UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Acceso directo a la memoria de vídeo en la plataforma NDS

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

1 Resumen de las ideas clave

En este artículo vamos a ver cómo se puede dibujar en las pantallas de la *Nintendo DS* (NDS) usando la técnica de *framebuffer* o acceso directo a la memoria de vídeo.

El ejemplo básico que nos inspira apareció en el sitio web de *Drunken Coders*[1] [1], un grupo de desarrolladores de la escena *homebrew* (esto es, que no utiliza el SDK oficial propuesto por el fabricante de la consola y con tremendas restricciones sobre su publicidad) para el desarrollo de aplicaciones para NDS. Esperamos que este trabajo sirva para reconocer el tremendo esfuerzo de estos grandes desarrolladores, máxime en las épocas en que no había nada construido. Su trabajo ha servido de inspiración y de fuente a muchos. ¡Gracias!

Ya no es posible consultar el tutorial original en la red, así que el presente artículo ofrece una versión de ese trabajo original, actualizada a las versiones existentes de las librerías sobre las que se basa. De todas formas, dejaremos los comentarios originales para que quede constancia del trabajo original.

Para facilitar el seguimiento de este trabajo se han dispuesto todos los elementos del proyecto que aquí se comenta en *GitHub* **[2]**. Nos ocuparemos en este artículo de comentar las acciones necesarias para dibujar en las pantallas de la NDS a nivel de píxel, así que átate el cinturón porque vamos a adentrarnos en una pequeña aventura espacial.

2 Objetivos

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

- Será capaz de identificar el concepto de una imagen en mapa de bits (*raster* o *bitmap*) con un modo de trabajo de la NDS.
- Habrá visto cómo identificar las instrucciones que constituyen el acceso directo a la memoria de vídeo de la plataforma NDS.
- Dispondrá de ejemplos de código que se proponen como ejercicios prácticos de iniciación al acceso a los píxeles de la pantalla.

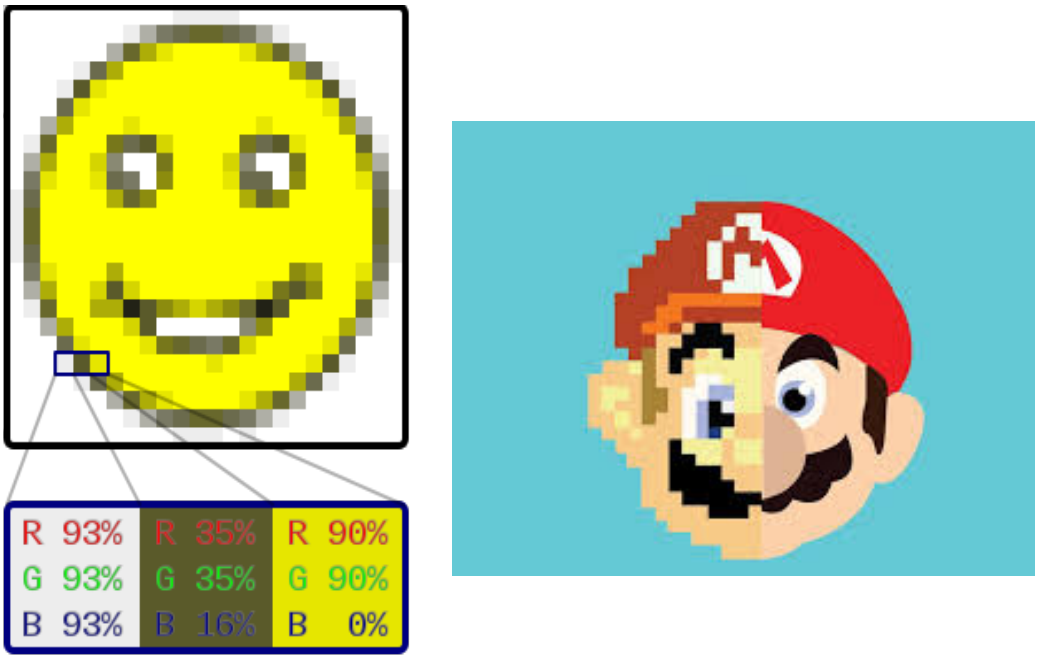
No es un objetivo instalar las herramientas que permiten la creación del ejecutable, así como la carga del mismo en la consola, de hecho, nosotros utilizaremos un emulador (*DeSmuMe* [3]) para generar las capturas. Para ello se puede recurrir a los trabajos de [5] o [6].

