

Capítulo 2

Trabajando con textos

2.0.1. Textos básicos

En el ejemplo anterior vimos como escribir texto en la pantalla superior, profundicemos un poco más en este tema. Ya hemos visto que necesitamos utilizar `PA.InitText(pantalla,fondo)`; para poder usar el texto, en una o dos pantallas simultáneamente; para el *fondo* debemos de elegir uno de los cuatro disponibles, el 0 es el más cercano y por tanto visible sobre el resto y el 3 es el más profundo de los cuatro. La función `PA.OutputSimpleText(pantalla,tile_x,tile_y,"cadena_texto")`, dispone de 4 parámetros. El primero es el número de pantalla, 1 para la pantalla superior y 0 para la inferior (touchscreen); `tile_x` y `tile_y` son las coordenadas donde situar el texto, no están medidas en píxeles si no en *tiles*, que es un cuadrado formado por píxeles (concretamente 8x8 píxeles), en este caso hay 32 (desde el 0 al 31) *tiles* horizontales y 24 (del 0 al 23) verticales en cada pantalla, en vez de píxeles nos movemos por unos pequeños "bloques" llamados *tiles*.

`PA.OutputSimpleText` puede ser bastante útil, en cambio es una función bastante primitiva y no es muy cómodo trabajar con ella. Sólo imprime cadenas de texto, lo que no nos permitiría por ejemplo mostrar la puntuación. Existe otra función, `PA.OutputText(pantalla,tile_x,tile_y,"cadena_texto",param1,...paramN)`, muy similar a la ya comentada pero podemos pasarles parámetros para que los muestre. Por ejemplo `PA.OutputText(1,0,0,"Puntos : %d",puntos)`, donde `int puntos=123`; es una variable entera y el ejemplo imprimiría : "Puntos : 123". Veamos un par de ejemplos más para terminar de entender el funcionamiento.

Recorte de Cap02/cap02ej01/source/main.c

```

11     int entero = 123456;
12     int dia = 15;
13     int mes = 8;
14     int año = 2007;
15     float decimal = 987.654321;
16     char cadena[30] = "Esto es otra cadena de texto";
17
18     PA_OutputText(1,0,0,"Una simple cadena de texto");
19     PA_OutputText(1,0,1,"Esto es un entero : %d",entero);
20     PA_OutputText(1,0,2,"Un decimal : %f6",decimal);
21     PA_OutputText(1,0,3,"Texto : %s",cadena);
22     PA_OutputText(1,0,4,"Fecha : %d/%d/%d",dia,mes,año);
23     PA_OutputText(1,0,5,"Operaciones : %d+%f3",entero,
24                   decimal);
24     PA_OutputText(1,0,6,"= %f5",(entero+decimal));

```

¿Qué aprendemos de este ejemplo? Vemos que podemos mostrar cadenas con variables de una forma muy sencilla, incluyendo %d para imprimir un entero, %fX para un decimal donde la X es el número de decimales después del punto que queremos mostrar y %s para imprimir cadenas de texto independientemente de su longitud. Al final de la función debemos de pasarle como parámetros las variables que queramos mostrar en el orden en el que aparecen en la cadena formateada. PA_OutputText(1,0,5,"Operaciones : %d+%f3",decimal, entero) esta cadena, aunque no da error si que produce resultados no deseados (convierte el entero a flotante, pero el flotante no lo puede convertir a entero). Además hemos de tener cuidado con estas funciones. Si el texto es demasiado largo y sobre pasa la pantalla, seguirá escribiendo en la línea siguiente por lo que hemos de tener cuidado y controlar la longitud de las cadenas o de otra forma obtendremos resultados no deseados al mezclarse dos textos diferentes en pantalla.

