

Capítulo 3

¡Click!

Realmente desconozco el ruido con el que identificar a los botones, pero de eso precisamente trataremos en este capítulo, del acceso a los botones y el lápiz de la pantalla táctil, también conocido como *Stylus*.

3.1. Unas breves notas

La Nintendo DS tiene 8 botones, útiles para el juego, no cuento el de encendido ni el control de volumen; y un D-pad, comúnmente llamado "cruceta". Dispone de 4 botones (A,B,X,Y) dispuestos de la misma forma que en la Super Nintendo, es decir, derecha, abajo, arriba e izquierda respectivamente, y situados a la derecha de la pantalla inferior. Tiene dos *trigger's* o disparadores (R,L) también llamados *shoulder's* situados en la parte superior. Finalmente tiene los botones de Start y Select que según el modelo de la consola cambia su disposición, aunque son pequeños y quedan relegados a un segundo plano porque no son cómodos de pulsar, no en mitad del juego. El D-pad o pad direccional será lo que utilizemos principalmente para mover a nuestros personajes por la pantalla. El Stylus o lápiz para la pantalla táctil se guarda dentro de la propia consola, en una ranura que también varía según el modelo. A continuación se muestra la consola en su versión *Lite*, la más actual, y se pueden apreciar todos los botones salvo *L* y *R* que están en la parte superior de la consola y en la imagen quedan



ocultos.

3.2. Los botones y el pad

Ahora que ya sabemos donde está cada cosa podemos empezar a trabajar un poco en el tema. La verdad es que PALib facilita mucho el trabajo otorgándonos tres eventos de estado para los botones y el pad, ¿tres even qué? tres eventos de estado. Es decir, un botón o dirección tiene tres estados posibles : acabo de presionar (*Newpress*), presionado (*Held*) y acabo de soltarlo (*Release*). ¿Cómo saber cuando un botón está en un estado determinado? Es la mar de sencillo, usaremos unas variables predefinidas que nos devolverán 0 en el caso de que no se esté cumpliendo el estado y 1 si es cierto, de la manera *Pad.Estado.Botón*. Por ejemplo, si queremos comprobar si se está pulsando el botón "A" hacemos una llamada `if(Pad.Held.A){ /* acciones */ }`. Los botones disponibles son *A*, *B*, *X*, *Y*, *R*, *L*, *Right*, *Left*, *Up*, *Down*, *Start* y *Select*, espero que nadie tenga problemas para identificar cual le corresponde a cada uno. Hay otro valor más, un botón virtual llamado *Anykey*, que evidentemente se disparará al pulsar cualquier tecla.

Pad.Newpress.Botón toma únicamente el valor 1 si el valor en el frame anterior era 0, es decir, sólo un instante justo cuando lo apretamos. Es muy útil ya que en la práctica se utilizará más que *Held*, por ejemplo para lanzar un disparo, no nos interesa que lance un disparo en cada frame, sólo cada vez que se pulsa.

Pad.Held.Botón toma el valor 1 justo después de que *Newpress* vuelva a cero, siempre y cuando estemos presionando el botón, 0 en caso de que el botón esté libre. Nos será útil por ejemplo para cargar energía, o si tenemos una metralladora. Si lo utilizamos sobre el pad, será perfecto para que nuestro personaje se desplace de una manera suave y no tengamos que pulsar el botón para cada paso.

Pad.Release.Botón toma el valor 1 cuando se suelta un botón, dura un frame y luego vuelve a cero. Útil por ejemplo si estamos cargando energía y queremos lanzar el ataque al soltar el botón.

3.3. El Lápiz (Stylus)

Como ya habíamos dicho, una de los principales atractivos de la consola es su pantalla táctil y las posibilidades que ofrece como dispositivo de entrada. El lápiz, *Stylus*, tiene bastantes métodos y propiedades de las que hacer uso, actúa de forma similar a un botón con los métodos : *Newpress*, *Held* y *Release*, pero además tiene otros como *DbClick* que se activa si se realiza un doble click con el lápiz. O interesantes propiedades como *Vx* y *Vy* para la velocidad de desplazamiento, *Pressure* para saber la presión que se ejerce sobre la pantalla o *Uptime* y *Downtime* que miden el tiempo que ha estado levantado, respectivamente presionado, de la pantalla. No considero que tenga que explicar un ejemplo sobre el funcionamiento básico de los botones y el lápiz, el lector podrá encontrar un ejemplo bastante ilustrativo en *Cap03/cap03ej01*.

