Gestión del sonido en la implementación del tutorial "The Bouncing Ball" para NDS

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

1 Resumen de las ideas clave

En este artículo, vamos a recorrer parte de la creación de Mukunda Johnson [1], quien realizó desarrollos *homebrew (los que utilizan* el kit de desarrollo no oficial) muy interesantes para Nintendo DS (NDS). Entre otras cosas, realizó un tutorial[1] sobre el desarrollo de aplicaciones para NDS que permite iniciarse en este complejo mundo. Sobre el trabajo de Johnson, Jaén [2] realizaría un trabajo de actualización y ampliación del tutorial original, al tiempo que se corrigen algunas erratas del texto original, algunos cambios del código debidos a modificaciones de las librerías en que se basa y ampliando algunos apartados con pequeñas "demos" de conceptos.

Ya no es posible consultar el tutorial original en la red, así que el presente trabajo ofrece una versión conjunta de estos dos trabajos citados, actualizándolos a las versiones presentes de las librerías sobre las que se basan. Para facilitar el seguimiento de este trabajo se han dispuesto todos los elementos del proyecto que aquí se comenta en *GitHub* [3], junto a los documentos originales mencionados. Quiero rendir un homenaje a estos desarrolladores que han contribuido con su código, y con sus explicaciones, a que otros puedan adentrarse en este difícil campo de desarrollo de aplicaciones para videoconsolas, que son plataformas muy populares y diferentes del ordenador de sobremesa.

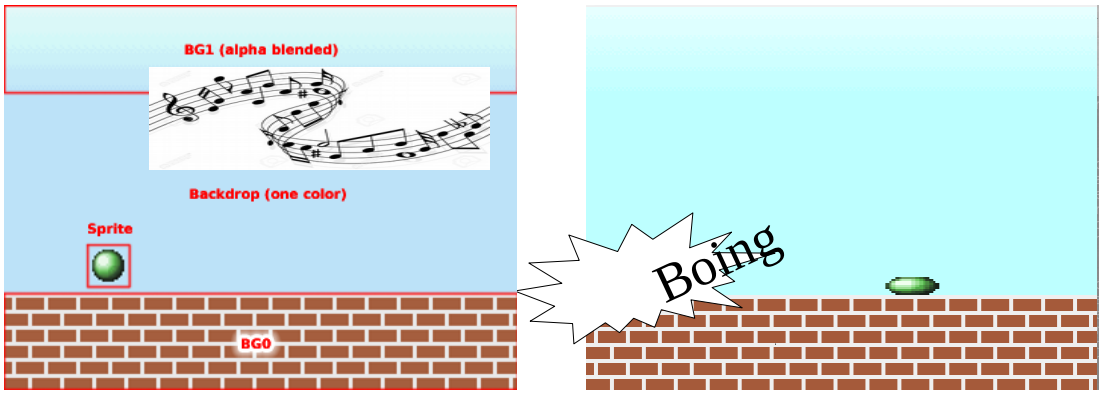

Figura 1: Esquema gráfico de las dos situaciones donde introducir elementos sonoros: (a) música de fondo y (b) un efecto puntual: rebote de la pelota.

Tomaremos como base para este artículo la aplicación para la NDS que explican los trabajos de [1] y [2]. En ella, sobre una escena con dos planos y utilizando la botonera de la NDS, se puede mover una pelota, a la que también afecta el efecto de la "gravedad" y el "rozamiento con el aire" que la hacen avanzar en una trayectoria casi parabólica y que, al tocar el suelo, sufre una deformación y rebota. **Nos ocuparemos en este artículo de** comentar las acciones necesarias para añadir a esa aplicación una música de fondo (a veces llamada banda sonora o BS) y un sonido puntual (clip de audio, efecto especial o FX); la Figura 1 muestra dos bocetos para ilustrar estas situaciones. Asumiremos que la aplicación ya está

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

desarrollada y nos centraremos en los aspectos concernientes al uso del audio como recurso.

2 Objetivos

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

- Será capaz de entender el proceso de transformación de los recursos de naturaleza sonora que se incorporan a una aplicación para la plataforma NDS.

- Podrá experimentar con un código que utiliza los conceptos básicos de desarrollo de aplicaciones de videojuego en la consola NDS, centrándose en el uso del sonido para esta plataforma.