---

[1] Publicado originalmente en <http://drunkencoders.com/>, esta URL ahora alberga solo un enlace a un contenido diferente al tutorial en que nos hemos inspirado. En *Github* <https://github.com/drunken-coders> ha aparecido un repositorio con la leyenda “Breaking new ground in console homebrew development”. Esperamos que puedan recuperar allí todos los ejemplos que habían conseguido elaborar en estos años atrás.

3 Introducción

La NDS representa una imagen en sus pantallas en formato ***bitmap*** **o** ***raster*** (de mapa de bits*)*, esto es, que internamente se representa como una matriz de puntos o píxeles. Cada uno de estos puntos es capaz de almacenar la información de color en un espacio de representación de color. Este es, generalmente, el RGB, que utiliza tres valores: uno para el rojo, otro para el verde y otro para el azul. Son los colores primarios de este espacio de color aditivo. La Figura 1a muestra una imagen[2] en formato de mapa de bits, donde se puede ver el detalle de los valores RGB de tres píxeles de la misma: los valores de blanco tienen altas las tres componentes, los oscuros en cambio los tienen todos muy bajos y el resto de colores se forman como combinación de los primarios, p. ej., el amarillo como rojo (R) y verde (G).



a) *b)*

Figura 1: Imagen en mapa bits: (a) ejemplo y (b) comparativa gráfica de formatos raster y vectorial. Fuente: en nota a pie de página.

El *raster* no es el único formato para especificar una imagen, en la Figura 1b[3], podemos ver una imagen en la que la mitad de un motivo simétrico se representa con la técnica de *raster* (a la izquierda) y con gráficos vectoriales (a la derecha). Vamos a definir estas dos técnicas brevemente, para destacar los puntos fuertes de ambas dos cuestiones de cada

- La imagen en *raster* está hecha con una rejilla de puntos, a los que se les asigna un color.

---

[2] Para ampliar la definición, se puede acceder a la página de la Wikipedia <https://en.wikipedia.org/wiki/Raster_graphics> de donde se ha sacado esta imagen.

[3] Imagen de <https://www.behance.net/gallery/42717355/Super-Mario-themed-Infographic-Raster-vs-Vector-Image>.

    - Pierde calidad al ampliarse la imagen, ya que la rejilla tiene una resolución máxima predefinida. Se utiliza mayormente en fotografía y en la implementación de la memoria de vídeo y las tarjetas en los dispositivos con pantalla de píxeles. Por ello, el contenido de una imagen se traslada directamente a memoria y se puede modificar punto a punto.
- La imagen en vectorial está construida con la ejecución de operaciones de pintado de formas geométricas.
    - Por ello su precisión (calidad) en el pintado se mantiene con el cambio de escala. Su uso está enfocado al diseño, la ilustración y la impresión sobre textil, papel o en 3D. Los ficheros suelen ser de menor tamaño. Si el dibujo no se representa directamente con las funciones de pintado, el proceso de pintado puede un tiempo apreciable en realizarse.

El *hardware* de la NDS está compuesto por varios componentes: el audio, la botonera, las pantallas, la memoria, los motores gráficos o de vídeo, el *WiFi*, etc. Si nos diera por desmontar la consola, no lo recomiendo, veríamos que estos "chips" están integrados en un gran circuito integrado, no se ven como elementos discretos. La memoria de vídeo contiene un área que se denomina ***framebuffer***, este área de la RAM mantiene el mapa de bits que se muestra en las pantallas. El componente de vídeo genera, a partir de esa información, la señal que se visualiza en las pantallas.

Un *framebuffer* se puede describir como un mapa directo de valores de memoria a colores en pantalla. Simplemente escribiendo el valor de color correcto en la memoria podemos establecer un píxel como mejor nos parezca. La memoria de *framebuffer* se puede describir completamente por tres cosas: la dirección en la que comienza, el formato de color de los píxeles y el número de píxeles por línea horizontal. La memoria para un *framebuffer* es un mapa lineal único, de modo que las primeras entradas W corresponden a la fila superior de píxeles en la pantalla (W en este caso es el "ancho" del *buffer*). La siguiente fila ocupa las entradas W a 2 * W - 1.

Veamos un primer ejemplo básico para ver cómo se accede a la memoria de vídeo y que sirva para algo. Vamos a hacer una linterna, eso siempre viene bien.