2.1. Borrar cadenas

Hasta ahora sólo hemos escrito una vez en pantalla y no hemos ido más lejos, por lo general los textos son dinámicos y cambian constantemente, por ejemplo la puntuación o el número de vidas. Ya hemos visto la posibilidad de escribir variables en pantalla, lo que lo hace más dinámico, pero cada vez que queramos actualizar esos datos hemos de invocar a la función PA_OutputText(pantalla,tile_x ,tile_y , "cadena_texto",param1,...paramN), lo específico porque hay lenguajes en los que no es necesario, pero no es el caso. Ahora os cuento un pequeño secreto que cómo funciona realmente esta función, lo que hace no es más ni menos que poner en pantalla pequeños gráficos que contienen las letras pero no se preocupa de borrar lo anterior. Por ejemplo si escribimos

```

PA_OutputText(1,0,5,"Esto es una cadena larga");
PA_OutputText(1,0,5,"Esto es otra");

```

El resultado final sería "Esto es otracadena larga" (nótese que se escriben ambas cadenas en las mismas coordenadas). Para evitarlo tenemos que escribir una cadena de "espacios vacíos" entre las dos, PA_OutputText(1,0,5,"");. Como

el segundo ejemplo del capítulos :

```

Recorte de Cap02/cap02ej02/source/main.c
13     int entero = 123456;
14
15     //Ejemplos fallidos
16     PA_OutputText(1,0,0,"Esto es un ejemplo fallido");
17     PA_OutputText(1,0,0,"PA_OutputText no borra");
18     PA_OutputText(1,0,1,"con variables tambien : %d",
19         entero);
20     entero = 99;
21     PA_OutputText(1,0,1,"con variables tambien : %d",
22         entero);
23
24     //Ejemplos funcionales
25     PA_OutputText(1,0,5,"Esto es un ejemplo fallido");
26     PA_OutputText(1,0,5,"
27     PA_OutputText(1,0,5,"Hay que hacerlo manualmente");
28     PA_OutputText(1,0,6,"con variables tambien : %d",
29         entero);
30     entero = 99;
31     PA_OutputText(1,0,6,"
32     PA_OutputText(1,0,6,"con variables tambien : %d",
33         entero);

```

2.2. Texto en Cajas

Desde mi punto de vista, las opciones que hemos comentado son las más utilizadas, pero puede suceder que queramos encerrar el texto dentro de una caja para por ejemplo mostrar conversaciones entre personajes. Para ello utilizamos `PA_BoxText(pantalla,inicio_x,inicio_y,fin_x,fin_y,texto,numero_de_caracteres)`, las unidades *inicio_x*, *inicio_y*, *fin_x*, *fin_y* vienen dadas en los famosos tiles, texto puede ser una cadena de texto literal o una variable de tipo `string`¹).

```

Recorte de Cap02/cap02ej03/source/main.c
14     PA_BoxText(1,6,6,24,18,"Esto es un texto dentro de un
15         rectangulo limitado. Bastante util para mostrar
16         dialogos en RPGs",(12*18));
17     char cadena[100] = "Esto es un texto dentro de un
18         rectangulo limitado. Bastante util para mostrar
19         dialogos en RPGs";
20     PA_BoxText(0,6,6,24,18,cadena,(12*18));

```

Finalmente *numero_de_caracteres* denota el número de letras que se imprimirán en la caja de texto, independientemente de la longitud de la cadena (si el número de caracteres es mayor que la longitud de la cadena no se imprimiran más). Quizás estén pensando para qué sirve limitar el número de caracteres a imprimir, pues facilita mucho la tarea de crear ese efecto tan *COOL* que nos gus-

¹Aunque `string` no es un tipo de variable nativo de C/C++, hablaremos de `string` para referirnos a un array/vector del tipo `char`

ta de los RPG's, de que el texto vaya apareciendo poco a poco, simplemente aumentando en 1 el valor de la variable *número_de_caracteres*, pueden ver un ejemplo de esto en (*Cap02/cap02ej04/source/main.c*).

2.3. Colores

Tenemos dos opciones para colorear el texto, cada una tiene sus ventajas e inconvenientes. La primera quizás sea la más sencilla pero también la que más útil nos pueda resultar, `PA_SetTextCol(pantalla, color_rojo, color_verde, color_azul)`, donde las variables *color_XXX* toman valores desde 0 hasta 31. La típica composición de color a partir de los tres básicos. El problema de esta función es que como se puede observar no se ha especificado el texto que queremos colorear, porque cambia el color de manera global, un color diferente por pantalla. Todo texto escrito con anterioridad o de forma posterior al uso de esta función obtendrá el color que le indiquemos. Lo pueden ver en los ejemplos 5 y 5b de este capítulo (*cap02ej05* y *cap02ej05b*).