No es un objetivo instalar las herramientas que permiten la creación del ejecutable y la carga del mismo, por lo que asumiremos que se dispone del emulador *DeSmuMe* [4]. Si está interesado en los detalles de estos aspectos se puede recurrir a los trabajos de [5] y [6]. Tampoco es un objetivo de este trabajo entrar a ver las características del hardware de la NDS como plataforma optimizada para tareas de videojuego. Si el lector tiene curiosidad o necesidad de profundizar en los elementos de la NDS, es recomendable consultar [7].

Antes de ver un ejemplo, hemos de hablar un poco de teoría, en concreto cómo se utiliza el audio en la NDS. Veamos qué consideraciones hay que tener en cuenta para poder generar sonidos a partir de ficheros en esta videoconsola.

3 Introducción

En el ámbito de los videojuegos se habla habitualmente de dos conceptos relativos al sonido en las aplicaciones que se desarrollan y que son:

- Sonidos. Que hace referencia a la reproducción de sonidos simples que pueden sonar en múltiples instancias y a la vez. Estos sonidos suelen cargarse completamente en memoria y descargarse cuando no se requieren.

- Música. Hace referencia al sonido del que solo hay una instancia en reproducción y que, por su duración, hay que llevar cuenta de sus requerimientos de memoria que serán altos.

3.1 Arquitectura de la NDS para soporte de audio

La NDS ofrece, para la salida de audio, 16 canales *hardware* independientes: esto es, que pueden funcionar con parámetros diferentes de frecuencia de muestreo, tamaño de muestra y número de muestras; y se pueden habilitar de manera separada, ser reproducidos en modo bucle o no, variar su volumen, etc. Todo ello con un sistema basado en 2 canales (estéreo) y balance (*panning*) entre ellos, que funciona a una frecuencia máxima de muestreo 22 Khz. También dispone de un micrófono que proporciona una señal monofónica, de 8 o 12 bits PCM.

El subsistema *hardware* de sonido de la NDS está compuesto por tres elementos. El primero se denomina *Simple Sound Engine* (SSE) que proporciona las operaciones básicas de manipulación de sonido (como volumen o balance) y el acceso al micrófono. Además, dispone de un *Programmable Sound Generator* (PSG), que proporciona la generación de formas de onda simples (o tonos) y, por último, también dispone de un "generador de ruido blanco" (*noise generator*) para sintetizar sonidos más complejos.

El formato nativo de sonido en la NDS se conoce habitualmente como *raw format,* está basado en muestras codificadas en forma de ondas, sin comprimir, en PCM o ADPCM, de 8 o 16 bits; dejando de la mano del software la reproduccion de formatos más complejos y el acceso a ficheros. Aquí veremos ejemplos de uso de archivos y como, de la mano de la librería *Maxmod* [1], se pueden reproducir formas de ondas (*sample-playing*), ficheros estructurados (*mod-playing*) y gestionar el modo *streaming* (buffers).

3.2 Alternativas para el uso del audio en la NDS

Existen algunas aplicaciones muy interesantes que hacen uso del audio de manera muy especializada como DSSpeech, AndroDaft o DSDaft y librerías[2] para decodificación de OggVorbis (como *Helix-OSDL* o *Tremor*) y de MP3 (como *libmad* y *Helix mp3dec*).

En nuestro caso, la elegida ha sido *MaxMod [1]* cuya aplicación de demostración (vease Figura 2) da muestras de su capacidad, para reproducir sonidos en formato WAVE o MIDI, cargarlos en memoria, alterar los parámetros o cancelar su reproducción.

[2] Puede encontrar documentación sobre ellas en < http://osdl.sourceforge.net/main/documentation/misc/nintendo-DS/homebrew-guide/HomebrewForDS.html>.