3.3.1. Reconocimiento de Escritura

Con el lápiz podemos escribir letras y símbolos que sean reconocidos por el programa, por ejemplo el *Brain Training* utiliza este sistema para conocer nuestras respuestas. Otros juegos como *Castlevania*, *Dawn of Sorrow* lo usan para realizar los *sellos mágicos* para acabar con los enemigos, en *Wario, Master of Disguise*, según lo que dibujemos en la pantalla nuestro personaje adoptará unos poderes u otros. Lamentablemente el sistema implementado no es todo lo bueno que quisiéramos y por eso no trataremos el tema en profundidad con ejemplos.

Sin embargo, si que podemos decir el pequeño truco que utilizan estos juegos y aplicaciones, y es en **la suposición de la respuesta correcta**. Pongamos por ejemplo que estamos realizando sumas o restas, es más o menos evidente que los números 1 y 7 son bastante parecido y se escriben de forma similar. Así por ejemplo si la respuesta correcta es un 1 y escribimos un 7 (no muy marcado) el programa está diseñado para admitirlo como una respuesta correcta. Perdemos “realismo” a cambio de disponer de mucha más jugabilidad y que en definitiva el juego sea más fácil y agradable.

3.3.2. Teclado

El que si suele funcionar bien es el teclado, PALib incorpora un teclado, aunque hay varias implementaciones más o incluso hacer uno propio. Como

ventajas podemos destacar que es fácil de usar y nos ahorra bastante código, además podemos situarlo donde queramos. En contra tenemos que es poco personalizable, aunque si es cierto que podemos cambiar los gráficos del teclado por unos customizados, al tratarse de un mapa compuesto por tiles y haber tantas letras resultará casi imposible adaptar cualquier diseño más o menos elaborado.

```

Recorte de Cap03/cap03ej02/source/main.c
10     PA_InitText(1, 0);
11
12     PA_InitKeyboard(2);
13
14     PA_KeyboardIn(20, 95);
15
16
17     PA_OutputSimpleText(1, 7, 10, "Escribe un texto : ");
18
19     s32 nletter = 0;
20     char letter = 0;
21     char text[200];
22
23     while (1)
24     {
25
26
27         letter = PA_CheckKeyboard();
28
29         if (letter > 31) { // Si hay una nueva letra
30             text[nletter] = letter;
31             nletter++;
32         }
33         else if (letter == PA_TAB){// TAB pulsado
34             u8 i;
35             for (i = 0; i < 4; i++){ // agrega 4
36                 text[nletter] = ' ';
37                 nletter++;
38             }
39
40         }
41         else if ((letter == PA_BACKSPACE)&&nletter) {
42             // Tecla de Retroceso
43             nletter--;
44             text[nletter] = ' '; // Borramos la
45                 ultima letra
46         }
47         else if (letter == '\n'){ // Intro presionado
48             text[nletter] = letter;
49             nletter++;
50
51         }
52
53         PA_OutputSimpleText(1, 8, 11, text); //
54             Escribimos el texto
55         PA_WaitForVBL();
56     }

```

En este pequeño ejemplo podemos ver el funcionamiento estándar del teclado, lo iniciamos, lo mostramos en pantalla y luego tenemos un bucle para leer qué carácter se a pulsado. Con `PA_KeyboardIn(x, y)` haremos aparecer el teclado desde abajo con un movimiento de entrada hasta las coordenadas X e Y, de igual

forma lo puedes sacar de la pantalla con `PA_KeyboardOut()`. Además permite un poco jugar con los colores de las teclas con la función `PA_SetKeyboardColor(color1, color2)`, donde *color1* y *color2* toman los valores 0 (azul), 1 (rojo) y 2 (verde). Si todo esto fuera insuficiente podemos cargar un nuevo teclado con la función `PA_InitCustomKeyboard(fondo, imagenTeclado)`, donde *imagenTeclado* es una imagen que previamente hemos convertido con `PAGfx` en formato de tiles.

¡TRUCO!

¿Quieres tener un teclado mejor y más personalizado? Es sencillo si no te importa perder el estado de *tecla pulsada*. Lo que debemos hacer es simplemente cargar el teclado que viene por defecto, hacer un fondo con el nuevo gráfico y mostrarlo encima del teclado. Las teclas seguirán funcionando igual, pero el gráfico que se mostrará será el nuestro.

3.4. Combinaciones de Teclas

También llamadas “*combos*” y son las que después de una serie de teclas pulsadas en orden nos permiten ejecutar algún evento especial, como los conocidos *ayuken*'s. El tratado a continuación es un ejemplo bastante simplón pero a la vez efectivo, que servirá para las opciones más básicas de esta técnica.

3.4.1. Definición del problema

Necesitamos que se pulsen unas determinadas teclas en un orden específico. Además necesitamos que el sistema sea escalable, es decir que sirva tanto para una sola combinación como para varias especificadas.