La segunda opción nos permite algo más de libertad a la hora de colorear diferentes caracteres o palabras, pero en cambio nos limita los colores. Con `PA_SetTextTileCol(pantalla, color)` disponemos de 10 opciones, la variable *color* varía de 0 (blanco) a 9 (negro), predefinidas para cambiar el color del texto cuando queramos. Lo que hace es cambiar el color actual con el que vamos a escribir, es como cambiar de bolígrafo, lo escrito anteriormente no se modifica. Además podemos cambiar de color en mitad de una misma cadena, con cierto forma. Más claro se ve con un ejemplo :

Recorte de *Cap02/cap02ej06/source/main.c*

```

11     u8 k;
12     for (k = 0; k < 10; k++)
13     {
14         PA_SetTextTileCol(1, k);
15         PA_OutputSimpleText(1, 2, 2+k, "Mas colores
16             que Benetton!");
17     }
18     int entero = 123;
19     PA_OutputText(1, 0, 14, "%c1Incluso %c2en %c3una %
20         c4misma %c5f%c6r%c7a%c8s%c9e %c3%d", entero);

```

Recomiendo al lector que observe el resultado final de esas líneas de código para terminar de asimilar el ejemplo, la aplicación escribirá 10 veces la cadena *Mas colores que Benetton!* con los 10 colores disponibles una debajo de la otra; y luego imprimirá la cadena *Incluso en una misma frase 123* en diversos colores. Como se puede observar se usa `%cX` para cambiar el color en mitad de una cadena, a partir de ese elemento todo lo que se escriba tendrá el color indicado.

2.4. Usando Fuentes Personalizadas

La fuente predefinida no es mala, pero quizás queramos personalizarla, por ejemplo darle un aspecto más futurista medieval con el propósito de adaptarla a la estética general del juego. Tenemos una función que cambiará la fuente que vayamos a utilizar `PA_InitCustomText(pantalla,fondo,nueva_fuente)`, *pantalla* y *fondo* deben de ser los mismo que hayamos iniciado anteriormente con `PA_InitText(pantalla, fondo)`, *nueva_fuente* es el nombre identificativo de nuestra fuente personalizada cargado en memoria (Ejemplo *Cap02/cap02ej07*). ¡**CUIDADO!** Tenemos que iniciar la fuente personalizada antes de usarla ya que borra todo el texto anteriormente escrito.

Ahora tenemos que aprender a añadir esas fuentes personalizadas a nuestro proyecto, el proceso será el mismo que para añadir fondos y sprites, y al principio puede resultar algo complicado. Nos situamos en la carpeta donde instalamos DevKitPro, nos dirigimos a *PALib/Tools/* y copiamos la carpeta *PAGfx* o *PAGfx - Linux* según corresponda, aunque la versión de Linux habrá que compilarla e instalar. La pegamos dentro de la carpeta *source* de nuestro proyecto y la renombramos a *gfx*, debe de quedar algo así : *DevKitPro/nuestro_proyecto/source/gfx/*. En esa misma carpeta copiamos el archivo *Cap02/cap02ej07/source/gfx/nuevafuente.gif*, que será nuestra fuente personalizada. Ejecutamos **PAGC Frontend**, aunque también podemos hacerlo por la línea de comandos así resulta más sencillo, nos dirigimos a la pestaña de *Backgrounds*, **Add File**, seleccionamos nuestro archivo, finalmente pulsamos sobre **Save and Convert** y cuando termine el prompt cerramos. Esta aplicación lo que hace es convertir las imágenes y fondos en archivos aptos para ser cargados por nuestra aplicación, pero aún no están cargados. Para hacerlo simplemente hemos de escribir las siguientes dos líneas al principio de nuestro código (o debajo del include de PALib) :

```
#include "gfx/all_gfx.h"
#include "gfx/all_gfx.c"
```

