - input.c/.h

- Los que aportan el contenido y funcionalidades:

  - CargaSistema.c/.h
  - CreaFichero.c/.h
  - Ky_geturl.c/.h

La **figura 62** es un esquema con el que pretendo aclarar la conexión que hay entre los ficheros dentro de la aplicación.

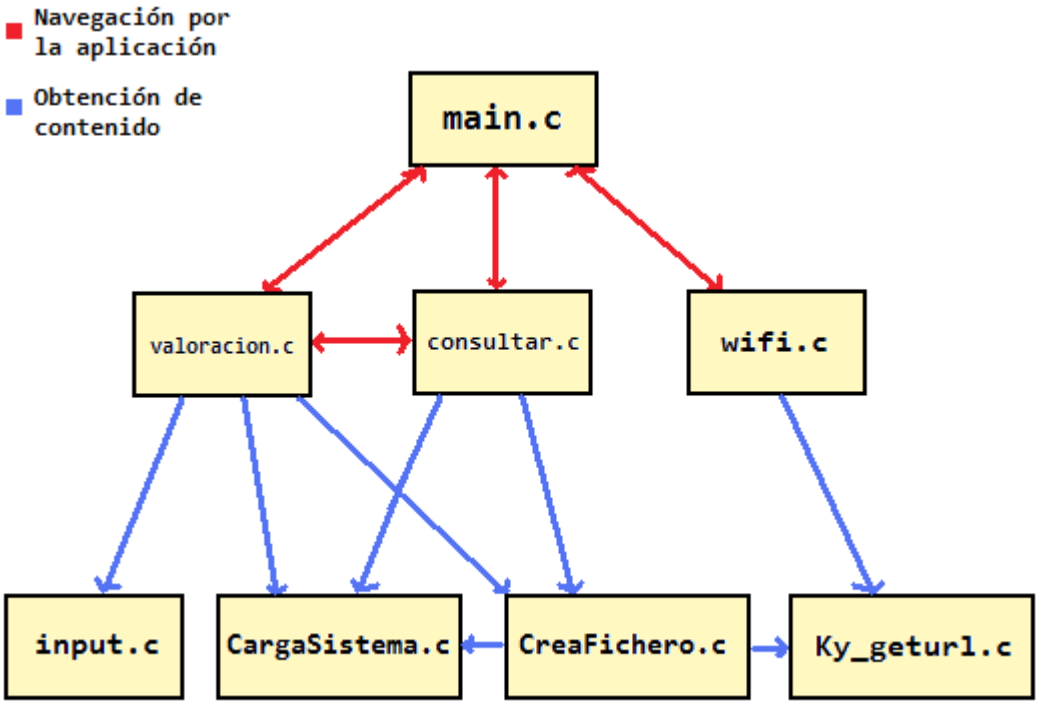

Figura 62: Esquema de la aplicación

Voy a explicar el objetivo de cada fichero uno a uno, además de mostrar el contenido que genera para que sirva a su vez como guía de usuario de cara a la aplicación.

main.c

Comienzo por el main.c, que es el que arranca la aplicación como es de suponer. La tarea que lleva el main es principalmente el inicio y el menú principal, que consta de tres apartados: **Evaluar**, **Consulta** y **Wi-Fi**. Se encargará de realizar la bifurcación hacia cada apartado, como se muestra en el esquema.

Aparte de la bifurcación, se encarga de crear todo el entorno gráfico que se muestra por la consola en la parte que le corresponde (el inicio y el menú). También es el que lleva en cuenta el número de trabajo en el menú de títulos en valoracion.c que ha sido selecionado durante una evaluación o consulta, gracias a la variable *int opcion,* el cual se pasa por referencia a las funciones.

Otro detalle que realiza es la opción de poder apagar o encender la iluminación de las pantallas de la consola al pulsar el sprite que se ve en la esquina inferior derecha de la **figura 63** de color verde con el dibujo de una bombilla.

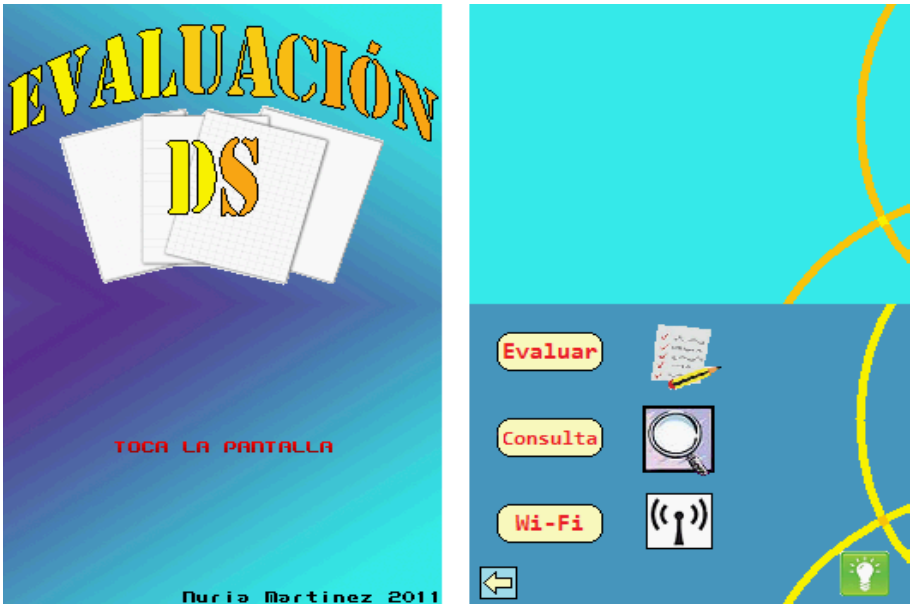

Figura 63: Partes que crea main.c

valoracion.c

Este fichero es el que lleva toda la trayectoria de la evaluación de un trabajo, mostrado en la **figura 64**. Es el que más líneas de código tiene y quizás el más complejo. Aparte de encargarse de todo el entorno gráfico que se vea durante esta fase, se encarga de llamar a CargaSistema.c para que éste le proporcione la información de los títulos del fichero TurnosAsignados.txt y CreaFichero.c, donde plasma toda la información que se ha realizado en la evaluación en un nuevo .txt.