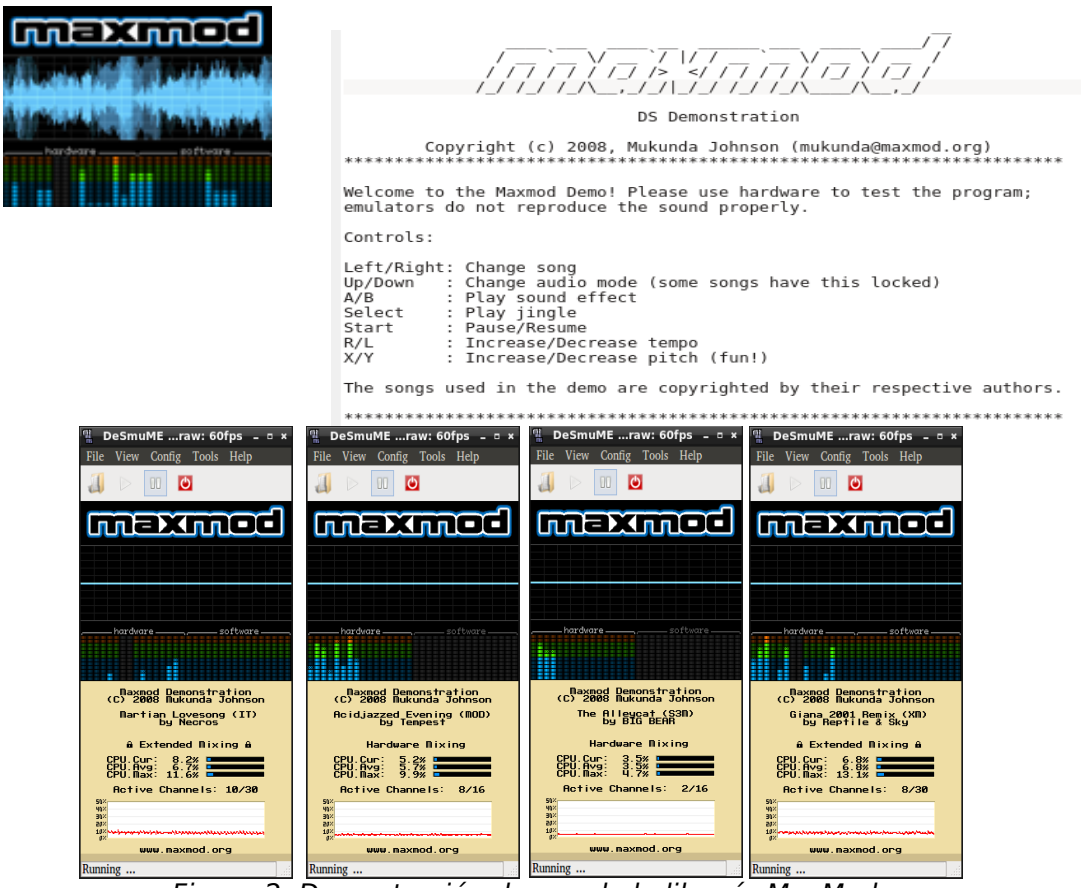

Figura 2: Demostración de uso de la librería MaxMod.

4 Estructura y contenido del proyecto

El proyecto tiene la apariencia visual que se muestra en Figura 3a y se estructura en la forma en que se muestra en la Figura 3b, con un subdirectorio para el código fuente (*source*), otro para los recursos gráficos en formato de mapas de bits (*graficos*) y otro para el audio (*sonidos*). La Figura 3c muestra la previsualización que ofrece el navegador de archivos de los ficheros de medios, para que apreciemos el medido uso de recursos de este ejemplo: los tres ficheros PNG son los únicos necesarios para el desarrollo de la escena.

Además, en el directorio del proyecto, hay un fichero *Makefile* que es el encargado de organizar el trabajo a realizar con los diferentes tipos de archivos para obtener el ejecutable final. Para ello, en él se configuran los subdirectorios que contiene cada recurso (código fuente, imágenes y sonidos) y las operaciones a realizar con cada herramienta para convertir y combinar todos los recursos en el fichero resultante final.