Estas líneas se encargan de cargar todos los archivos exportados con PAGfx. Hay algunos aspectos de este programa que no hemos comentado pero que ya lo haremos en el apartado de sprites y fondos, que será más extenso. Sobre el formato de la nueva fuente, hemos de decir que es un mapa de tiles, es decir, cada letra ha de ser una imagen de 8x8 píxeles, espacio más que suficiente, y han de estar en el orden dado por ese archivo. Para editar la imagen lo mejor es usar la opción "cuadrícula" o "grid" de algunos editores de imágenes y darle una separación de 8 píxeles tanto horizontal como verticalmente para conocer los límites de cada letra. Con un poco de maña y paciencia podemos tener cuantas fuentes diferentes queramos personalizadas para nuestros juegos y aplicaciones.

2.5. Texto Avanzado

Finalmente y para cerrar este capítulo nos quedan dos tipos de texto, muy similares entre sí, 16c y 8bit. La diferencia más notable es la cantidad de colores entre ellos, mientras que los textos 16c tienen 10 colores (yo tampoco sé de donde sale el 16 entonces) los de 8bits pueden adquirir su color a través de la

paleta definida, teniendo un abanico de 256 colores posibles. Además el de tipo 8bit puede rotarse en ángulos de 90 grados, centrarse o hacerse transparente. La ventaja frente al texto que ya hemos visto, es que dispone de 5 tamaños diferentes, lo que resulta bastante útil.

2.5.1. 16cText

Al igual que en los casos anteriores tenemos que iniciar el fondo para trabajar con este tipo de texto, `PA_Init16cBg`(pantalla, fondo). Luego simplemente lo usamos de la manera `PA_16cText`(pantalla, inicio_x, inicio_y, fin_x, fin_y, texto, color, tamaño, numero_de_caracteres), donde el *color* toma valores de 0 a 9 y *tamaño* de 0 a 4. De igual forma podemos añadir nuestras fuentes personalizadas con la función `PA_16cCustomFont`(slot, fuente), donde *fuente* es el nombre identificativo con la que la hemos cargado (ver la sección de 2.4 para más información sobre como cargar la fuente) y el *slot* es un número entre 0 y 9, contando con que desde el 0 al 4 están ocupados por las predefinidas, aunque se puede sobrescribir. Ejemplos `Cap02/cap02ej08` y `Cap02/cap02ej08b`.

2.5.2. 8bitText

Es similar a todo lo visto anteriormente, el siguiente ejemplo reúne todas las opciones de este tipo de fuentes :

```
Recorte de Cap02/cap02ej09/source/main.c
11 //Como no hay ninguna paleta cargada, agregamos
    algunos colores "a mano"
12 PA_SetBgPalCol(1, 1, PA_RGB(23, 1, 14));
13 PA_SetBgPalCol(1, 2, PA_RGB(31, 0, 0));
14 PA_SetBgPalCol(1, 3, PA_RGB(0, 0, 31));
15 PA_SetBgPalCol(1, 4, PA_RGB(0, 31, 0));
16
17
18 //Debemos iniciar el fondo en 8bit
19 PA_Init8bitBg(1, 3);
20
21 //Varias pruebas de texto
22 PA_SmartText(1, 0, 1, 255, 20, "Hello Mario!", 1, 0,
    1, 100);
23 PA_SmartText(1, 0, 20, 255, 40, "Hello Mario!", 2, 1,
    1, 100);
24 PA_SmartText(1, 0, 40, 255, 60, "Hello Mario!", 3, 2,
    1, 100);
25 PA_SmartText(1, 0, 60, 255, 80, "Hello Mario!", 4, 3,
    1, 100);
26 PA_SmartText(1, 0, 80, 255, 100, "Hello Mario!", 1, 4,
    1, 100);
27
28
29 //Texto rotado 90 grados, OJO! el origen de
    coordenadas tambien rota
30 PA_SmartText(1, 1, 1, 190, 20, "Hello Mario!", 3, 3,
    3, 100);
31
32 //Texto centrado
33 PA_CenterSmartText(1, 0, 100, 255, 120, "Hello Mario!"
    , 2, 4, 1); // Same as Smart text, but no letter
    limit
```

En el ejemplo *Cap02/cap02ej09b* pueden encontrar como utilizar fuentes personalizadas.