Todo el contenido para crear una evaluación se guarda en variables globales:

- **bool lectura:** Se encarga de comprobar que la aplicación solo lea los títulos del fichero TurnosAsignados.txt una vez, y así no tenga que llamar a CargaSistema.c en cada evaluación.
- **int count:** Lleva la cuenta de títulos que hay en el fichero TurnosAsignados.txt.
- **Int nota1,... nota6:** Son las seis notas de los aspectos a evaluar en cada trabajo.
- **char text[250]:** Aquí irá almacenado el comentario de la evaluación mediante el teclado virtual o el reconocimiento de caracteres, hasta un total de 250 caracteres.
- **int contador:** Sirve para mostrar al usuario el número de caracteres que le quedan como máximo para escribir en el comentario.
- **Int tope:** Es el encargado de controlar el cursor de color rojo que se ha creado para el teclado y el reconocimiento, para ayudar al usuario a la hora de borrar o sustituir caracteres.
- **s32 nletter:** Es un contador que inserta en el vector de *text* la letra obtenida por la función *PA_CheckKeyboard(),* es decir, la siguiente letra a la derecha.
- **bool mayusRS:** Lleva el control de las minúsculas y las mayúsculas en el reconocimiento de caracteres.
- **bool infoImg:** sirve para cambiar la imagen que se muestra como guía para el reconocimiento de caracteres hacia el usuario.

También se encarga de hacer la bifurcación entre la elección del teclado o el reconocimiento de caracteres en la función *int modoEscritura()*, y la posibilidad de enviar o no a la web la evaluación realizada en ese momento en la función *int escribirFichero(int opcion),* donde *opcion* es, como se ha comentado anteriormente, el número de trabajo seleccionado a evaluar.

Al final de cada evaluación, las variables globales se reinician para la próxima evaluación, excepto *bool lectura*.

Como una de las mejoras de la aplicación era la navegación por la aplicación, desde cualquier punto de la evaluación que se encuentre el usuario podrá retroceder en cualquier momento, desde un paso anterior en la evaluación hasta el inicio completo de la aplicación.

Figura 64: Partes que crea valorar.c

Nota: el orden de la evaluación según la **figura 64** es de izquierda a derecha superior, seguido de la inferior (exceptuando la posibilidad de teclado o reconocimiento).

consultar.c

En este punto de la aplicación, muestra el menú de títulos pero desde valoracion.c, y una vez se elige un título muestra, si lo hubiera, el contenido de una evaluación. Da la posibilidad directa de modificar el contenido realizando de nuevo la evaluación o de enviar dicha evaluación a la web vía Wi-Fi. Si se le da a enviar, primero obtiene en variables los datos de la evaluación (el número de los aspectos y el comentario) mediante la función *bool obtieneDatosConsulta(opcion).*

Si no existe el fichero .txt de la evaluación, simplemente da la opción a realizar su evaluación (**figura 65** parte derecha).

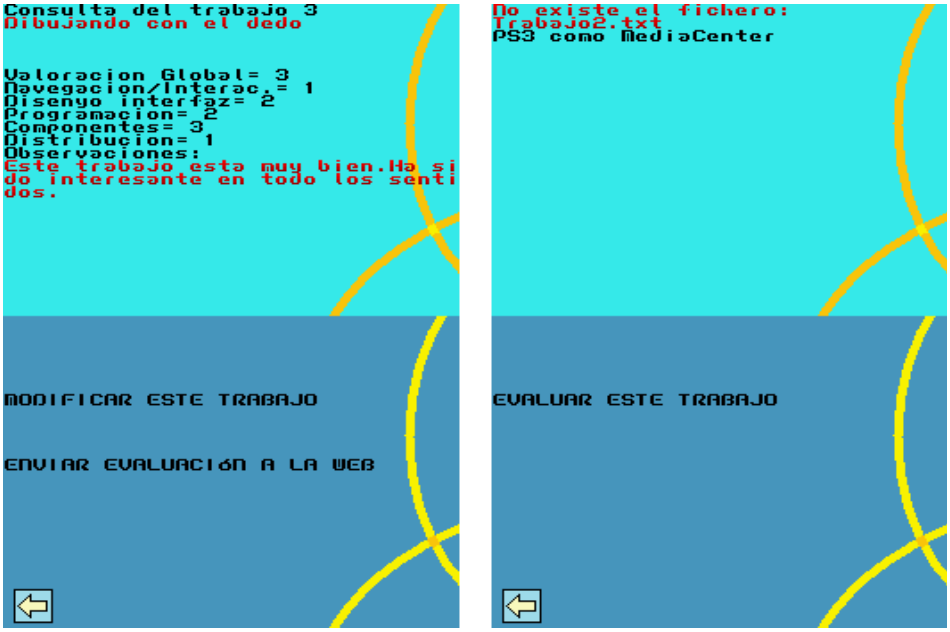

Figura 65: Partes que crea consultar.c

wifi.c

Aquí la aplicación realiza una descarga del fichero **TurnosAsignados.txt** desde la web vía WFC desde donde está almacenada y la deposita dentro de la tarjeta flashcard, en el mismo directorio donde se encuentre el ejecutable. En el caso de mi código, como mostraba la **figura 48**, está en **web-sisop.disca.upv.es/~numarso/TurnosAsignados.txt.**

Para realizar esta tarea, se ayuda de Ky_geturl.c, que es la encargada directa de realizar la conexión a la URL. La **figura 66** muestra visualmente esta parte.