3.1 Un primer ejemplo

La memoria de vídeo está mapeada dentro de la memoria principal de la NDS. Esto significa que si escribimos un valor a partir de una cierta dirección o posición de memoria, algo aparecerá en pantalla. Veamos qué es eso que debemos escribir y que ya existe una definición para esa zona de memoria, así que no tendremos que memorizar esa dirección.

Veamos cómo se puede hacer siguiendo el ejemplo de código de [1]. En la línea 7 se establece la pantalla inferior en modo de texto y le asigna uno de los dos motores gráficos de la NDS, esto nos permitiría sacar mensajes con la instrucción *printf* o sus variantes como si fuera una terminal del computador de escritorio.

```
1       #include <nds.h>
2       #include <stdio.h>
3
4       int main( void )  {
5        int i
6
7        consoleDemoInit();                  //initialize the DS Dos-like functionality
8        videoSetMode(MODE_FB0); //set frame buffer mode 0., enable VRAM A
9        // for writing by the cpu and use  as a framebuffer by video hardware
10       vramSetBankA(VRAM_A_LCD);
11       while(1) {
12          u16 color = RGB15(31,0,0);                          //red
13          scanKeys();
14          int held = keysHeld();
15          if(held & KEY_A) color = RGB15(0,31,0);             //green
16          if (held &amp; KEY_X) color = RGB15(0,0,31);
17
18          swiWaitForVBlank();
19          for(i = 0; i &lt; 256*192; i++) {                   //fill video
20            VRAM_A[i] = color;
21          }
22       return 0;
23      }
```

Figura 2: Ejemplo básico de acceso al área de memoria de framebuffer. Código extraído de [1].

Para utilizar el modo *framebuffer* en la otra pantalla NDS, la línea 8 establece el modo gráfico que permite el acceso directo a los puntos de cada pantalla y la línea 10 configura a qué región de la memoria corresponde.

El resto del programa es un bucle (líneas 11 a la 21) sin fin, que inicializa una variable con los valores RGB del color rojo (línea 12) y que comprueba si se ha pulsado una tecla (línea 13), en cuyo caso se cambia el color asignado.

En cualquiera de los casos y una vez terminado la generación de la señal de vídeo que corresponde a las pantallas (línea 18), se utiliza el color asignado para dar valores a las 256*192 posiciones de un vector *VRAM_A* (líneas 12 a la 21). Ese vector *VRAM_A* es el vector que comienza en la posición de la memoria de vídeo que corresponde al banco A (*BankA*); sin necesitar saber qué valor hexadecimal tiene y facilitando la lectura del código. Ah, por cierto 256x192 (ancho x alto) es la resolución, en píxeles, de las pantallas de la NDS.

Podemos ver la salida del ejemplo de código de [1] en la Figura 3. Dejaremos a la curiosidad del lector proponer las modificaciones para que la pantalla se muestre en color azul.