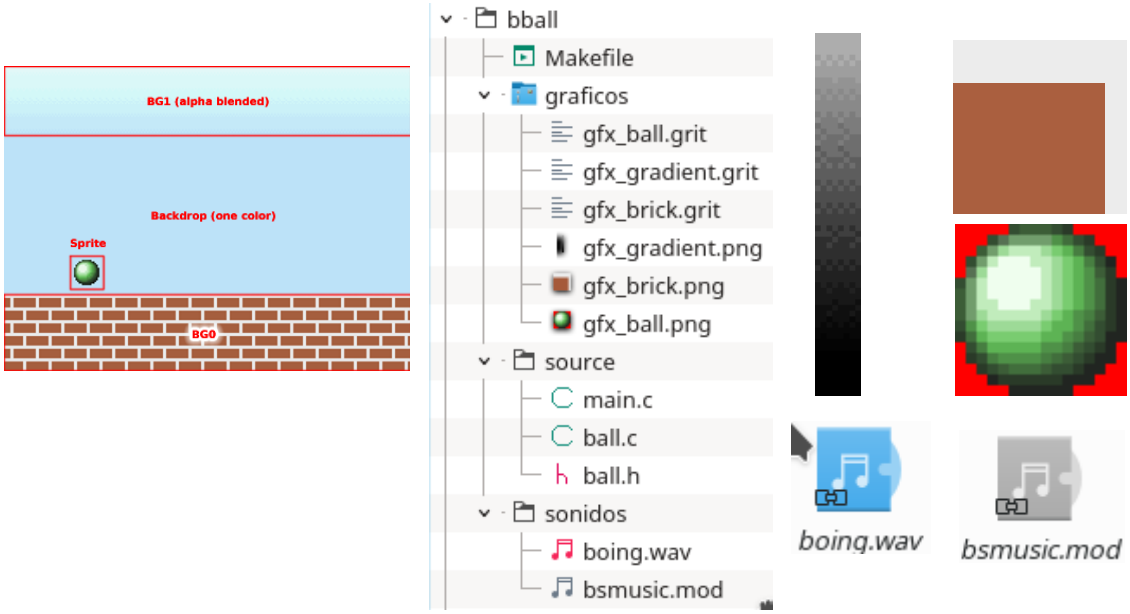

Figura 3: El proyecto: (a) imagen final del tutorial extraída de [2], (b) archivos que lo componen y (c) previsualización de los recursos gráficos y sonoros.

Los recursos sonoros están en el subdirectorio "sonidos" del proyecto, Figura 3b y se han obtenido de repositorios libres ([8] y [9]). Sus características son las que muestra la Figura 4. El fichero de sonido original no es posible utilizarlo en la NDS por sus altos requerimientos, especialmente la frecuencia de muestreo de 44 Khz que es mucho mayor que lo que soporta la consola. Por ese motivo, hay que convertirlo a unos parámetros más sencillos. Sin perder calidad para su propósito, con *Sox* [10] hemos obtenido una versión apta para ser utilizada y el lector interesado podrá probarlo, si tiene curiosidad.

```
$ file bball/sonidos/*
boing.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 8
bit, mono 10000 Hz
bsmusic.mod: 4-channel Protracker module sound data Title:
"elekfunk !"

$ file 140867__juskiddink__boing.wav
140867__juskiddink__boing.wav: RIFF (little-endian) data, WAVE audio,
Microsoft PCM, 24 bit, stereo 44100 Hz
$ sox 140867__juskiddink__boing.wav -t wav -r 10000 -c 1 -b 8
bball/sonidos/boing.wav
$
```

Figura 4: Comprobación de características y conversión del audio para el proyecto.

Con el uso del *Makefile*, se automatiza el resto del proceso de conversión del formato de los ficheros de audio a un formato de banco de sonidos que aglutina los diferentes sonidos en forma de ondas sin compresión,

que es soportado directamente por la NDS. Veamos ahora esta conversión.

4.1 De archivos de audio a banco de sonidos

La aplicación ***mmutil***[1] permite convertir los ficheros de audio que vamos a utilizar en un formato que la NDS puede leer. Puede trabajar a partir de ficheros de audio en formato de forma de ondas (WAVE) y de audio estructurado (MOD, S3M, XM e IT).

Para ello, cada vez que se modifiquen los ficheros de sonido (*boing.wav* y *bsmusic.mod*, en este caso), el *mmutil* los convertirá (como muestra la Figura 5) a un fichero binario (*sounbank_bin*) que contiene todos los sonidos que ha encontrado (en el directorio indicado en el proyecto), en formato *raw*. Además, creará los archivos de cabecera *sounbank.h* y *sounbank_bin.h*) en formato de C/C++ para declarar las variables que contienen los sonidos y sus características. Todo ello será enlazado junto con el código, que es una manera de importar los recursos sonoros en el ejecutable final.

Figura 5: Conversión de ficheros de audio con mmutil.