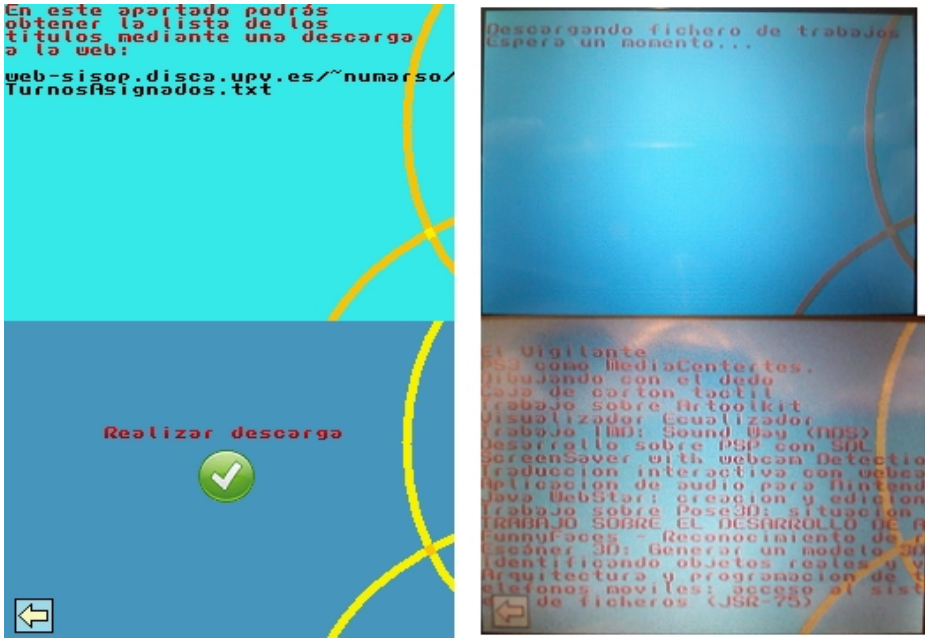

Figura 66: Partes que crea wifi.c

input.c

Sólo es el encargado de mostrar por pantalla el teclado virtual. Más abajo se encuentra comentados los fragmentos de código del teclado y del reconocimiento original que vienen en los ejemplos de PAlib.

CargaSistema.c

Es el gran encargado de leer el fichero de **TurnosAsignados.txt,** y meter la información en una matriz, que es una variable global, *char LineasFichero[40][100].* Cuando valorar.c llama a la función *int leerFichero(bool *lectura),* es ésta la que se ejecuta una única vez durante la aplicación, por lo tanto, siempre que se quiera obtener información se acudirá a la matriz, gracias a otras funciones que se encargan de ello.

También es el que proporciona el número de trabajos que existen en el fichero, además de que es el que se encarga de colorear de rojo los títulos cuando los seleccionamos en el menú.

Tiene la función *bool muestraConsulta(int opcion)*, que se encarga de leer y mostrar las evaluaciones ya realizadas dentro del menú de consulta.

Si TurnosAsignados.txt no existe en el directorio, es imposible pasar a la siguiente etapa de la evaluación.

CreaFichero.c

Éste por el contrario, es el encargado de escribir las evaluaciones en un nuevo fichero gracias a la función,

- void escribir(int opcion,int nota1,int nota2,int nota3,int nota4,int nota5,int nota6,char* text)

cuyo nombre será **TrabajoX.txt**, donde "X" es el número del trabajo elegido (*int opcion*) y también estarán en el directorio donde esté el ejecutable. A la hora de escribir en el fichero, se comprobó que no permite escribir acentos y el carácter "ñ", por lo que palabras como programación y diseño han pasado a ser "programacion" y "disenyo". La **figura 67** es una captura del fichero creado por la aplicación llamado **Trabajo4.txt**.

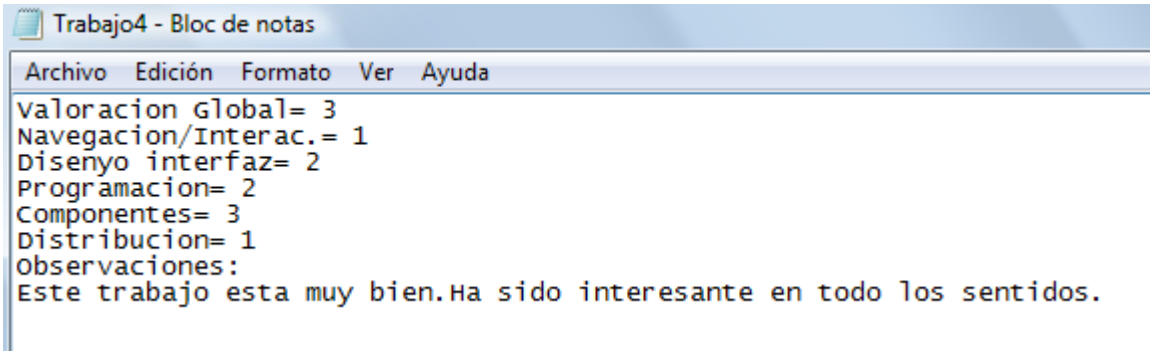

Figura 67: Trabajo3.txt

Luego si tiene una función en el que lee las evaluaciones *bool obtieneDatosConsulta(int opcion),* para obtener los datos de la evaluación y enviarla vía WFC a la web, desde el menú de consulta.