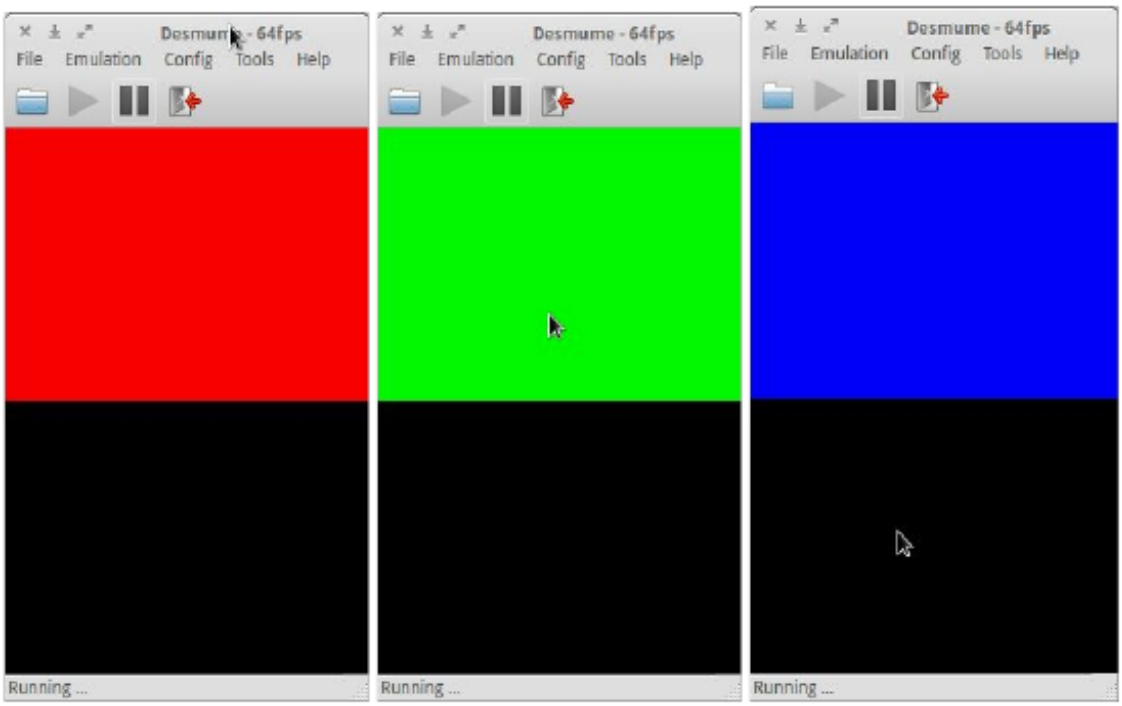


Figura 3: Salida del código de la Figura 2 "Frame buffer".

3.2 Especificando el color en la NDS

A la hora de poner color a un píxel en pantalla en la NDS existen dos formas de representar el color: utilizando una paleta de colores y de forma directa.

El color directo significa que un valor (codificado con 16 bits) controla directamente la intensidad del rojo, verde y azul que se asigna al píxel. Técnicamente, la NDS utiliza dos formatos de color directos, la variación entre los dos es mínima. La primera forma es que el color directo usa 5 bits para representar el brillo que puede tener cada componente de color. Nos referimos a este formato como 555 o, a veces, color de 15 bits. Estos 5 bits equivalen a 2^5=32 niveles de intensidad para cada uno de los tres colores. Para describir un color en este formato, necesitamos combinar nuestros valores deseados de rojo, verde y azul. para formar un número de 15 bits. Los componentes de color se almacenan de la siguiente manera: BBBBBGGGGGRRRRR. Esto se puede traducir como los 5 bits menos significativos contienen el componente rojo, los siguientes 5 bits contienen el verde y los 5 restantes el azul. Nos referimos a esto como formato BGR.

El otro formato de color directo es casi idéntico al que acabamos de comentar. La única diferencia es en el bit más significativo (representado como la 'x' en xBBBBBGGGGGRRRRR). Cuando se utiliza este formato este bit se conoce como el bit 'alfa' y cuando se establece en 0 evitará que el color (expresado en BGR) aparezca en ese punto de la pantalla. La mayoría de las operaciones de gráficos de 16 bits en la NDS utilizan este bit "alfa" para determinar la transparencia del píxel renderizado.

Aunque los formatos de color directo proporcionan una amplia gama de colores, tienen un serio inconveniente: ocupan 16 bits por cada píxel. Aunque la NDS tiene una mayor cantidad de memoria y potencia de los procesadores (en comparación con los sistemas existentes cuando

apareció la NDS), todavía palidece en comparación con la mayoría de las consolas y dispositivos portables modernos. Se sorprenderá de lo rápido que llenará la memoria con una imagen de 16 bits y de como carga el sistema (*hardware*) si intenta cargar imágenes de profundidades de color superiores a 16 bits.

Para aliviar esto, la NDS utiliza diferentes técnicas para ahorrar espacio. El más frecuente es el uso de paletas de color (o tablas de color). En lugar de especificar componentes de color directamente, construimos una tabla de colores y especificaremos un índice en esta tabla. Digamos que tenemos una tabla de 256 colores que contiene 256 valores de color directo. Luego podemos establecer píxeles en pantalla para estos valores especificando un índice. Como la tabla es pequeña, solo necesitaríamos un índice de 8 bits para describir el color del píxel; ¡lo que supone un ahorro del 50%! La NDS admite [1] paletas indexadas de 8 y 4 bits.

4 Ejemplo de uso del *framebuffer:* "Pixels And things"

Para colocar con precisión los píxeles en pantalla, debemos especificar la ubicación. Esto lo haremos usando un sistema de coordenadas cartesianas con el centro de coordenadas, esto es el punto (0,0), en la esquina superior izquierda de la pantalla. La distancia desde el lado izquierdo de la pantalla generalmente se conoce como la coordenada X del píxel y la distancia desde la parte superior es la coordenada Y.