A partir de los nombres de ficheros de sonido, *mmutil* genera, en nuestro caso, los ficheros de cabecera (los ".h") con el contenido que se observa en la Figura 6. Esta figura muestra cómo se invoca a *mmutil* y qué se obtiene tras su ejecución. En ella se puede ver que [1] si hay ficheros modulares dan lugar a definiciones de variables con prefijo "MOD_". En caso de haber ficheros de forma de ondas, serán asignados a variables cuyo nombre se formará con el prefijo "SFX_". En ambos casos, el resto del nombre de la variable se obtendrá del nombre del fichero. En nuestro ejemplo solo tenemos una instancia de cada uno, así que vemos un *MOD_BSMUSIC* y un *SFX_BOING*.

Además, se crean una serie de definiciones para tener cuenta de cuántos elementos se han incorporado al banco de sonidos (*MSL_NSONGS* y *MSL_NSAMPS*) y su longitud (*MSL_BANKSIZE*). Finalmente vemos los nombres de la estructura de datos que contiene todo el sonido, el banco de sonidos (*soundbank_bin*), junto con otros dos vectores que contienen las direcciones, donde empieza cada elemento sonoro.

```
$ mmutil sonidos/bsmusic.mod sonidos/boing.wav -d -osoundbank.bin
-hsoundbank.h
$ cat build/soundbank.h
#define MOD_BSMUSIC      0
#define SFX_BOING        31
#define MSL_NSONGS        1
#define MSL_NSAMPS       32
#define MSL_BANKSIZE     33
$
$  cat build/soundbank_bin.h
extern const u8   soundbank_bin_end[];
extern const u8   soundbank_bin[];
extern const u32 soundbank_bin_size;
$
```

Figura 6: Ejemplos de conversión de sonidos con mmutil.

5 Desarrollo

Ya hemos dicho que los contenidos del tutorial original y las versiones del código actualizadas se han ido disponiendo en *GitHub* [3]. La versión final es la que contiene la operativa para gestionar el uso del audio y, esa parte, es la que se puede ver aquí separda en dos partes:

- El Listado 1 recoge dos grupos de acciones
  - Las que se ocupan de la carga de los recursos de audio (el banco es cargado con la instrucción de la línea 33) y de la inicialización de valores para la música y los sonidos (líneas 35 a la 39). También se podría haber configurado la reproducción en modo contínuo o bucle, el *pitch* del sonido, etc.
    Y la activación de la “banda sonora” o música de fondo, con la línea 41 de este Listado 1.
  - Las que liberan los recursos asociados, líneas 45 a la 47 de este listado, que se hace al final en este caso, pero se podría hacer en cualquier momento en que se sepa que no se van a utilizar.
- Por su parte, el Listado 2 muestra cómo se activan los efectos especiales de sonido al producirse los eventos que han de remarcar. En este caso, el sonido de rebote de la pelota, (línea 24 del Listado 2), seguirá sonando tanto como lo indique el fichero utilizar, como su duración es corta, no se notará mucho.
  Si consultamos la documentación, veríamos que también se pueden realizar otras acciones como comprobar si un sonido está en ejecución o detener cualquiera de los que estén en marcha.

```
1.    #include <gfx_ball.h>
2.    #include <gfx_brick.h>
3.    #include <gfx_gradient.h>
4.    #include "ball.h"
5.
6.    ball g_ball;
7.    void resetBall( void );
8.    void setupGraphics(int *bg0, int *bg1 );
9.    void updateGraphics(int bg0, int bg1 );
10.   void processLogic( void );
11.
12.   // Para el uso del sonido
13.   #include "soundbank.h"
14.   void inicializarSonidos( void );
15.   void liberarRecursosSonidos( void );
16.
17.   int main( void ) {
18.     int bg0, bg1;
19.
20.      setupGraphics( &bg0, &bg1 );
21.      resetBall();
22.      inicializarSonidos();
23.
24.      while(1) {
25.           processLogic();                    //Actualizacion objetos (moverlos,
velocidades, etc)
26.           swiWaitForVBlank();
27.           updateGraphics( bg0, bg1  );       //Pintar los  graficos
28.      }
29.      liberarRecursosSonidos();    // Descargar de la memoria los recursos de
audio
30.   } // Fi de main
...
31.
32.   void inicializarSonidos( void) {
33.    mmInitDefaultMem((mm_addr)soundbank_bin);  //Inicialiamos Maxmod
34.
35.      mmLoad(MOD_BSMUSIC);                    //Cargamos modulo de musica
36.      mmSetModuleVolume( 512 ); // New volume level. Ranges from 0 (silent)
to 1024 (normal).
37.
38.      mmLoadEffect( SFX_BOING );              //Cargamos sonido FX
39.      mmEffectVolume( SFX_BOING, 128 );    //Ranges from 0 (silent) to 255
(normal).
40.
41.      mmStart(MOD_BSMUSIC, MM_PLAY_LOOP);      //Comenzamos la musica de
fondo
42.   } // Fi de inicializarSonidos
43.
44.   void liberarRecursosSonidos( void ) {
45.      mmStop();
46.      mmUnload(MOD_BSMUSIC);
47.      mmUnloadEffect(SFX_BOING);
48.   }// Fi de liberarRecursosSonidos
```