De la función que se encarga de enviar una evaluación a la web voy a explicar varias cosas de su código. Para empezar la cabecera, cuyos parámetros de entrada serían el número de trabajo seleccionado, las notas de la evaluación de cada aspecto y el comentario y como parámetro de salida la variable *int err*, por si hubiera error en el envío:

- int putHttp(int opcion,int nota1,int nota2,int nota3,int nota4,int nota5,int nota6,char* text);

Como ya habré comentado anteriormente, se ha necesitado ayuda de un servidor en PHP que tuve que implementar y que explicaré en el **apartado 9**. Realizando pruebas de envío, pude comprobar que cuando en el comentario había un espacio en blanco, ej. "hola mundo", el servidor enviaba hasta "hola", descartando lo que había después de un espacio. Entonces tuve que hacer un pequeño truco, que luego el servidor PHP se encargaría de corregirlo de nuevo, y es sustituir el carácter " " por "_" (ej. "hola_mundo").

```
char *espacio;

    do{
            espacio = strstr (text," ");
            strncpy (espacio,"_", 1);
    }while(espacio!=NULL);
```

Sobre el tema del envío de una evaluación, se ha hecho mediante la tranferencia de variables por medio de la URL. Entonces, como la URL a enviar iba a ser siempre dinámica, se ha realizado de dicha manera:

```
char url[350] ;
int err;
sBuffer bfr;

…

sprintf(url, "web-sisop.disca.upv.es/~numarso/recibir_fichero.php?
num=%d&var[]=%d&var[]=%d&var[]=%d&var[]=%d&var[]=%d&var[]=
%d&var[]=%s", opcion, nota1, nota2, nota3, nota4, nota5, nota6,
text);

…

err = ky_GetUrl(url, &bfr);
```

Y ya el servidor se encarga del resto. La **figura 68** representa el envío final de las evaluaciones a la web gracias al servidor.

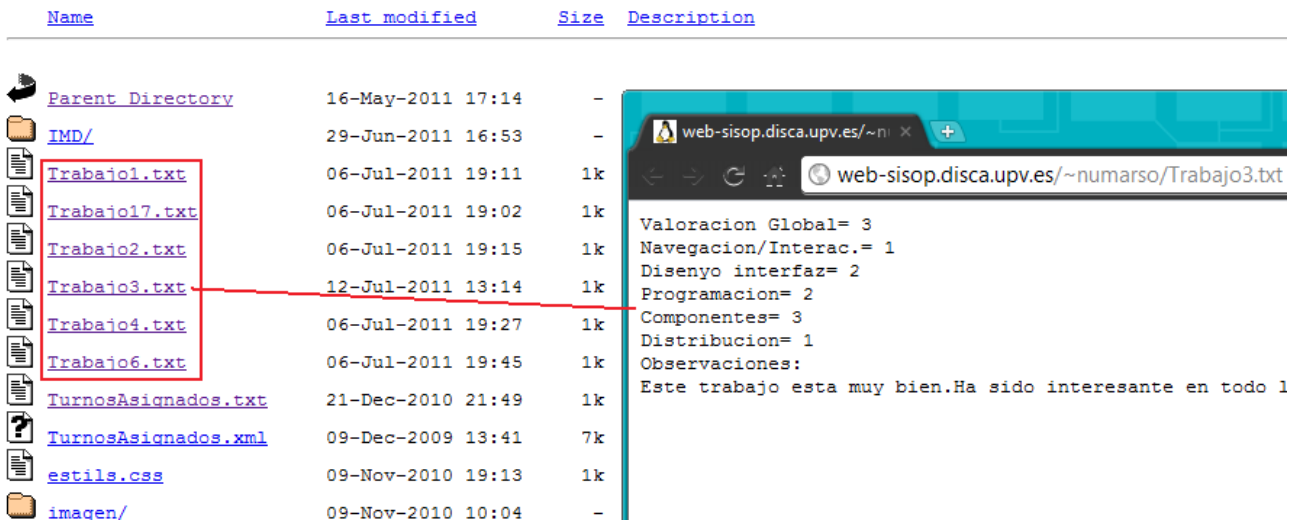

Figura 68: Trabajos enviados desde la aplicación a la web

ky_geturl.c

Es el mismo código que se encuentra en el ejemplo de DownloadFile de PAlib. Realiza las gestiones de transferencia de datos, tanto de descarga como de subida de los ficheros, además de hacer numerosos controles de errores. La función principal es:

```
int ky_GetUrl (const char *url, sBuffer *bfr)
{
     sLink lnk;
     int err;

     ky_InitBuffer(bfr);

     err = ky_InitLink(&lnk, url);
     if (err != ky_SUCCESS) {
          ky_FreeLink(&lnk);
          return err - 30;
     }

     err = ky_Connect(&lnk);
     if (err != ky_SUCCESS) {
          ky_FreeLink(&lnk);
          return err - 20;
     }
     err = ky_GetUrlEx(&lnk, bfr);
     ky_FreeLink(&lnk); /* FreeLink calls Disconnect */
     return err;
}
```

Primero se encarga de inicializar el buffer, luego con *ky_InitLink()* añade a la URL lo necesario para realizar la petición HTTP, intenta conectar con *ky_connect()* y finalmente realizar la transferencia. La variable *err*, es el error principal que pueda dar la ejecución, si todo va bien será 0.

9 Servidor PHP

Para lograr transferir ficheros desde la Nintendo DS hasta la web, se ha necesitado ayuda de un servidor en PHP (*Personal Home Page Tools*), que es un **lenguaje de de programación interpretado**, diseñado generalmente para la creación de páginas web dinámicas.

En versiones inferiores a la 4, no se puede tratar con archivos. Pues se ha realizado con la versión 5. Como ya se habrá comentado, para lograr la transferencia se ha introducido las variables como parámetros dentro del enlace hipertexto de la página destino. Ejemplo de la sintaxis para lograr esto sería:

```
<a      href="destino.php?variable1=valor1&variable2=valor2&...">Mi
enlace</a>
```

Respecto el servidor que he implementado, cuyo nombre es **recibir_fichero.php**, decir que

```
$num = $_GET['num'];
$variables = $_GET['var'];
```

son las encargadas de coger los parámetros de la URL y añadirlas al array *variables,* donde *num* es el número del trabajo enviado que se usa para crear el nombre definitivo del fichero a crear, y *var es* el contenido de la evaluación (aspectos y observación).

El servidor también realiza control de errores, en el caso de que no se haya introducido el número del trabajo, también comprueba si es un número, luego que los argumentos sean válidos y si ha habido un error al realizar el fopen. Es importante que en la web haya permisos de escritura para que el servidor haga su trabajo con éxito.