Para determinar el desplazamiento correspondiente en la memoria de *framebuffer*, haremos un cálculo basado en las coordenadas *X* e *Y* que deseamos pintar. Puesto que la memoria está dispuesta linealmente, para llegar a una fila, multiplicamos el número de píxeles en una línea por el valor de la coordenada *Y*. Luego agregamos el valor de *X* y obtendremos la dirección de memoria que buscábamos

El ejemplo de código que muestra la Figura 4 utiliza el concepto de *framebuffer* para generar una animación de un cielo estrellado donde se iluminan los píxeles para representar las estrellas. Así que tendrá que acercarse para verlas. Además cada punto hará su camino a través de la pantalla a una velocidad aleatoria. Cuando llegue al final lo volveremos a mover al principio con un valor de altura y velocidad aleatorios. De estar forma se obtiene un efecto de campo de estrellas en movimiento. (un buen *Star-Trekie* sabrá apreciarlo 😀).

Comenzamos con una estructura de datos (en las líneas 4 a la 9) para definir un estrella, agrupando las coordenadas *X* e *Y*, la velocidad, y el color de cada una. Con ella se define un vector de *NUM_STARS* elementos, que será el número máximo de estrellas que mostrará el ejemplo.

```
1       #include <nds.h>
2       #include <stdlib.h>
3       #define NUM_STARS 40
4       typedef struct  {
5               int x;
6               int y;
7               int speed;
8               unsigned short color;
9       }Star;
10      Star stars[NUM_STARS];
11      void MoveStar(Star* star) {
12              star->x += star->speed;
13              if(star->x >= SCREEN_WIDTH)       {
14                      star->color = RGB15(31,31,31);
15                      star->x = 0;
16                      star->y = rand() % 192;
17                      star->speed = rand() % 4 + 1;
18              }
19      }
20      void ClearScreen(void) {
21          int i;
22          for(i = 0; i < 256 * 192; i++)  VRAM_A[i] = RGB15(0,0,0);
23      }
24      void InitStars(void) {
25              int i;
26              for(i = 0; i < NUM_STARS; i++)      {
27                      stars[i].color = RGB15(31,31,31);
28                      stars[i].x = rand() % 256;
29                      stars[i].y = rand() % 192;
30                      stars[i].speed = rand() % 4 + 1;
31              }
32      }
33      void DrawStar(Star* star) {
34              VRAM_A[star->x + star->y * SCREEN_WIDTH] = star->color;
35      }
36      void EraseStar(Star* star) {
37              VRAM_A[star->x + star->y * SCREEN_WIDTH] = RGB15(0,0,0);
38      }
39      int main( void )  {
40              int i;
41              videoSetMode(MODE_FB0);
42              vramSetBankA(VRAM_A_LCD);
43              ClearScreen();
44              InitStars();
45              while(1)        {               //we like infinite loops in console dev!
46                      swiWaitForVBlank();
47
48                      for(i = 0; i < NUM_STARS; i++)      {
49                              EraseStar(&stars[i]);
50                              MoveStar(&stars[i]);
51                              DrawStar(&stars[i]);
52                      }
53              }
54              return 0;
55      }
```

Figura 4: Day 3 Pixels and Things. Código extraído de [1].

Ahora, veamos que el programa principal (a partir de la línea 39) define el modo de vídeo de una de las pantallas en modo *framebuffer* y, primero que nada, borrará los píxeles (línea 43). En otras palabras, nos estamos asegurando de comenzar con una pantalla en negro.

La línea 44 inicializa todas las estrellas, estableciendo el color a blanco, la velocidad a un valor aleatorio entre 1 y 4, así como las X e Y a valores aleatorios en la pantalla. Observe el uso del operando '%' para recortar los valores aleatorios al rango de 0 al número de columnas o filas, para la X e Y respectivamente.

A partir de ahí el bucle sin fin (líneas 45 a la 53), será el encargado de mover, dibujar y borrar cada estrella. Para borrar ha de asignar a la posición actual de la estrella en pantalla (y por asociación en la dirección correspondiente del *framebuffer*) a color de fondo (negro en nuestro caso).