3.4.2. Planteamiento de la solución

Lo primero que debemos pensar es cómo definir una combinación de teclas y cómo almacenar varias en nuestra aplicación. La respuesta natural parece, sin duda, es utilizar un vector, también conocidos como array o matrices unidimensionales. No es más que una serie de datos, una lista de valores (espero que nadie se confunda con lo de lista). En C no podemos especificar combinaciones del tipo “un cuarto de luna A” o “arriba, abajo, arriba y B”, en cambio si que podemos asignar a cada tecla un valor numérico (`int`) o un carácter (`char`). Obtemos por identificar a cada botón por un número, por ejemplo el 1 para la A, el 2 para la B, el 3 para la X y así con el resto. Podríamos ir más allá y añadir algunos valores, que más tarde necesitaremos, como la longitud de la combinación o si está activa o no. En ese caso guardaríamos nuestro *combo* en una variable de tipo “*struct*”.

Recorte de Cap03/cap03ej03/source/main.c

```

8 // Estructura de un combo
9 typedef struct{
10     //vector de 15 huecos, es decir, combinaciones de
11     //hasta 15 teclas
12     u8 combo[15];
13     //para no liarla viendo si la siguiente tecla es el
14     //vacío o no
15     u8 length;
16     //controla si esta activo
17     bool active;
18 }combotype;

```

Ya tendríamos una forma de añadir nuestras combinaciones al juego. Ahora hay que crear una función para que compruebe si se ha realizado un combo o no, esta función que llamaremos `checkCombo(u8 c)`, será llamada cada vez que se pulse una tecla.

Recorte de Cap03/cap03ej03/source/main.c

```

54 //lanzamos el evento checkCombo con la tecla
55 //correspondiente
56 if(Pad.Newpress.A) { checkCombo(1); };
57 if(Pad.Newpress.B) { checkCombo(2); };
58 if(Pad.Newpress.X) { checkCombo(3); };
59 if(Pad.Newpress.Y) { checkCombo(4); };
60 if(Pad.Newpress.L) { checkCombo(5); };
61 if(Pad.Newpress.R) { checkCombo(6); };
62 if(Pad.Newpress.Up) { checkCombo(8); };
63 if(Pad.Newpress.Down) { checkCombo(10); };
64 if(Pad.Newpress.Left) { checkCombo(7); };
65 if(Pad.Newpress.Right) { checkCombo(9); };
66 if(Pad.Newpress.Start) { checkCombo(11); };
67 if(Pad.Newpress.Select) { checkCombo(12); };

```

Como se puede comprobar, se utiliza el evento `Newpress` de las teclas y botones para llamar a la función una única vez (justo en el momento en el que se pulsa). Después llamamos a la función `checkCombo(codigDeTecla)`. Para terminar resta hacer la comprobación por todas las existentes combinaciones de teclas, esta parte es más bien técnica aunque la idea general es sencilla :

- Se pulsa la primera tecla.
- Se comprueba una por una las combinaciones existentes que esa tecla coincide con la primera tecla de la combinación.
- Si en alguna combinación falla, ésta se “desactiva”.
- Se presentan dos posibles casos : falló en todas ó coincide con alguna.
- Si falla en todas el sistema se resetea.
- Si acierta en alguna se dan dos casos : se llegó final de la combinación o no.

Recorte de Cap03/cap03ej03/source/main.c

```

88 void checkCombo(u8 c) {
89     pseudoTime = 0;
90     //comprobamos si esta dentro del tiempo aceptado
91     if(pseudoTime<comboTime) {
92         bool doingCombo = false;
93         //vamos uno a uno por los combos definidos
94         for(i = 0;i<numberCombos;i++) {
95             //si la tecla coincide con el orden
96             if (combos[i].combo[comboLength]==c &&
97                 combos[i].active) {
98                 doingCombo = true;
99                 //si es el final del combo
100                if (combos[i].length==
101                    comboLength+1) {
102                    //ejecutamos el combo
103                    i
104                    doCombo(i);
105                };
106            } else {
107                //si la tecla no coincide lo
108                desactivamos
109                //esto es por si tenemos una
110                lista grande de
111                combinaciones posibles
112                //ahorramos recursos
113                combos[i].active = false;
114            };
115        };
116        //si no hace ningun combo reseteamos el
117        sistema
118        if(!doingCombo) {
119            comboLength = 0;
120            for( i = 0;i<numberCombos;i++) {
121                combos[i].active = true;
122            };
123        } else {
124            //si lo estamos haciendo, vamos una
125            posicion siguiente
126            comboLength++;
127        };
128    } else {
129        comboLength = 0;
130    };
131 };

```