Código del servidor:

```
<?php

$opciones = array("Valoracion Global= ", "Navegacion/Interac.= ", "Disenyo
interfaz= ", "Programacion= ", "Componentes= ", "Distribucion= ",
"Observaciones:\n");

$num = $_GET['num'];
$variables = $_GET['var'];

if(empty($num) || !is_numeric($num)) die("No se ha introducido el número del
trabajo.");
if(count($variables) != count($opciones)) die("Argumentos incorrectos.");

$handle = fopen("Trabajo" . $num . ".txt", "w");

if(!$handle) die("No se puede crear el archivo.");

for($i=0; $i<count($variables); $i++){
    if($i == count($variables)-1){
```

```
        $comment = str_replace("_", " ", $variables[$i]);
        $variables[$i] = $comment;
    }
      fwrite($handle, $opciones[$i] . $variables[$i] . "\n");
}

fclose($handle);

?>
```

En `if($i == count($variables)-1)` del bucle, comprueba que el argumento a tratar es el comentario de texto de la evaluación, y realiza la sustitución de "_" a " " para arreglar lo que se había comentado en el código de CrearFichero.c del **apartado 8.**

10 Conclusión

Para ir acabando, expondré la experiencia que he obtenido al realizar el proyecto con sus más y sus menos sobre Nintendo DS y el concepto de Wi-Fi. Además, aunque la aplicación fuera desarrollada en una versión anterior, el código del proyecto se empezó a implementar desde cero totalmente, reutilizando código de forma muy puntual de la antigua, ya que la manera de enfocar el problema cambió radicalmente.

De cara a la implementación de la aplicación, se ha conseguido realizar las mejoras que se estimaban en un principio y que la versión antigua necesitaba de cara al usuario. La interfaz gráfica añadida con sprites y fondos diseñados de manera personal han fortalecido más el conocimiento hacia imágenes digitales, a la hora de saber los formatos a usar, colores máximos, trabajar con una paleta de color, etc. Además, gracias a los sprites, la interacción usuario-máquina es más intuitiva.

La navegación ha sido otro éxito, pues la versión antigua lo hacía todo de manera secuencial sin permitir al usuario rectificar en caso de error. Ahora, el usuario tiene total libertad de moverse por la aplicación y si es durante una evaluación, los datos se van manteniendo si han llegado a ser modificados.

El teclado virtual y el reconocimiento de PAlib llegaban a ser bastante simple y de manejo también secuencial. Al añadirle un cursor, también se ha conseguido facilitar la escritura para moverse por el texto y rectificar letras mal escritas sin tener que borrar todo el comentario para corregirla.

Se ha conseguido, generalmente, aprovechar al máximo la pantalla táctil de la consola gracias a los sprites y al crear áreas para reconocer cuando se selecciona por medio de tocar texto. También el usar de forma intuitiva el pad, como el botón A para aceptar y el botón B para cancelar, y el uso de la cruceta.

Lo más duro de del proyecto ha sido la parte de Wi-Fi, pues aunque estoy contenta por haber logrado realizar transferencia de datos, tanto descarga como subida de ficheros pero con el inconveniente de no poder usar transferencia de datos más de una vez, obligando a apagar y encender la aplicación.

Y a lo que la librería dswifi se refiere, la experiencia ha sido dura y ha dado muchos problemas, pues debido a la poca documentación existente, por no decir casi nula y su complejo código, obtener información cuesta mucho tiempo y dedicación. Además de que varios ejemplos no han funcionado, sobretodo a la hora de realizar conexiones a puntos de acceso vía DHCP.

También ha dado problemas los distintos firmware que existen para la consola, pues buscar uno compatible con todo me dedicó su tiempo de búsqueda y de pruebas, al final siendo la más óptima para todo *Wood R4 v1.29*.

Como conclusión final, creo que trabajar con Nintendo DS depende mucho del hardware (router compatible, flashcard, versión consola...) y de las librerías utilizadas, además de que veo la librería dswifi como la más compleja de utilizar. Decir también que he crecido profesionalmente con este proyecto y que me ha gustado mucho la experiencia.

Agradecer a Manuel Agustí Melchor, director del proyecto, por toda la motivación, ayuda aportada y dedicación durante el proyecto final de carrera.

11 Trabajos futuros

De cara a la aplicación, se proponen una serie de ampliaciones para obtener mejoras de la funcionalidad del trabajo de una manera más optimizada.

- Poder realizar, mientras que la aplicación esté funcionando, más de una transferencia de datos, además de analizar el error.
- Conseguir realizar la configuración Wi-Fi hacia un punto de acceso desde la propia aplicación sin tener que depender de nada externo, como lo hace el programa de DSOrganize., además de explicar cómo.
- Estudiar otras aplicaciones que trabajen con Wi-Fi como Wifi Setup.
- La posibilidad de que las URL se lean de un fichero INI en la SD.
- Trabajar con librerías para tratar XML, ya sea libxml2, mxml, ezxml... y obtener los títulos de los trabajos de TurnosAsignados.xml, además de escribir las evaluaciones en otro XML.
- Probar el proyecto en Linux y comprobar que todo funciona correctamente.
- En este trabajo se ha dado de lado a lo que al sonido se refiere. Como trabajo futuro no estaría demás añadir sonidos y alguna música de fondo para hacer de la aplicación más amena.