Ahora, podemos calcular la nueva ubicación de la estrella a donde se moverá. Para ello, solo agregamos su velocidad a su posición *x* actual (solo se mueven en el eje horizontal en este caso). Será necesario verificar si la estrella se ha salido de la pantalla. Para hacer eso, comparamos su ubicación x con el ancho de la pantalla. Si es mayor, sabemos que estamos fuera de la pantalla y procederemos a reubicarla en la primera columna, con otra velocidad y un valor Y aleatorios, haciendo que parezca que se ha encendido una nueva estrella en la pantalla.

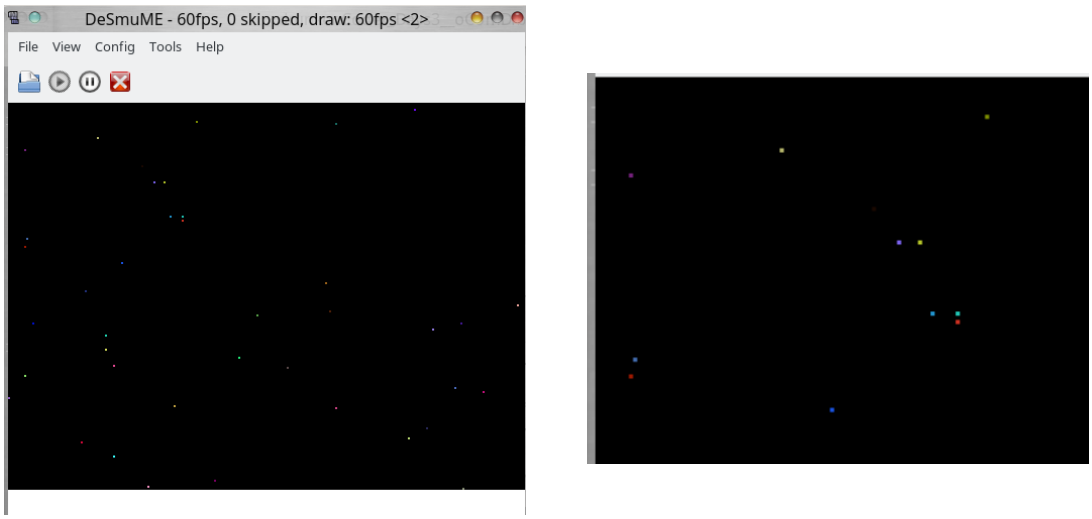
El código de la Figura 4 genera una salida como la que vemos en la Figura 5, en la cual se han recortado la pantalla inferior, puesto que está en blanco y lo que muestran no aporta nada.

a) *b)*

Figura 5: Resultado de ejecutar el ejemplo de la Figura 4 “Pixels And Things”: (a) pantalla superior y (b) ampliación de la parte superior izquierda para ver las estrellas.

5 Conclusiones y cierre

Hemos recorrido el camino: hemos explorado el modo de acceso directo a la memoria de vídeo de la consola NDS y hemos explorado dos ejemplos que nos permiten modificar los valores de cada punto de la pantalla. El trabajo es más laborioso cuanto más complejos sean los objetos a mostrar, pero todo es ponerse. Los computadores actualmente, en todas sus versiones utilizan este modo de trabajo por defecto. La NDS introdujo este modo en un equipo con unas capacidades gráficas y de recursos muy limitados, lo cual es mérito de sus diseñadores.



a) *b)*

Figura 6: Resultado de ejecutar el ejemplo stars_tsw: (a) pantalla superior y (b) ampliación de la parte superior izquierda para ver las estrellas.

Quiero animar al lector a modificar el ejemplo mostrado para que se puedan tener estrellas de colores aleatorios y que la dirección del movimiento también se pueda cambiar, al estilo de la Figura 6. El ejemplo *stars_tsw* que acompaña al repositorio en GitHub [2] de este artículo[4] es una adaptación del ejemplo que hemos visto: se puede modificar la dirección de movimiento de las estrellas con las flechas del cursor y el color de las estrellas no es siempre blanco.

6 Bibliografía

[1] DOVOTO. (2012). Day 3. Ubicado originalmente en <http://www.drunkencoders.com>, no está disponible actualmente.

[2] M. Agustí, (2020). Repositorio del proyecto "NDS-homebrew-development". Disponible en <https://github.com/magusti/NDS-hombrew-development>.

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

[4] Este ejemplo estaba en la red, en 2012, en la URL <http://www.theskyway.net/en/nds-dev3/>.

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

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

[6] .GBATEK. Gameboy Advance / Nintendo DS - Technical Info. Disponible en <https://www.akkit.org/info/gbatek.htm>.

[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.