Listado 1: Contenido nuevo en main.c *para poder utilizar el audio.*

```
1.      #include <nds.h>
2.      #include "ball.h"
3.
4.      #include <maxmod9.h>
5.      #include "soundbank.h"
6.
7.      void ballUpdate( ball* b ) {          // Físicas
8.       b->x += (b→xvel>>4);                              //Gravetat: apply air
friction to X velocity
9.       b->xvel = (b->xvel * (256-c_air_friction)) >> 8;  //Gravetat: apply air
friction to X velocity
10.      b->xvel = clampint( b->xvel, -max_xvel, max_xvel );     // clamp X velocity
to the limits
11.      b->yvel += c_gravity;                             // add gravity to Y velocity
12.      b->y += (b→yvel);                               // add Y velocity to Y position
13.
14.      if( b->y + c_radius >= c_platform_level ) {                 // Bounce on the
platform
15.       b->xvel = (b->xvel * (256-c_ground_friction)) >> 8;     // apply ground
friction to X velocity
16.        if( b->y > c_platform_level - min_height )    {    // the ball has been
squished to min. Height?
17.         b->y = c_platform_level – min_height;           // mount Y on platform
18.         // negate Y velocity, also apply the bounce damper
19.         b->yvel = -(b->yvel * (256-c_bounce_damper)) >> 8;
20.         // clamp Y to mininum velocity (minimum after bouncing, so the ball does
not settle)
21.         if ( b->yvel > -min_yvel ) b->yvel = -min_yvel;
22.        }
23.        b->height = (c_platform_level - b->y) * 2;                  // calculate the
height
24.        handFX = mmEffect( SFX_BOING );       // Hacer sonar para simular el
rebote de la pelota
25.      }
26.      else {
27.        b->height = c_diam << 8;
28.      }
29.     } // Fi ballUpdate
```

Listado 2: Contenido nuevo en ball.c para poder utilizar el audio.

6 Conclusiones y cierre

Con este artículo hemos vuelto a poner a disposición de los interesados en el desarrollo de aplicaciones *homebrew* sobre la NDS, un recurso que creemos es de utilidad para servir de apoyo a los que se inician en este complejo contexto de desarrollo de aplicaciones para plataformas de videoconsolas.

Las actualizaciones necesarias del código y la ubicación en un repositorio en *GitHub* [3], son nuestra modesta aportación. Quiero

agradecer desde aquí el trabajo de los autores que figuran en la bibliografía y especialmente el de J. D. Jaén con quien llevamos adelante el desafío de completar la parte inacaba con la incorporación de la música de fondo y el efecto de audio del rebote de la pelota. Sirva este trabajo como homenaje a estos desarrolladores que han contribuido con su código y con sus explicaciones a que otros puedan adentrarse en este apasionante y difícil campo de desarrollo de aplicaciones para computadores con una arquitectura tan especializada y poco documentada.

Esperamos que el lector se anime a descargar el proyecto desde el *GitHub* [3], leer el original y probar a ejecutar los desarrollos expuestos, por ejemplo, con el emulador *DeSmuMe*[4]. Y, por supuesto, experimentar y modificar el código que se publica en el repositorio del proyecto.

7 Bibliografía

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

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

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

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

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

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

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

[8] *Freesound*. Disponible en <http://freesound.org/>.

[9] *The Mod Archive* . Disponible en <https://modarchive.org/>.

[10] *Sound eXchange | HomePage*. Disponible en <http://sox.sourceforge.net/>.