12 Bibliografía

- Wikipedia: Nintendo DS
http://es.wikipedia.org/wiki/Nintendo_DS
http://es.wikipedia.org/wiki/Nintendo_DS_Lite

- Diferencia entre modelos de DS
http://www.qvideojuegos.com/diferencias-entre-distintos-modelos-de-nintendo-ds.html

- Hardware Nintendo DS
http://kanads.blogspot.com/2006/12/arquitectura-de-nintendo-ds.html
http://kanads.blogspot.com/2006/12/arquitectura-de-nintendo-ds-ii.html
http://www.nintendo.es/NOE/es_ES/especificaciones_tcnicas_1184.html
http://www.nintendo.es/NOE/es_ES/systems/datos_tcnicos_674.html
http://www.ndsemulator.com/nds-emulator.htm

- FlashCard
http://nds.scenebeta.com/tutorial/cual-es-la-mejor-flashcard-para-nintendo-ds
http://nds.scenebeta.com/noticia/wood-r4

- Información general para uso de “homebrew” (librerías, tutoriales, homebrew, foros...)
http://nds.scenebeta.com

- Documentación y tutoriales de PAlib
http://palib.info/wiki/doku.php?id=homepage
http://www.espalteam.com/foros/showthread.php?s=43d1e552a5f911b2d886d77ebaf45290&p=997#post997
http://palib-dev.com/manual/main.html
http://www.elotrolado.net/hilo_tutorial-palib-como-utilizar-el-wifi_1019685

- Documentación libnds
http://libnds.devkitpro.org/
http://puyover.netau.net/proyectos/Apuntes%20NDS.pdf

- Graffiti (Palm OS)
http://es.wikipedia.org/wiki/Graffiti_(Palm_OS)

- DevkitPro
http://devkitpro.org/

- Code::Blocks
http://www.codeblocks.org/downloads/26
http://niozero.frostdisk.com/2008/08/tutorial-codeblocks-devkitpro.html

- Emulador Winds
http://windsprocentral.blogspot.com/p/winds-pro.html

- Tutorial DS2key
http://www.scenespain.net/foro/tutoriales-nds/(tutorial)-ndsndsl-como-pad-de-pc-con-ds2key/

- DSOrganize
http://www.dragonminded.com/ndsdev/dsorganize/

- Configurar conexión Wi-Fi vía WFC
http://www.nintendo.com/consumer/wfc/en_na/ds/connect.jsp
http://www.nintendo.com/consumer/downloads/NWFC_spanish.pdf
http://nds.scenebeta.com/tutorial/lista-de-errores-wifi-en-nds
http://nds.scenebeta.com/tutorial/nintendo-ds-y-wifi-primer-contacto

- Dswifi
http://www.akkit.org/dswifi/index.html
http://www.1emulation.com/forums/index.php?showforum=81
http://code.google.com/p/inferno-ds/source/browse/trunk/ethernds.c?r=192
http://wiki.lidsol.org/index.php?title=(libnds)_How_to_set_up_dswifi

- Lista de ejemplos Wi-Fi
http://www.ds-xtra.com/Category:DS_Wi-Fi_Homebrew

- Tutorial funcionamiento WIFI link
http://www.elotrolado.net/hilo_tutorial-funcionamiento-del-wifi-link_641301
http://bocabit.elcomercio.es/tutorial/tutorial-como-configurar-wifi-max-en-windows-vista.php
http://www.mayflash.com/pc/pc041/001.htm

- Información PHP
http://es.wikipedia.org/wiki/PHP
http://es.php.net/manual/en/function.fopen.php
http://www.desarrolloweb.com/articulos/317.php

- libxml2, mxml y ezxml
http://xmlsoft.org/
ftp://ftp.zlatkovic.com/libxml/
http://crysol.org/es/node/522
http://mxml.sourceforge.net/
http://ezxml.sourceforge.net/

Anexo I

Aquí voy a explicar la manera de configurar el Wi-Fi en Nintendo DS para ser usado vía WFC. Para llevar a cabo la tarea es necesario disponer de:

- Una conexión a Internet de banda ancha, como cable o DSL.
- Un router inalámbrico o un Conector USB Wi-Fi de Nintendo.
- Un juego compatible con la Conexión Wi-Fi de Nintendo.

Dentro del juego hay que entrar en la pantalla de configuración de la Conexión Wi-Fi de Nintendo. Son dos las opciones de conexión: mediante la función de búsqueda automática o bien manualmente

A continuación puedes elegir entre distintas opciones de conexión. La forma más sencilla de acceder a la Conexión Wi-Fi de Nintendo es mediante la función de búsqueda automática de la Nintendo DS.

Búsqueda automática

Primero entrar en ajustes de las conexiones Wi-Fi, seleccionar una ranura cualquiera y darle a "buscar un punto de acceso".

Figura 69.1: Configuración vía WFC

Entonces la consola mostrará los nombres de los puntos de acceso inalámbricos disponibles en la zona. Al lado de cada PA se verá un dibujo de un candado y la cobertura, siendo tres las posibilidades que muestra el color del candado.

Figura 69.2: Configuración vía WFC

El color azul, tal como muestra un candado abierto, significa que es una conexión abierta sin seguridad, por lo tanto se puede acceder a ella. El candado cerrado de color rojo significa que el router inalámbrico de la zona Wi-Fi tiene la seguridad WEP activada y ésta es necesaria para poder permitir su acceso. Finalmente, el candado gris significa que la configuración de seguridad del router no es compatible con la WFC de Nintendo.

Configuración manual

En el caso de obtener en casa un router inalámbrico con seguridad WEP se deberá optar por la segunda opción de configuración vía WFC de Nintendo. Siguiendo los mismo pasos de la **figura 66.1** , ahora seleccionando "configuración manual".

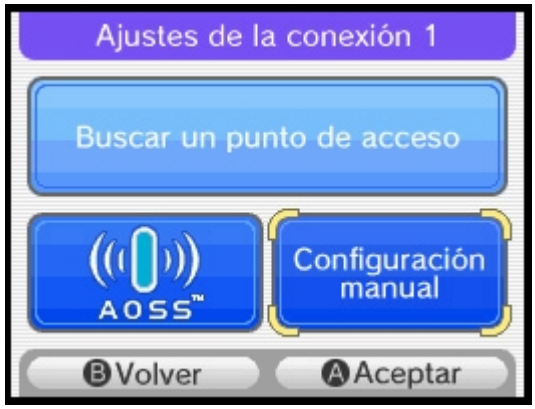

Figura 69.3: Configuración vía WFC

Aparecerá la siguiente lista de opciones de configuración donde se deberá de tocar el icono "Editar" para cada valor.

Figura 69.4: Configuración vía WFC

El SSID o Service Set Identifier (Identificador de conjunto de servicio) es el nombre que se asigna al punto de acceso de la red inalámbrica tal como aparece en el router. configurado.

La clave WEP es una opción de seguridad del router. WEP es la única seguridad compatible con la Conexión Wi-Fi de la Nintendo DS. Para usar la Nintendo DS en este punto de acceso, se tiene que configurar el router para la seguridad WEP o bien eliminar la seguridad del router. Se tendrá que introducir una clave WEP en tu Nintendo DS para usar el punto de acceso, en el caso de que no tenga seguridad, este campo se deja en blanco.

En algunos casos esta es toda la información manual que se necesita en un principio. Si tocamos "Probar conexión", al principio de la lista de entradas, se confirmará si la DS logra conectarse. Si la conexión funciona, entonces se puede dejar “Obtener dirección IP automáticamente” y “Obtener DNS automáticamente“ en **Sí,** finalmente guardar la configuración.

Errores

Durante la configuración o por problemas técnicos, cabe la posibilidad de que no se establezca la conexión deseada. Para ello, la consola siempre muestra un mensaje de error con una explicación del posible problema acompañado de un número identificador (rango desde 10000 hasta 98020). Existe una larga lista de errores, voy a poner algunos ejemplos de los más comunes por los usuarios:

- **20102**: Existe un problema con tu identificación (la ID de la Conexión WFC de Nintendo). Borra los datos de tu identificación y vuelve a crear una nueva.
- **30000:** La consola no esta aún configurada para conectarse a ningún punto de acceso.
- **50000 – 50099:** La consola no encuentra ningún punto de acceso.
- **51099:** Ninguno de los puntos de acceso encontrados es compatible con la consola. Si se ha hecho una configuración manual, también puede ser que la SSID es incorrecta.
- **51300 – 51302:** Puede deberse a varias causas:
  - El código WEP que has introducido en la consola no es correcto.
  - El router tiene un código de seguridad, pero no es WEP, sino otro distinto (WPA).
  - El router tiene activado un Filtro MAC de seguridad. Tienes que desactivar dicho filtro, o configurarlo para que permita a la consola conectarse (añadiendo la MAC de la consola a la lista de MAC permitidas por el filtro).
  - El router esta funcionando en modo 11g exclusivamente (“Mode 11g Only”).

Anexo II

En el código fuente de la librería dswifi, se puede encontrar el cómo se accede a los registros donde se encuentra la información de la conexión WFC de Nintendo, dentro del fichero **wifi_arm7.c**. La función encargada es:

- GetWfcSettings();

En el source de la carpeta common, se encuentra **wifi_shared.h** (ya nombrado en el **apartado 7.3** con DSOrganize, que también lo incluye), donde podemos encontrar la estructura principal que maneja el Wi-Fi y que usa el ARM7.

```
typedef struct WIFI_MAINSTRUCT {
     unsigned long dummy1[8];
     // wifi status
     u16 curChannel, reqChannel;
     u16 curMode, reqMode;
     u16 authlevel,authctr;
     vu32 flags9, flags7;
     u32 reqPacketFlags;
     u16 curReqFlags, reqReqFlags;
     u32 counter7,bootcounter7;
     u16 MacAddr[3];
     u16 authtype;
     u16 iptype,ipflags;
     u32 ip,snmask,gateway;

     // current AP data
     char ssid7[34],ssid9[34];
     u16 bssid7[3], bssid9[3];
     u8 apmac7[6], apmac9[6];
     char wepmode7, wepmode9;
     char wepkeyid7, wepkeyid9;
     u8 wepkey7[20],wepkey9[20];
     u8 baserates7[16], baserates9[16];
     u8 apchannel7, apchannel9;
     u8 maxrate7;
     u16 ap_rssi;
     u16 pspoll_period;

     // AP data
     Wifi_AccessPoint aplist[WIFI_MAX_AP];

     // probe stuff
     u8 probe9_numprobe;
     u8 probe9_ssidlen[WIFI_MAX_PROBE];
     char probe9_ssid[WIFI_MAX_PROBE][32];

     // WFC data
```

```
     u8 wfc_enable[4]; // wep mode, or 0x80 for "enabled"
     Wifi_AccessPoint wfc_ap[3];
     unsigned long wfc_config[3][5]; // ip, snmask, gateway,
primarydns, 2nddns
     u8 wfc_wepkey[3][16];


     // wifi data
     u32 rxbufIn, rxbufOut; // bufIn/bufOut have 2-byte
granularity.
     u16 rxbufData[WIFI_RXBUFFER_SIZE/2]; // send raw 802.11 data
through! rxbuffer is for rx'd data, arm7->arm9 transfer

     u32 txbufIn, txbufOut;
     u16 txbufData[WIFI_TXBUFFER_SIZE/2]; // tx buffer is for data
to tx, arm9->arm7 transfer

     // stats data
     u32 stats[NUM_WIFI_STATS];

     u16 debug[30];

    u32 random; // semirandom number updated at the convenience of
the arm7. use for initial seeds & such.

     unsigned long dummy2[8];

} Wifi_MainStruct;
```

Y el código de *GetWfcSettings()*, donde con cada bucle hace los tres recorridos posibles (tres conexiones WFC por consola) de cada configuración. Donde se aprecia cómo el ARM7 trabaja con la estructura anterior.

- volatile Wifi_MainStruct * WifiData = 0;

```
void GetWfcSettings() {
     u8 data[256];
     int i,n, c;
     unsigned long s;
     c=0;
     u32 wfcBase = ReadFlashBytes(0x20, 2) * 8 - 0x400;
     for(i=0;i<3;i++) WifiData->wfc_enable[i]=0;
     for(i=0;i<3;i++) {
          readFirmware( wfcBase +(i<<8),(char *)data,256);
          // check for validity (crc16)
          if(crc16_slow(data,256)==0x0000    &&    data[0xE7]==0x00)
{ // passed the test
               WifiData->wfc_enable[c] = 0x80 | (data[0xE6]&0x0F);
```

```
                WifiData->wfc_ap[c].channel=0;
                for(n=0;n<6;n++) WifiData->wfc_ap[c].bssid[n]=0;
                for(n=0;n<16;n++)            WifiData->wfc_wepkey[c]
[n]=data[0x80+n];
                for(n=0;n<32;n++)                              WifiData-
>wfc_ap[c].ssid[n]=data[0x40+n];
                for(n=0;n<32;n++) if(!data[0x40+n]) break;
                WifiData->wfc_ap[c].ssid_len=n;
                WifiData->wfc_config[c][0]=((unsigned    long    *)
(data+0xC0))[0];
                WifiData->wfc_config[c][1]=((unsigned    long    *)
(data+0xC0))[1];
                WifiData->wfc_config[c][3]=((unsigned    long    *)
(data+0xC0))[2];
                WifiData->wfc_config[c][4]=((unsigned    long    *)
(data+0xC0))[3];
                s=0;
                for(n=0;n<data[0xD0];n++) {
                     s |= 1<<(31-n);
                }
                s=   (s<<24)   |   (s>>24)   |   ((s&0xFF00)<<8)   |
((s&0xFF0000)>>8); // htonl
                WifiData->wfc_config[c][2]=s;
                c++;
           }
     }
}
```

Anexo III

El servidor java fue realizado por unos compañeros de clase, para otro trabajo de Nintendo DS. Este servidor hace lo propio pero está modificado por Manuel Agustí, del cual me he ayudado para realizar pequeñas pruebas para probar los ejemplos.

```
import java.net.*;
import java.io.*;
import java.util.*;

/*  Comentaris:
  Servidor d'eco (TCP) mínim, tret de pràctiques de Xarxes o
similar,
 Compilar i executar en
 $ javac SCTP.java
 $ java SCTCP

 Llançar dos, mínim, telnets contra el servidor
 $ telnet localhost 7077

 Modificacions:
 (C) M. Agustí.
  He ficat codic per:
    - Fer eco en el servidor
    - Dir el número de conexions
    - Identificar a qui envia el mensatge.
   Pero deuria ser una cadena de caracters (apodo o nick name) o
similar que se demanara al conectar.
*/

    //http://stackoverflow.com/questions/197986/what-causes-javac-
to-issue-the-uses-unchecked-or-unsafe-operations-warning
@SuppressWarnings("unchecked")
public class SCTCP2 extends Thread
{
    Socket id;
    static ArrayList lista = new ArrayList();

    public SCTCP2(Socket s) { id = s; }
    public void publicar(String s)
    {
          //aqui publicaremos las linea recibida en todos los del
arraylist
        // por ahora lo reciben todos, no se que error tiene el if
ni el pintwriter
        for(int i = 0; i < lista.size(); i++)
        {
        //un if posible
            Socket saux;
            saux=(Socket)lista.get(i);
```

```
                                try{   PrintWriter   salida   =   new
PrintWriter(saux.getOutputStream(), true);
            //if (lista.get(i)!= this.id)    salida.println(s);
            if(!saux.equals(id))
             {
/*
              salida.print(i);
              salida.print(">>");
*/
              salida.println(s);
             }
/*
            // No funciona per diferenciar de qui ve el mensatge
if (!saux.equals(id))
{
//salida.printf("conexio %d: %s.\n", this.id, s);
salida.printf("conexio ");
salida.print(i);
salida.print("  - ");
salida.println(s);
}
*/
        }
        catch(Exception e) { }
        }
                                                                    }

    public void run()
    {
        int seguimos = 1;
        String linea,mayusculas = "";
        try
        {
                 BufferedReader entrada = new BufferedReader(new
InputStreamReader(id.getInputStream()));
                                    PrintWriter   salida   =   new
PrintWriter(id.getOutputStream(), true);
            //salida.printf("Estas conectado al servidor\n");
                salida.printf( String.format( "Estas conectado al
servidor: benvingut %d.\n", id.getPort()) );
            //publicar("GOING ON");
                 publicar( String.format( "Avis a clients: %d ha
entrat", id.getPort()) );

            while(seguimos == 1)
            {
                    linea = entrada.readLine();   //la cazamos la
inicial
                System.out.print(id.getPort());
                System.out.printf("-- %s.\n", linea);
```

```
                mayusculas = linea.toUpperCase();
                if(mayusculas.startsWith("QUIT")) break;
                //aquí trataremos el error
                else
                   //publicar(linea);
                                publicar(  String.format(  "%d>  %s",
id.getPort(), linea));

            }
              salida.println("uno fuera");
              id.close();
              lista.remove(id);//deberia borrar aki de la lista
              System.out.printf("Conexiones %d.\n", lista.size());

            //publicar("LEFT OUT");
                          publicar(  String.format(  "%d>  abandona",
id.getPort()));
        }
        catch(Exception e)
        {
            e.printStackTrace();
              //salida.println("debes de estar conectado para que
esto funcione.");

        }
    }
    public static void main(String arg[]) throws IOException{
        ServerSocket ss = new ServerSocket(7077);
        System.out.println("Socket llansat sobre 7077.");

       while(true)
        {
            Socket s = ss.accept();
            SCTCP t = new SCTCP(s);
            if(lista.size() < 100)
            {
              lista.add(s);//agregamos el cliente y seguimos
//SCTCP.java:68: warning: [unchecked] unchecked call to add(E) as
a member of the raw type java.util.ArrayList
              System.out.printf("Conexiones %d.\n", lista.size());
              t.start();
            }
            else s.close();
        }
    }
}
```